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.

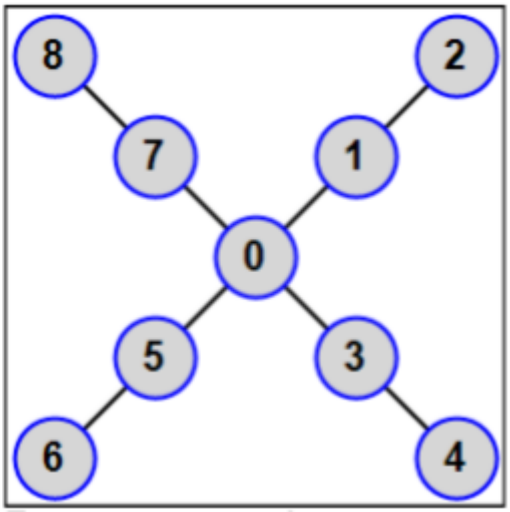
Студент гр. 9381 _____ Птичкин С. А.
Преподаватель _____ Татаринов Ю. С.

Санкт-Петербург
2021

Задание.

Вариант № 11.

Число процессов K является нечетным: K = 2N + 1 (1 < N < 5); в каждом процессе дано целое число A. Используя функцию MPI_Graph_create, определить для всех процессов топологию графа, в которой главный процесс связан ребрами со всеми процессами нечетного ранга (1, 3, …, 2N − 1), а каждый процесс четного положительного ранга R (2, 4, …, 2N) связан ребром с процессом ранга R − 1 (в результате получается N-лучевая звезда, центром которой является главный процесс, а каждый луч состоит из двух подчиненных процессов R и R + 1, причем ближайшим к центру является процесс нечетного ранга R).

