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

```
nereus@Nereus:~/CLionProjects/untitled1$ mpirun -np 8 a.out
Invalid MIT-MAGIC-COOKIE-1 key
10
Введите количество чисел для суммирования:
 Сгенерированый массив: [95, 8, 78, 25, 18, 77, 75, 71, 47, 7, ]
Сумма 103 посчитанная главным процессом (0)
Частичная сумма 25 возвращена из процесса 2
Частичная сумма 78 возвращена из процесса 1
Частичная сумма 18 возвращена из процесса 3
Частичная сумма 77 возвращена из процесса 4
Частичная сумма 75 возвращена из процесса 5
Частичная сумма 71 возвращена из процесса 6
Частичная сумма 47 возвращена из процесса 7
Итого: 494
nereus@Nereus:~/CLionProjects/untitled1$
```

Рис. 1

Тоже самое для двух процессов и 11 элементов рис. 2

```
nereus@Nereus:~/CLionProjects/untitled1$ mpirun -np 2 a.out
Invalid MIT-MAGIC-COOKIE-1 key
11
Введите количество чисел для суммирования:
 Сгенерированый массив: [23, 94, 97, 45, 3, 95, 81, 81, 89, 23, 0, ]
Сумма 357 посчитанная главным процессом (0)
Частичная сумма 274 возвращена из процесса 1
Итого: 631
nereus@Nereus:~/CLionProjects/untitled1$
```

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();
}
```