**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 |  | Грунская Н.Д. |
| 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; с1=a3+b0; c2=a0+b3; c3=a2+b1.

a0

b0

c0

a1

b1

c1

a2

b2

c2

a3

b3

c3

Текст программы task1.cpp.

#include <cstdlib>

#include <iostream>

#include <ctime>

#include "mpi.h"

#define MAX 10

using namespace std;

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

{

const int tag = 198;

int rank, size, n, i, ibeg, iend, ai,bi,ci;

MPI\_Status status;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

srand(rank);

ai = (rand() % 100) + 1;

bi = (rand() % 100) + 1;

printf("rank = %d, a%d = %d, b%d = %d\n", rank,rank,ai,rank,bi);

printf("Sending: a%d = %d, b%d = %d\n",rank,ai,rank,bi);

switch (rank){

case 0:

{

MPI\_Send(&bi, 1, MPI\_INT, 1, tag, MPI\_COMM\_WORLD);

MPI\_Send(&ai, 1, MPI\_INT, 2, tag, MPI\_COMM\_WORLD);

MPI\_Recv(&ai, 1, MPI\_INT, 1, tag, MPI\_COMM\_WORLD,&status);

MPI\_Recv(&bi, 1, MPI\_INT, 2, tag, MPI\_COMM\_WORLD,&status);

break;

}

case 1:

{

MPI\_Send(&ai, 1, MPI\_INT, 0, tag, MPI\_COMM\_WORLD);

MPI\_Send(&bi, 1, MPI\_INT, 3, tag, MPI\_COMM\_WORLD);

MPI\_Recv(&ai, 1, MPI\_INT, 3, tag, MPI\_COMM\_WORLD,&status);

MPI\_Recv(&bi, 1, MPI\_INT, 0, tag, MPI\_COMM\_WORLD,&status);

break;

}

case 2:

{

MPI\_Send(&ai, 1, MPI\_INT, 3, tag, MPI\_COMM\_WORLD);

MPI\_Send(&bi, 1, MPI\_INT, 0, tag, MPI\_COMM\_WORLD);

MPI\_Recv(&ai, 1, MPI\_INT, 0, tag, MPI\_COMM\_WORLD,&status);

MPI\_Recv(&bi, 1, MPI\_INT, 3, tag, MPI\_COMM\_WORLD,&status);

break;

}

case 3:

{

MPI\_Send(&ai, 1, MPI\_INT, 1, tag, MPI\_COMM\_WORLD);

MPI\_Send(&bi, 1, MPI\_INT, 2, tag, MPI\_COMM\_WORLD);

MPI\_Recv(&ai, 1, MPI\_INT, 2, tag, MPI\_COMM\_WORLD,&status);

MPI\_Recv(&bi, 1, MPI\_INT, 1, tag, MPI\_COMM\_WORLD,&status);

break;

}

}

printf("Received: a = %d, b = %d\n",ai,bi);

printf("c%d = %d\n\n",rank,ai+bi);

MPI\_Finalize();

return 0;