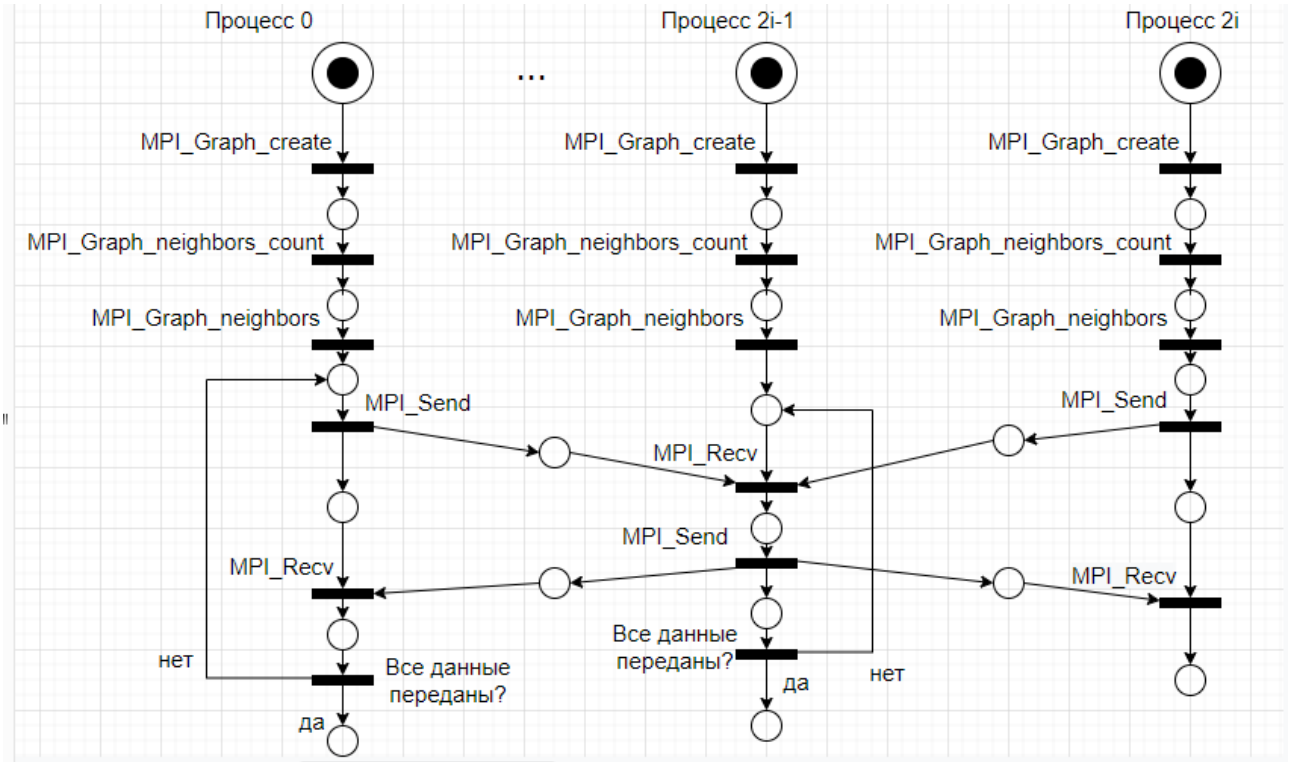


Переслать число A из каждого процесса всем процессам, связанным с ним ребрами (процессам-соседям). Для определения количества процессов-соседей и их рангов использовать функции MPI_Graph_neighbors_count и MPI_Graph_neighbors, пересылку выполнять с помощью функций MPI_Send и MPI_Recv. Во всех процессах вывести полученные числа в порядке возрастания рангов переславших их процессов.

Краткое описание алгоритма.

Сначала для 2N+1 процессов создаётся топология графа в виде N-лучевой звезды. На каждом процессе хранится число A, которое совпадает с рангом этого процесса. Затем в созданной топологии для каждого процесса определяется число соседей и сами соседние процессы. Затем между соседними процессами происходит обмен числами A, которые записываются в массив с длиной, равной количеству соседей, на каждом процессе. Каждый процесс выводит полученные данные в порядке возрастания номеров процессов-соседей.

Формальное описание алгоритма.



Листинг программы.

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

int main(int argc, char* argv[]) {
	int ProcNum, ProcRank;
	MPI_Status Status;
	MPI_Init(&argc, &argv);
	MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
	MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
	int* index = new int[ProcNum];
	int A = ProcRank;
	index[0] = ProcNum/2;
	for (int i = 1; i < ProcNum; i++) { // инициализируем index
		if (i % 2 == 0) {
			index[i] = index[i-1] + 1;
		}
		else {
			index[i] = index[i - 1] + 2;
		}
	}

	int num_edges = ProcNum/2;
	num_edges *= 4;
	int* edges = new int[num_edges];
	// инициализируем edges
	for (int i = 0, j = 1; i < index[0]; i++, j+=2) {
		edges[i] = j;
	}
	for (int i = 1, e = index[0]; i < ProcNum; i++, e++) {
		if (i % 2 == 0) {
			edges[e] = i - 1;
		}
		else {
			edges[e] = 0;
			e++;
			edges[e] = i + 1;
		}
	}
	MPI_Comm GRAPH_COMM;
	//Создаём новый коммуникатор
	MPI_Graph_create(MPI_COMM_WORLD, ProcNum, index, edges, 1,
&GRAPH_COMM);
	int nneighbours;
	MPI_Graph_neighbors_count(GRAPH_COMM, ProcRank, &nneighbours);
    int* neighbors = new int[nneighbours];
	int* recv_data = new int[nneighbours];
	MPI_Graph_neighbors(GRAPH_COMM, ProcRank, nneighbours,
neighbors);

