МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра вычислительной техники

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Параллельные вычисления»
Тема: «Передача данных по процессам»

Студентка гр. 1307 _____ Голубев М.А.

Преподаватель _____ Манжиков Л.П.

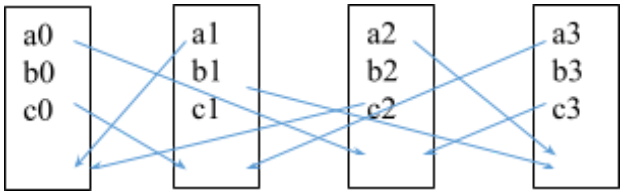
Санкт-Петербург
2025

Цель работы.

Освоить функции передачи данных между процессами.

Задание 1.

1) Запустить 4 процесса.
2) На каждом процессе создать переменные: ai,bi,ci, где I – номер процесса. Инициализировать переменные. Вывести данные на печать.
3) Передать данные на другой процесс. Напечатать номера процессов и поступившие данные. Найти: c0=a1+b2; c1=a3+b0; c2=a0+b3; c3=a2+b1.



Текст программы lab2_1.c.

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
   int rank, size;
   int ai, bi, ci;
   int received_a, received_b;

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);

      if (size != 4)
   {
      if (rank == 0)
      {
         printf("Эта программа требует ровно 4 процесса.\n");
      }
      MPI_Finalize();
      return 1;
   }

   ai = rank + 1;
   bi = (rank + 1) * 10;
   ci = 0;
   printf("Процесс %d: ai = %d, bi = %d, ci = %d\n", rank, ai, bi, ci);

   if (rank == 0)
   {
      MPI_Send(&ai, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
      MPI_Send(&bi, 1, MPI_INT, 1, 1, MPI_COMM_WORLD);
      MPI_Recv(&received_a, 1, MPI_INT, 3, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

```
        MPI_Recv(&received_b, 1, MPI_INT, 3, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
    else if (rank == 1)
    {
        MPI_Send(&ai, 1, MPI_INT, 2, 0, MPI_COMM_WORLD);
        MPI_Send(&bi, 1, MPI_INT, 2, 1, MPI_COMM_WORLD);
        MPI_Recv(&received_a, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(&received_b, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
    else if (rank == 2)
    {
        MPI_Send(&ai, 1, MPI_INT, 3, 0, MPI_COMM_WORLD);
        MPI_Send(&bi, 1, MPI_INT, 3, 1, MPI_COMM_WORLD);
        MPI_Recv(&received_a, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(&received_b, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
    else if (rank == 3)
    {
        MPI_Send(&ai, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
        MPI_Send(&bi, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
        MPI_Recv(&received_a, 1, MPI_INT, 2, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(&received_b, 1, MPI_INT, 2, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }

    printf("Процесс %d получил данные: a = %d, b = %d\n", rank, received_a, received_b);

    if (rank == 0)
    {
        ci = received_a + received_b;
    }
    else if (rank == 1)
    {
        ci = received_a + received_b;
    }
    else if (rank == 2)
    {
        ci = received_a + received_b;
    }
    else if (rank == 3)
    {
        ci = received_a + received_b;
    }

    printf("Процесс %d: ci = %d\n", rank, ci);
    MPI_Finalize();
    return 0;
}
```

```
16
17  CMD ["sh", "-c", "mpicc $F_NAME -o main && mpiexec -n $N ./main"]
```

ПРОБЛЕМЫ 5 ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ПОРТЫ КОММЕНТ

```
mishhgun@Air-Mihail lab2 % docker run --rm mpich-2-1
Процесс 0: ai = 1, bi = 10, ci = 0
Процесс 1: ai = 2, bi = 20, ci = 0
Процесс 1 получил данные: a = 1, b = 10
Процесс 1: ci = 11
Процесс 2: ai = 3, bi = 30, ci = 0
Процесс 2 получил данные: a = 2, b = 20
Процесс 2: ci = 22
Процесс 3: ai = 4, bi = 40, ci = 0
Процесс 3 получил данные: a = 3, b = 30
Процесс 3: ci = 33
Процесс 0 получил данные: a = 4, b = 40
Процесс 0: ci = 44
```

Рисунок 1. Запуск программы на 4-х процессах.

