**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №1**

**по дисциплине «Параллельные алгоритмы»**

**Тема**: **Использование функций обмена данными «точка-точка» в библиотеке MPI.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1384 |  | Усачева Д. В. |
| Преподаватель |  | Татаринов Ю. С. |

Санкт-Петербург

2023

**Цель**

Познакомиться с функциями обмена данными «точка-точка» в библиотеке MPI.

## Задание

Вариант №8. Сдвиг массива, распределенного между узлами. Процесс 0 генерирует массив и раздает его другим процессам, после чего выполняется циклический сдвиг массива.

**Выполнение работы**

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

Для выполнения поставленной задачи нулевым процессом генерировался массив в coef раз больше числа процессов (без учета нулевого). Далее массив был распределен нулевым процессом между первым и последним процессами. Каждому процессу были предоставлены индексы элементов, которые он должен обработать. Получив новые индексы, процессы отправляют результат обработки нулевому, который собирает все данные и выводит результирующий массив. Для сдвига индексов массива была использована функция shiftArrIndex.

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

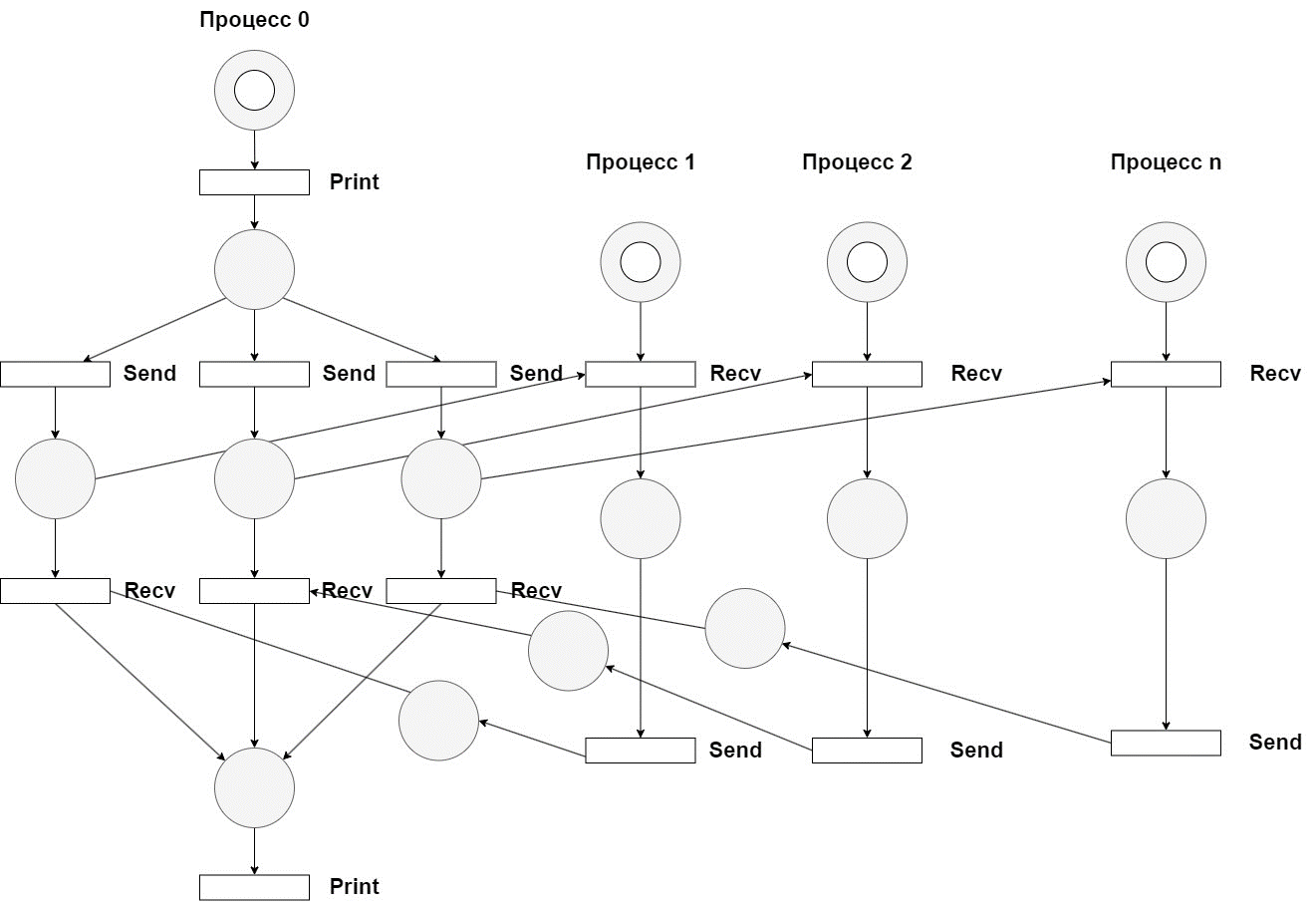


Рисунок 1 — Сеть Петри основной части алгоритма

Листинг 1 — Код программы lab1.c

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <mpi.h>**

**void printArr(int arr[], int lenArr){**

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

**printf("%d ", arr[i]);**

**}**

**printf("\n");**

**}**

**void shiftArrIndex(int\* arr, int lenArr, int coef) {**

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

**arr[i] = (arr[i] + 1) % lenArr;**

**}**

**}**

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

**int procNum, procRank;**

**double start, end;**

**int coef = 15;**

**int\* arr;**

**int\* shiftArr;**

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

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

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

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

**int lenArr = (procNum - 1) \* coef;**

**int\* tmpArr = (int\*)malloc(coef \* sizeof(int));**

**arr = (int\*)malloc(sizeof(int) \* lenArr);**

**shiftArr = (int\*)malloc(sizeof(int) \* lenArr);**

**if (procRank == 0) {**

**arr = (int\*)malloc(lenArr \* sizeof(int));**

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

**arr[i] = i + 1 ;**

**}**

**printf("Start array: ");**

**printArr(arr, lenArr);**

**//процесс 0 раздает массив другим процессам**

**for (int i = 1; i < procNum; i++) {**

**for(int j = 0; j < coef; j++){**

**tmpArr[j]=(i-1)\*coef + j;**

**}**

**MPI\_Send(tmpArr, coef, MPI\_INT, i, 0, MPI\_COMM\_WORLD);**

**}**

**//процесс 0 принимает новые индексы элементов массива**

**for (int i = 1; i < procNum; i++) {**

**MPI\_Recv(tmpArr, coef, MPI\_INT, i, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);**

**for(int j = 0; j < coef; j++){**

**shiftArr[(i-1)\*coef + j]=arr[tmpArr[j]];**

**}**

**}**

**printf("Shift