**x`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. 1384 |  | Усачева Д. В. |
| Преподаватель |  | Татаринов Ю. С. |

Санкт-Петербург

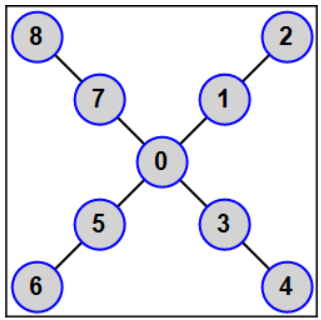
2023

**Цель**

Ознакомиться с виртуальными топологиями, их управлением и функциями в библиотеке MPI. Написать программу с использованием топологии графа.

## 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.

# Выполнение работы

Для выполнения поставленной задачи написана программа на языке C, код которой представлен ниже в листинге 1.

Для выполнения поставленной задачи создаются два массива index (количество исходящих дуг для каждой вершины,) и edges (последовательный список входящих дуг графа (матрица инцидентности)). Для их заполнения была написана функция fill.

Далее при помощи функции MPI\_Graph\_create создается топология графа. Функция MPI\_Graph\_neighbors\_count позволяет нам узнать количество соседних процессов, в которых от проверяемого процесса есть выходящие дуги. Далее создается массив «соседей» необходимой величины. Для его заполнения вызывается функция MPI\_Graph\_neighbors.

Получив списки своих соседей, процесс начинает пересылку целого числа (пересылаем ранг процесса), а после принимает числа от своих соседей в порядке в возрастания рангов и выводит необходимую информацию.

В конце своей работы каждый процесс освобождает созданный коммуникатор при помощи MPI\_Comm\_free.

Ниже представлена сеть Петри основной части алгоритма (см. рис 1).

