**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

|  |  |  |
| --- | --- | --- |
| Студентка гр. 9382 |  | Иерусалимов Н. |
| Преподаватель |  | Татаринов Ю.С. |

Санкт-Петербург

2021

**Цель работы**

Научиться обмену данными между процессами и дальнейшее использование результатов.

**Формулировка задания**

*Суммирование элементов массива:* Процесс 0 генерирует массив и раздает его другим процессам для вычисления локальных сумм, после чего вычисляет общую сумму.

**Выполнение**

Сначала объявляем нужные переменные. Смотрим ранг процесса и в зависимости от него далее действуем. Корневой процесс был выбран нулевым.

Если процесс нулевой то тогда мы спрашиваем у пользователя сколько элементов в массиве он хочет видеть. После чего создается массив с таким количеством элементов которое пользователь указал, числа случайные. Далее мы вычисляем какое количество элементов массива будет передано каждому процессу. Делаем это путем деления общего количества элементов на количество процессов. Т.е если у нас 12 элементов в массиве и задействовано 2 процесса тогда каждому процессу уйдет по 6 элементов для суммирования.  
С помощью цикла for проходимся по всем процессам и назначаем им ссылку от куда начать суммирование и где закончить. С помощью MPI\_Send отправляем эти данные и сами начинаем считать сумму для своих элементов массива. Далее нулевой процесс начинает ждать ответа от других, когда они закончат делать делегированную на них задачу. После чего суммирует их ответы между собой и не забывает про свое вычисление далее отправляет на экран результат.

Если же процесс был не нулевым он уходит в else и начинает ждать сообщения от главного с помощью MPI\_Recv как только приходит сообщение он начинает суммировать элементы массива которые ему были выданы и отправляет результат с помощью MPI\_Send. Весь код программы в Приложение А.

**Тестирование**

Давайте просуммируем 10 элементов массива с помощью 8 процессов и посмотрим на вывод рис. 1

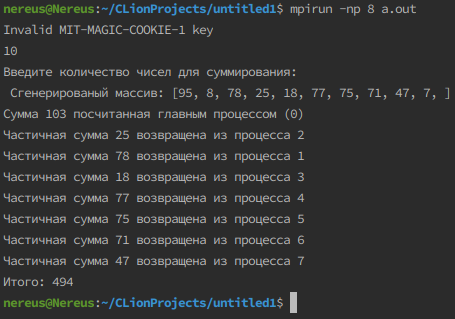


Рис. 1

Тоже самое для двух процессов и 11 элементов рис. 2

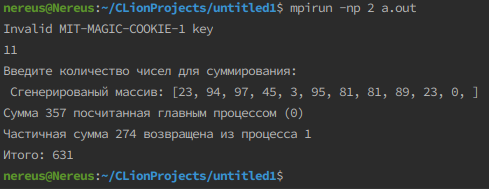


Рис. 1

**Выводы**

В данной лабораторной работе были использованы функции обмена данными «точка-точка» в библиотеке MPI. Была реализована задача по суммированию и равномерному распределению обязанностей по процессам.

**Приложение А.**

Файл - main.cpp

#include <stdio.h>  
#include <mpi.h>  
#include <iostream>  
#include <cstdlib>  
  
#define max\_rows 100000  
#define send\_data\_tag 2001  
#define return\_data\_tag 2002  
  
int array[max\_rows];  
int array2[max\_rows];  
  
int main(int argc, char \*\*argv)  
{  
 long int sum, partial\_sum;  
 MPI\_Status status;  
 int my\_id, root\_process, num\_rows, num\_procs,  
 an\_id, num\_rows\_to\_receive, avg\_rows\_per\_process,  
 sender, num\_rows\_received, start\_row, end\_row, num\_rows\_to\_send;  
  
 /\* Теперь реплицируем этот процесс для создания параллельных процессов.  
 \* С этого момента каждый процесс выполняет отдельную копию  
 \* этой программы \*/  
  
 MPI\_Init(&argc, &argv);  
  
 root\_process = 0;  
  
