**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра вычислительной техники**

отчет

**по лабораторной работе №4**

**по дисциплине «Параллельные вычисления»**

**Тема: «Коллективные функции»**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1307 |  | Грунская Н.Д. |
| Преподаватель |  | Манжиков Л.П. |

Санкт-Петербург

2025

**Цель работы.**

Освоить функции коллективной обработки данных.

**Задание 1 (по вариантам).**

Решить задание 1 из лаб. работы 2 с применением коллективных функций.

**Задание 2 (по вариантам).**

В полученной матрице (по результатам выполнения задания 1) найти:

Решить задание 1 или 2 из лаб. работы 3 с применением коллективных функций.

Текст программы task1.cpp.

#include <cstdlib>

#include <iostream>

#include <ctime>

#include "mpi.h"

using namespace std;

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

{

int rank, size;

MPI\_Status status;

MPI\_Init(&argc, &argv);

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

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

int ai=0, max=0;

srand(rank\* time(NULL));

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

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

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

if(rank==0){

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

}

MPI\_Finalize();

return 0;

}

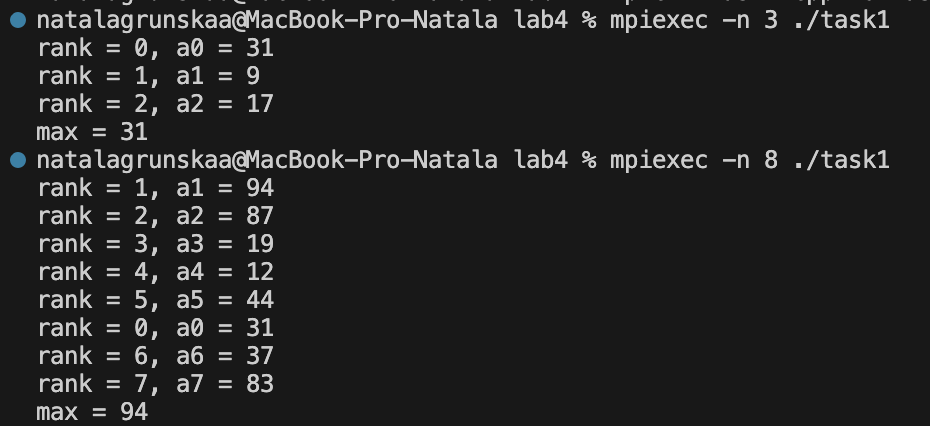


Рисунок 1. Запуск программы на 3-х и 8-и процессах.

Текст программы task2.cpp.

#include <stdio.h>

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include "mpi.h"

#define ROWS 5

#define COLUMNS 6

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

{

int rank, size;

MPI\_Init(&argc, &argv);

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

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

int matrix[ROWS][COLUMNS];

int modified\_matrix[ROWS][COLUMNS];

if (rank == 0)

{

srand(time(NULL));

std::cout << "Old matrix:\n";

for (int i = 0; i < ROWS; i++)

{

for (int j = 0; j < COLUMNS; j++)

{

matrix[i][j] = (rand() % 21) - 10; // Генерация чисел от -10 до 10

std::cout << matrix[i][j] << " ";

}

std::cout << "\n";

}

}

// Определяем количество строк для каждого процесса

int \*send\_counts = new int[size]; // Количество строк для каждого процесса

int \*displs = new int[size]; // Смещения для каждого процесса

int rows\_per\_process = ROWS / size;

int remainder = ROWS % size;

for (int i = 0; i < size; i++)

{

send\_counts[i] = (i < remainder) ? (rows\_per\_process + 1) \* COLUMNS : rows\_per\_process \* COLUMNS;

displs[i] = (i == 0) ? 0 : displs[i - 1] + send\_counts[i - 1];

}

// Количество строк, которые обрабатывает текущий процесс

int local\_rows = (rank < remainder) ? rows\_per\_process + 1 : rows\_per\_process;

int local\_elements = local\_rows \* COLUMNS;

// Локальный буфер для хранения строк, обрабатываемых текущим процессом

int \*local\_matrix = new int[local\_elements];

// Распределение строк матрицы с использованием MPI\_Scatterv

MPI\_Scatterv(matrix, send\_counts, displs, MPI\_INT, local\_matrix, local\_elements, MPI\_INT, 0, MPI\_COMM\_WORLD);

// Обработка локальной части матрицы

for (int i = 0; i < local\_rows; i++)

{

for (int j = 0; j < COLUMNS; j++)

{

int index = i \* COLUMNS + j;

if (local\_matrix[index] > 0)

{

local\_matrix[index] = 1;

}

else if (local\_matrix[index] < 0)

{

local\_matrix[index] = -1;

}

}

}

// Сбор обработанных данных обратно в процесс с рангом 0

MPI\_Gatherv(local\_matrix, local\_elements, MPI\_INT, modified\_matrix, send\_counts, displs, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (rank == 0)

{

std::cout << "\nNew matrix:\n";

for (int i = 0; i < ROWS; i++)

{

for (int j = 0; j < COLUMNS; j++)

{

std::cout << modified\_matrix[i][j] << " ";

}

std::cout << "\n";

}

}

// Освобождение памяти

delete[] send\_counts;

delete[] displs;

delete[] local\_matrix;

MPI\_Finalize();

return 0;

}

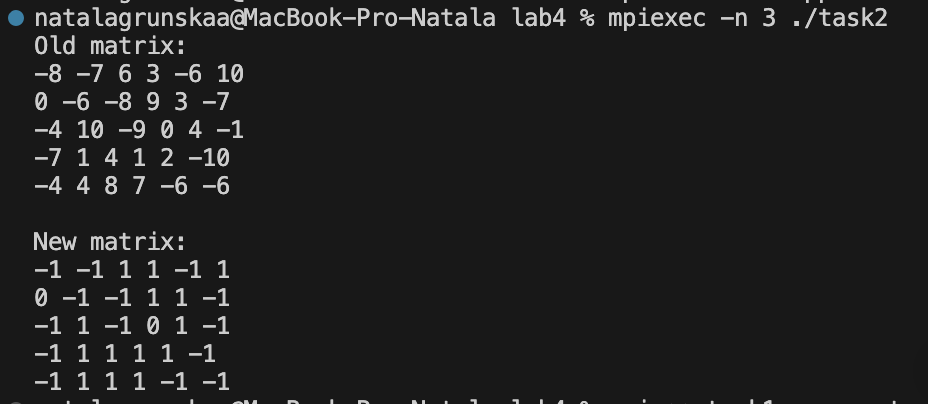


Рисунок 2. Запуск программы на 3 процессах

**Выводы.**

В ходе выполнения лабораторной работы были успешно освоены функции коллективной обработки данных в MPI. Получено понимание принципов их работы и практические навыки применения для реализации параллельных алгоритмов. Приобретен опыт использования коллективных операций для обмена данными между процессами и эффективного распределения вычислительной нагрузки. Также закреплены знания о необходимости синхронизации процессов при работе с коллективными функциями для обеспечения корректного выполнения параллельной программы.