	if (ProcRank == 0) {
		for (int i = 0; i < nneighbours; i++) {
```

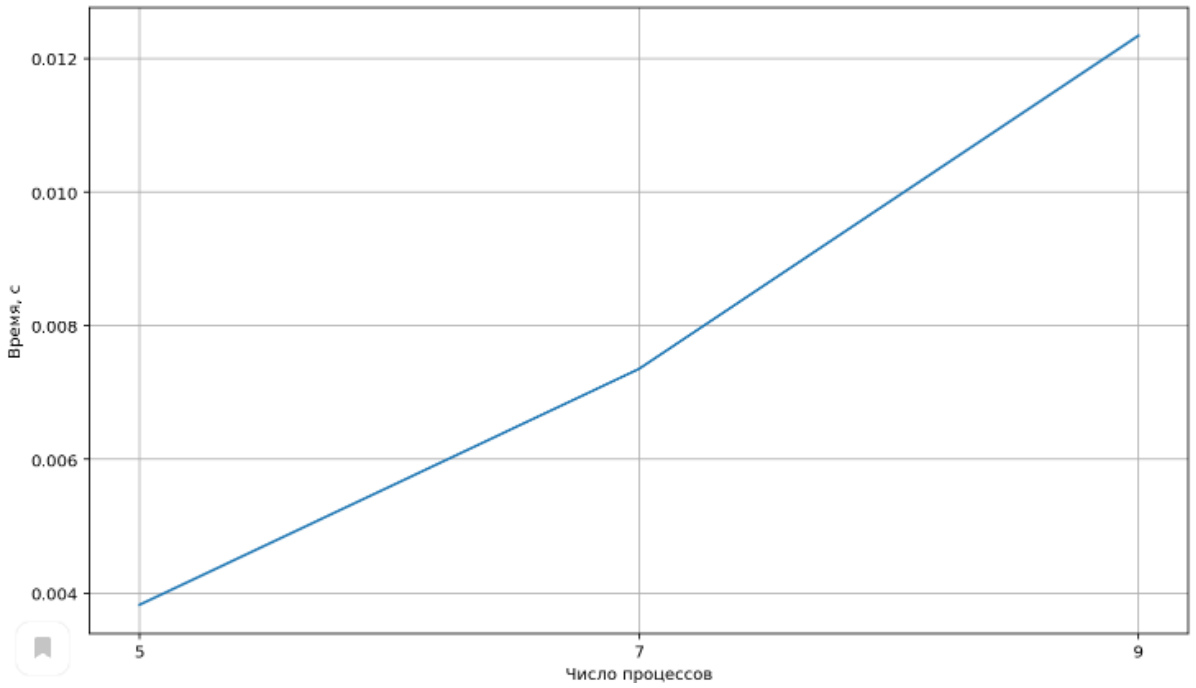
```
                MPI_Send(&A,    1,    MPI_INT,    neighbors[i],    0,
GRAPH_COMM);
                MPI_Recv(recv_data + i, 1, MPI_INT, neighbors[i], 0,
GRAPH_COMM, &Status);
           }
     }
     else if (ProcRank % 2 == 0) {
           for (int i = 0; i < nneighbours; i++) {
                MPI_Send(&A,    1,    MPI_INT,    neighbors[i],    0,
GRAPH_COMM);
                MPI_Recv(recv_data + i, 1, MPI_INT, neighbors[i], 0,
GRAPH_COMM, &Status);
           }
     }
     else {
           for (int i = 0; i < nneighbours; i++) {
                MPI_Recv(recv_data + i, 1, MPI_INT, neighbors[i], 0,
GRAPH_COMM, &Status);
                MPI_Send(&A,    1,    MPI_INT,    neighbors[i],    0,
GRAPH_COMM);
           }
     }