Задание 2.

Запустить n процессов и найти по вариантам:

1. Сумму нечетных элементов вектора;
2. Минимальный элемент;
3. Максимальный элемент;
4. Среднее арифметическое элементов вектора;
5. Сумму элементов кратных 3;
6. Количество четных положительных элементов;
7. Максимальный элемент среди отрицательных чисел;
8. **Минимальный элемент среди положительных чисел;**
9. Сумму элементов из заданного пользователем диапазона.

Текст программы lab2_2.c

```c
#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#include <time.h>
#include "mpi.h"

#define DEFAULT_ARR_SIZE 10

int main(int argc, char **argv)
{
   int rank, size;
   int local_min = INT_MAX;
```

```
    int global_min = INT_MAX;
    int array_size = 20;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (rank == 0)
    {
        if (argc > 1)
        {
            array_size = atoi(argv[1]);
            if (array_size <= 0)
            {
                printf("Размер массива не задан, используется размер по умолчанию
(20).\n");
                array_size = DEFAULT_ARR_SIZE;
            }
        }
        printf("Используемый размер массива: %d\n", array_size);
    }

    MPI_Bcast(&array_size, 1, MPI_INT, 0, MPI_COMM_WORLD);

    int *array = NULL;
    if (rank == 0)
    {
        array = (int *)malloc(array_size * sizeof(int));
        srand(time(NULL));
        printf("Массив: ");
        for (int i = 0; i < array_size; ++i)
        {
            array[i] = rand() % 100 - 50;
            printf("%d ", array[i]);
        }
        printf("\n");
    }

    int local_size = array_size / size;
    int *local_array = (int *)malloc(local_size * sizeof(int));
```

```
    MPI_Scatter(array, local_size, MPI_INT, local_array, local_size, MPI_INT, 0,
MPI_COMM_WORLD);

    for (int i = 0; i < local_size; ++i)
    {
        if (local_array[i] > 0 && local_array[i] < local_min)
        {
            local_min = local_array[i];
        }
    }

    MPI_Reduce(&local_min, &global_min, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);

    if (rank == 0)
    {
        if (global_min != INT_MAX)
        {
            printf("Минимальный положительный элемент: %d\n", global_min);
        }
        else
        {
            printf("В массиве нет положительных элементов.\n");
        }
        free(array);
    }

    free(local_array);
    MPI_Finalize();
    return 0;
}
```

```
mishhgun@MacBook-Air-Mihail lab2 % docker run mpich-2-2
Используемый размер массива: 20
Массив: -42 12 1 11 41 2 25 -1 43 49 -12 -14 -42 2 -39 17 -37 -18 18 1
Минимальный положительный элемент: 1
```

Рисунок 2. Запуск программы на 4-х процессах

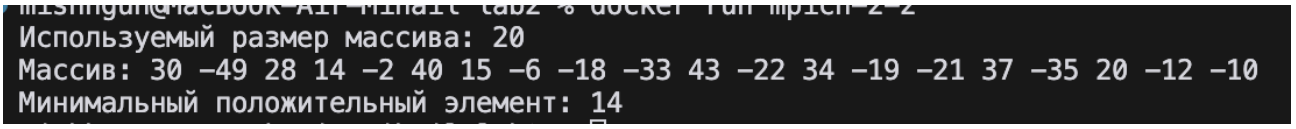


Рисунок 3. Запуск программы на 8-ми процессах

Выводы.

В ходе выполнения лабораторной работы были изучены ключевые идеи и возможности MPI, которые требуются для создания параллельных программ. В частности, были освоены процедуры инициализации, определения количества и рангов процессов, передачи и приема данных, распределения вычислений и сбора результатов.

Были получены практические умения работы с базовыми функциями MPI для создания параллельных приложений. Это является основой для разработки более сложных параллельных алгоритмов. Также был получен опыт разработки, тестирования и отладки параллельных программ.