}

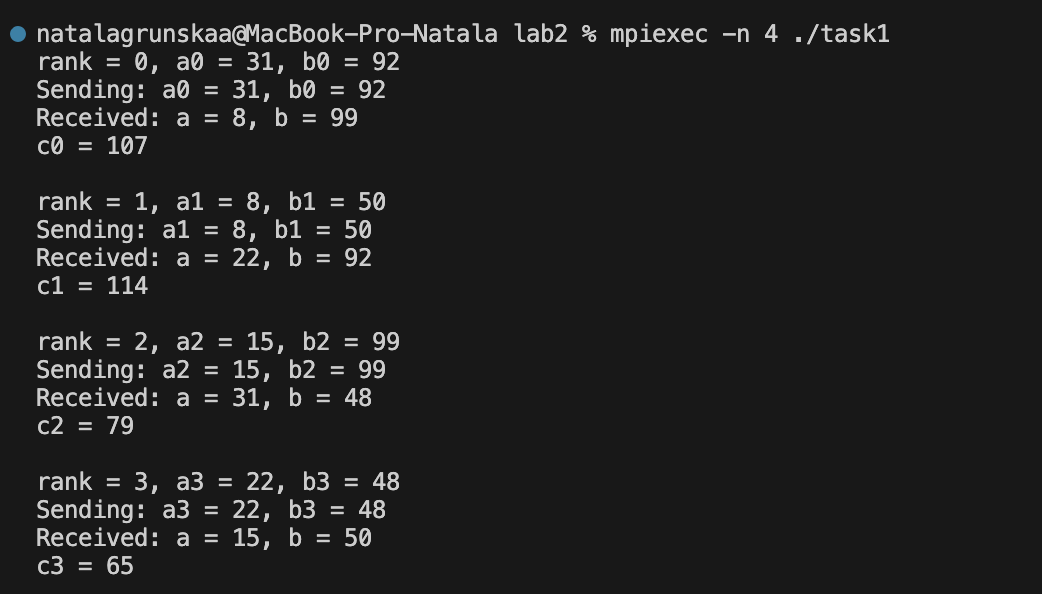


Рисунок 1. Запуск программы на 4-х процессах.

**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 task2.cpp

//3. Максимальный элемент;

#include <cstdlib>

#include <iostream>

#include <ctime>

#include "mpi.h"

using namespace std;

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

{

int rank, size;

MPI\_Status status;

int n= 20;

int vector[n];

srand(time(NULL));

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

int local\_max=0, max=0;

if (rank==0){

for (int i=0;i<n;i++){

vector[i]= (rand() % 100);

printf("%d ",vector[i]);

}

printf("\n");

}

MPI\_Bcast(&vector,n,MPI\_INT,0,MPI\_COMM\_WORLD);

for (int i=rank;i<n;i=i+size){

if(vector[i]> local\_max){

local\_max = vector[i];

}

}

MPI\_Reduce(&local\_max,&max,1,MPI\_INT,MPI\_MAX,0, MPI\_COMM\_WORLD);

if(rank==0){

printf("max = %d\n",max);

}

MPI\_Finalize();

return 0;

}

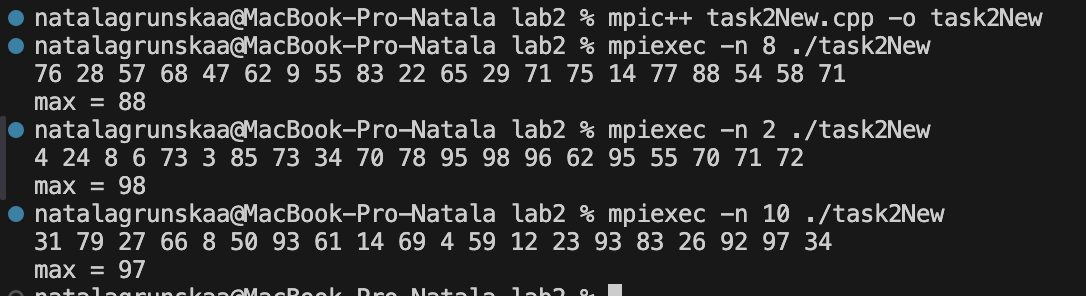


Рисунок 2. Запуск программы на 2-х,8-и, 10-и процессах

**Выводы.**

В результате выполнения лабораторной работы были освоены основные концепции и функции MPI, необходимые для реализации параллельных программ, включая инициализацию, определение количества и рангов процессов, отправку и прием данных, распределение вычислений и сбор результатов. Были получены практические навыки работы с базовыми функциями MPI для построения параллельных приложений, что является фундаментом для разработки более сложных параллельных алгоритмов. Также получен опыт разработки, тестирования и отладки параллельных программ.