     //Вывод данных
     printf("Process %d: ", ProcRank);
     for (int i = 0; i < nneighbours; i++) {
           printf("%d ", recv_data[i]);
     }
     delete[] index;
     delete[] edges;
     delete[] recv_data;
     delete[] neighbors;
     MPI_Finalize();
     return 0;
}
```

Выполнение работы.

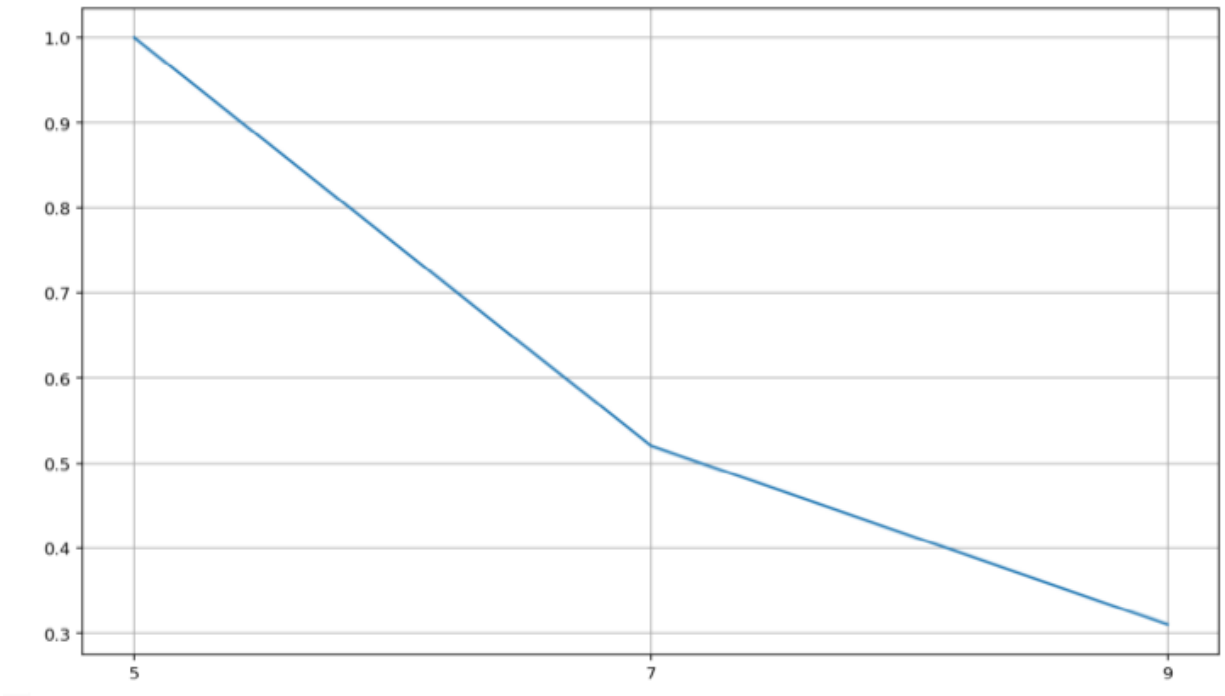
1) Была написана программа выполняющая поставленную задачу. Затем она была протестирована на различном числе процессов:

```
C:\Users\Сергей\Documents\Visual Studio 2015\Projects\par_alg_5\Debug>mpiexec -n 5 par_alg_5.exe
Process 4: 3
Process 0: 1 3
Process 1: 0 2
Process 2: 1
Process 3: 0 4
C:\Users\Сергей\Documents\Visual Studio 2015\Projects\par_alg_5\Debug>mpiexec -n 7 par_alg_5.exe
Process 5: 0 6
Process 0: 1 3 5
Process 6: 5
Process 2: 1
Process 3: 0 4
Process 4: 3
Process 1: 0 2
C:\Users\Сергей\Documents\Visual Studio 2015\Projects\par_alg_5\Debug>mpiexec -n 9 par_alg_5.exe
Process 2: 1
Process 5: 0 6
Process 3: 0 4
Process 1: 0 2
Process 6: 5
Process 8: 7
Process 7: 0 8
Process 0: 1 3 5 7
Process 4: 3
```

2) Был построен график зависимости времени выполнения программы от числа процессов:



3) Был построен график ускорения программы:



Вывод.

Таким образом, была написана программа, создающая виртуальную топологию графа типа N-лучевой звезды и производящая обмен сообщениями между соседними процессами. Соседи определялись с помощью специальных функций для работы с топологиями графа. Также программа была запущена на различном числе процессов. Были построен график зависимости времени выполнения от числа процессов и график ускорения. Даже по таким немногочисленным данным, можно сделать вывод, что с ростом числа процессов, время выполнения программы растёт, а ускорение падает. Это может быть связано с увеличением числа процессов, участвующих в обмене сообщениями с главным процессом. У остальных процессов количество соседей не изменяется.