array: ");**

**printArr(shiftArr,lenArr);**

**}**

**//другие процессы принимают свои части массива и делают циклический сдвиг на 1**

**//результат отправляется обратно 0 процессу**

**else {**

**arr = (int\*)malloc(sizeof(int) \* coef);**

**MPI\_Recv(arr, coef, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);**

**shiftArrIndex(arr, lenArr, coef);**

**MPI\_Send(arr, coef, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);**

**}**

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

**MPI\_Finalize();**

**if (procRank == 0){**

**printf("Time: %f",end-start);**

**}**

**return 0;**

**}**

**Ниже представлен вывод программы lab1.c**

**Листинг 2 — Вывод программы lab1.c**

**Start array: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45**

**Shift array: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 1**

**Time: 0.000140**

**Рассмотрим время работы программы в трех различных вариантах:**

Время работы программы зависит от количества процессов и от объема данных. При большем числе процессов увеличивается размер обрабатываемого массива.

**Таблица 1 — Среднее время выполнения.**

|  |  |
| --- | --- |
| **Количество процессов (Объем массива равен числу процессов\*15)** | **Среднее время на выполнение(мс)** |
| **2** | **0.060** |
| **4** | **0.0102** |
| **8** | **0.149** |
| **16** | **14.604** |
| **32** | **66.810** |

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

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

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

Sp (𝑛) = 𝑇1(𝑛)/𝑇p(𝑛)

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

Время работы программы зависит от количества процессов. Объем массива фиксированный и равен 64.

**Таблица 2 — Среднее время выполнения.**

|  |  |
| --- | --- |
| **Количество процессов** | **Среднее время на выполнение(мс)** |
| **3** | **0.148** |
| **5** | **0.227** |
| **9** | **0.316** |
| **17** | **0.439** |
| **33** | **0.822** |

**Ниже указаны графики зависимостей времени выполнения и ускорения (см. рисунки 4-5).**

**Рисунок 4 — График зависимости времени выполнения от числа процессов**

**Рисунок 5 — График зависимости ускорения от числа процессов**

Время работы программы зависит от объема данных. Количество процессов фиксированное и равно 5.

**Таблица 3 — Среднее время выполнения.**

|  |  |
| --- | --- |
| **Объем массива** | **Среднее время на выполнение(мс)** |
| **16** | **0.201** |
| **64** | **0.223** |
| **128** | **0.213** |
| **512** | **0.565** |
| **1024** | **0.554** |

**Ниже указаны графики зависимостей времени выполнения и ускорения (см. рисунки 6-7).**

**Рисунок 6 — График зависимости времени выполнения от объема массива**

**Рисунок 7 — График зависимости ускорения от объема массива**

**Выводы**

В ходе выполнения лабораторной работы были изучен и использованы функции обмена данными библиотеки MPI. После полученных экспериментальных результатов можно сделать вывод о том, что время выполнения увеличивается в любом из трех случаев. Для фиксированного количества процессов ускорение уменьшается медленнее.