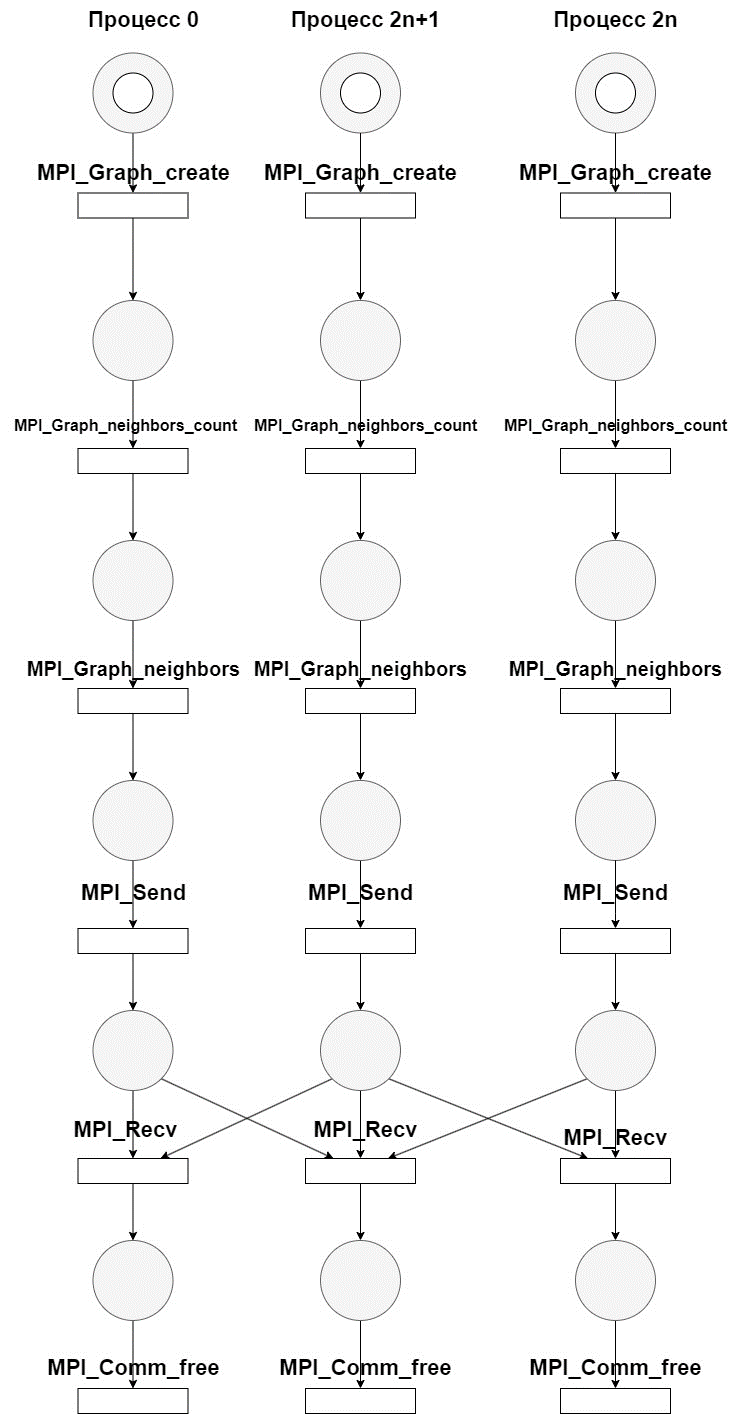


Рисунок 1 — Сеть Петри основной части алгоритма

Листинг 1 — Код программы lab5.c

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <mpi.h>**

**void fill(int\* index,int\* edges, int N) {**

**int id = 0;**

**for (int i = 0; i < N; i++) {**

**if (i == 0)**

**index[i] = (N - 1) / 2;**

**else if(i % 2 == 0)**

**index[i] = index[i - 1] + 1;**

**else {**

**edges[id]=i;**

**id++;**

**index[i] = index[i - 1] + 2;**

**}**

**}**

**int remains = id % 3;**

**while(id < (N - 1) \* 2) {**

**if (id % 3 == remains){**

**edges[id] = 0;**

**id++;**

**}**

**else if(id == (N + 1) / 2){**

**edges[id] = 2;**

**id++;**

**}**

**else if (id % 3 == (remains + 1) % 3 ) {**

**edges[id] = edges[id - 3] +2;**

**id++;**

**}**

**else {**

**edges[id] = edges[id - 1] - 1;**

**id++;**

**}**

**}**

**}**

**int main(int argc, char \*\*argv)**

**{**

**int procNum, procRank;**

**double start;**

**MPI\_Init(&argc, &argv);**

**start = MPI\_Wtime();**

**MPI\_Comm newComm;**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &procRank);**

**MPI\_Comm\_size(MPI\_COMM\_WORLD, &procNum);**

**if (procNum < 5 || procNum > 9 || procNum % 2 == 0){**

**if (procRank == 0)**

**printf("Количество процессов должно быть нечетным числом из [5,9]\n");**

**}**

**else {**

**int count;**

**int A;**

**int index[procNum];**

**int edges[(procNum - 1) \* 2];**

**fill(index, edges, procNum);**

**MPI\_Graph\_create(MPI\_COMM\_WORLD, procNum, index, edges, 0, &newComm);**

**MPI\_Graph\_neighbors\_count(newComm, procRank, &count);**

**int neighbors[count];**

**MPI\_Graph\_neighbors(newComm, procRank, count, neighbors);**

**for (int i = 0; i < count; i++)**

**MPI\_Send(&procRank, 1, MPI\_INT, neighbors[i], 0, newComm);**

**for (int i = 0; i < count; i++)**

**{**

**MPI\_Recv(&A, 1, MPI\_INT, neighbors[i], 0, newComm, MPI\_STATUS\_IGNORE);**

**printf("Процесс %d получил от процесса %d число %d \n", procRank, neighbors[i], A);**

**}**

**MPI\_Comm\_free(&newComm);**

**}**

**printf("Время работы %d процесса: %f\n", procRank, MPI\_Wtime() - start);**

**MPI\_Finalize();**

**return 0;**

**}**

**Ниже представлен вывод программы lab5.с**

**Листинг 2 — Вывод программы lab5.c для 5, 7 и 2 процессов**

**Процесс 0 получил от процесса 1 число 1**

**Процесс 1 получил от процесса 0 число 0**

**Процесс 2 получил от процесса 1 число 1**

**Процесс 3 получил от процесса 0 число 0**

**Процесс 0 получил от процесса 3 число 3**

**Процесс 1 получил от процесса 2 число 2**

**Время работы 2 процесса: 0.002352**

**Процесс 3 получил от процесса 4 число 4**

**Время работы 3 процесса: 0.002527**

**Процесс 4 получил от процесса 3 число 3**

**Время работы 1 процесса: 0.002522**

**Время работы 0 процесса: 0.002509**

**Время работы 4 процесса: 0.002675**

**Процесс 6 получил от процесса 5 число 5**

**Время работы 6 процесса: 0.003135**

**Процесс 5 получил от процесса 0 число 0**

**Процесс 5 получил от процесса 6 число 6**

**Время работы 5 процесса: 0.003419**

**Процесс 4 получил от процесса 3 число 3**

**Время работы 4 процесса: 0.003606**

**Процесс 0 получил от процесса 1 число 1**

**Процесс 0 получил от процесса 3 число 3**

**Процесс 0 получил от процесса 5 число 5**

**Процесс 1 получил от процесса 0 число 0**

**Процесс 2 получил от процесса 1 число 1**

**Время работы 2 процесса: 0.005231**

**Процесс 1 получил от процесса 2 число 2**

**Время работы 1 процесса: 0.005257**

**Процесс 3 получил от процесса 0 число 0**

**Процесс 3 получил от процесса 4 число 4**

**Время работы 3 процесса: 0.005851**

**Время работы 0 процесса: 0.005218**

**Количество процессов должно быть нечетным числом из [5,9]**

**Время работы 1 процесса: 0.000001**

**Время работы 0 процесса: 0.000062**

Так как мы всегда передаем число, нет необходимости прослеживать зависимость времени работы от объема данных. Рассмотрим зависимость времени работы программы от количества процессов.

**Таблица 1 — Среднее время выполнения.**

|  |  |
| --- | --- |
| **Количество процессов** | **Среднее время на выполнение(мс)** |
| **5** | 2.002 |
| **7** | 2.152 |
| **9** | 3.186 |

**Ниже указаны графики зависимостей времени выполнения и ускорения (см. рисунки 2-3).**

**Рисунок 2 — График зависимости времени выполнения от числа процессов**

**Ускорение времени работы программы можно вычислить по формуле:**

Sp (𝑛) = 𝑇1(𝑛)/𝑇p(𝑛)

**Рисунок 3 — График зависимости ускорения от числа процессов**

**Выводы**

В ходе выполнения лабораторной работы были изучены и использованы функция для создания топологии графа MPI\_Graph\_create и функции для получения информации о вершинах соседях MPI\_Graph\_neighbors\_count (количество соседей) и MPI\_Graph\_neighbors (список соседей). Так как в интервал, заданный в условии, попадает всего три нечетных числа, полученные графики не позволяют сделать точный вывод о зависимости времени работы программы от количества процессов. Однако по графикам прослеживается увеличение времени работы программы, это связано с тем, что при большем количестве процессов увеличивается количество соседей нулевого процесса, с которыми он должен обменятся числами. Так же скорость работы программы замедляется из-за увеличения времени заполнения массивов index и edges.