 /\* узнать МОЙ ИД процесса и сколько процессов было запущено. \*/  
  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_id);  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &num\_procs);  
  
 if(my\_id == root\_process) {  
  
 /\* Я должен быть корневым процессом, поэтому я запрошу пользователя  
 \*, чтобы определить, сколько чисел нужно суммировать. \*/  
  
 printf("\nВведите количество чисел для суммирования: ");  
 scanf("%i", &num\_rows);  
  
 if(num\_rows > max\_rows) {  
 printf("Слишком большое число\n");  
 exit(1);  
 }  
  
 avg\_rows\_per\_process = num\_rows / num\_procs;  
  
 /\* инициализировать массив \*/  
 srand(num\_rows);  
 printf("\n Сгенерированый массив: [");  
 for(int i = 0; i < num\_rows; i++) {  
 array[i] = rand() % 100;;  
 printf("%d, ",array[i]);  
 }  
 printf("]\n");  
  
 /\* распределяем часть bector каждому дочернему процессу \*/  
  
 for(int an\_id = 1; an\_id < num\_procs; an\_id++) {  
 start\_row = an\_id\*avg\_rows\_per\_process + 1;  
 end\_row = (an\_id + 1)\*avg\_rows\_per\_process;  
  
 if((num\_rows - end\_row) < avg\_rows\_per\_process)  
 end\_row = num\_rows - 1;  
  
 num\_rows\_to\_send = end\_row - start\_row + 1;  
  
 MPI\_Send( &num\_rows\_to\_send, 1 , MPI\_INT,  
 an\_id, send\_data\_tag, MPI\_COMM\_WORLD);  
  
 MPI\_Send( &array[start\_row], num\_rows\_to\_send, MPI\_INT,  
 an\_id, send\_data\_tag, MPI\_COMM\_WORLD);  
 }  
  
 /\* и вычисляем сумму значений в сегменте, присвоенном  
 \* корневому процессу \*/  
  
 sum = 0;  
 for(int i = 0; i < avg\_rows\_per\_process + 1; i++) {  
 sum += array[i];  
 }  
  
 printf("Сумма %li посчитанная главным процессом (0)\n", sum);  
  
 /\* и, наконец, я собираю частичные суммы из подчиненных процессов,  
 \* распечатываю их, добавляю к общей сумме и распечатываю \*/  
  
 for(int an\_id = 1; an\_id < num\_procs; an\_id++) {  
  
 MPI\_Recv( &partial\_sum, 1, MPI\_LONG, MPI\_ANY\_SOURCE,  
 return\_data\_tag, MPI\_COMM\_WORLD, &status);  
  
 sender = status.MPI\_SOURCE;  
  
 printf("Частичная сумма %li возвращена из процесса %i \n", partial\_sum, sender);  
  
 sum += partial\_sum;  
 }  
  
 printf("Итого: %li\n", sum);  
 }  
  
 else {  
  
 /\* Я должен быть подчиненным процессом, поэтому я должен получить свой сегмент массива  
 \* и сохранить его в «локальном» массиве array1. \*/  
  
 MPI\_Recv( &num\_rows\_to\_receive, 1, MPI\_INT,  
 root\_process, send\_data\_tag, MPI\_COMM\_WORLD, &status);  
  
 MPI\_Recv( &array2, num\_rows\_to\_receive, MPI\_INT,  
 root\_process, send\_data\_tag, MPI\_COMM\_WORLD, &status);  
  
 num\_rows\_received = num\_rows\_to\_receive;  
  
 /\* Вычислить сумму моей части массива \*/  
  
 partial\_sum = 0;  
 for(int i = 0; i < num\_rows\_received; i++) {  
 partial\_sum += array2[i];  
 }  
  
 /\* и, наконец, отправить мою частичную сумму корневому процессу \*/  
  
 MPI\_Send( &partial\_sum, 1, MPI\_LONG, root\_process,  
 return\_data\_tag, MPI\_COMM\_WORLD);  
 }  
 MPI\_Finalize();  
}