МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего профессионального образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №14

РАБОТА ЗАЩИЩЕНА С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | А. Ю. Сыщиков |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3  MPI, Сортировка элементов массива |
|  |
| по дисциплине: [Системы с параллельной обработкой информации](https://pro.guap.ru/inside_s#subjects/2436975) |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 1742 |  |  |  |  |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2021

1. **Цель работы**

Произвести сортировку элементов в столбцах (или строках) матрицы размерности NxM, с использованием распределения вычислений между процессами средствами MPI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варианта | N | M | Тип элемента вектора | Тип сортировки |
| 2 | 4 | 9 | Знаковый короткий целый | По возрастанию |

1. **Текст программы**

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <iostream>

#include <ctime>

#include <mpi.h>

using namespace std;

#define root 0

static int gsize, myid;

#define N 4

#define M 9

void initArrayNew(short int\* a, int id) {

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

if (id >= gsize - 1) { //если у нас кратный К элемент, то его надо бы сделать пустотны

a[i] = 11111;//создаем пустотный элемент

}

else {

a[i] = rand() % 20;

}

}

}

void initArray2(short int\* a) {

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

a[i] = rand() % 20;

}

}

void printArray(short int\* a, int m) {

printf("Process %d Massiv:", myid);

printf("[ ");

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

if (a[i] != 11111) {

printf("%hd ", a[i]);

}

else {

printf("NOPE ");

}

}

printf("]\n");

}

void sort(short int\* a) {

// Сортировка массива пузырьком

for (int i = 0; i < M - 1; i++) {

for (int j = 0; j < M - i - 1; j++) {

if (a[j] < a[j + 1]) {

// меняем элементы местами

short int temp = a[j];

a[j] = a[j + 1];

a[j + 1] = temp;

}

}

}

}

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

{

int kolvo\_str;//количество строк, которое будет выводить каждый процесс

int s = 0; //суммарное количество ячеек + используется при подсчете, как распределить память между процессами

short int sendbuf[N][M]; //изначальный массив

short int\* rbuf;//результат (отсортированные строки) в каждом процессе свои

int \* displs, \* scounts; //массив значений первого элемента для каждого процесса // массив количества ячеек для каждого процесса

int namelen;

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];

MPI\_Init(&argc, &argv);

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

displs = (int\*)malloc(gsize \* sizeof(int));//начальный индекс для каждого процесса

scounts = (int\*)malloc(gsize \* sizeof(int));//показывает количество элементов в каждом процессе

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

MPI\_Get\_processor\_name(processor\_name, &namelen);

int k = N / gsize; //определяем по скольку ячеек выдать каждому процессу

int l = N % gsize; //остаток ячеек, котоырй надо разделить между процессами

/\*Первый случай, когда данных больше, чем процессов, но не кратно числу процессов. Например 5 строчек, но 4 процесса.\*/

if ((l > 0) && (k > 0)) {//если остаток имеется и у нас хоть раз поделилось, то разделим между процессами

for (int i = 0; i < gsize; i++) { //раздаем всем процесса по k элементов

scounts[i] = k;

displs[i] = i \* k;

}

for (int i = 0; i < l; i++) {//первым процессам раздаем остаток

scounts[i]++;

}

displs[0] = 0;//пересчитываем диапазон начального элемента для каждого процесса

for (int i = 0; i < gsize - 1; i++) {

s = s + scounts[i];

displs[i + 1] = s;

}

s = M \* scounts[myid];

}

/\*Второй случай. Когда количество строк меньше количества процессов. Например, 5 строк и 6 процессов.\*/

else if ((l > 0) && (k == 0)) {

for (int i = 0; i < gsize; i++) { //раздаем всем процесса по k элементов

scounts[i] = 1; //раздаем всем по 1

displs[i] = i;

}

s = M;

}

/\*Третий случай. Когда количество строк равно количествую процессов. Например, 5 строк и 5 процессов\*/

else if (l == 0) {//если у нас нет остатка, то есть всем по k

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

scounts[i] = k;

displs[i] = i \* k;

}

s = k \* gsize \* M;

}

/\*кажому процессу назначаем его количество строк\*/

if (myid < N) {

kolvo\_str = scounts[myid];

}

else {//если айди процесса больше или равен N, то значит он будет бездействовать, то есть 0 строк.

kolvo\_str = 0;

}

int MPI\_Barrier(MPI\_COMM\_WORLD);

if (myid == 0) {

printf("Process %d on %s The original Massiv:\n", myid, processor\_name);

fflush(stdout);

if (((l > 0) && (k > 0))) {/\*Если первый случай, то\*/

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

initArray2(sendbuf[i]);

printf("Original Massiv:");

printArray(sendbuf[i], M);

fflush(stdout);

}

}

if (((l > 0) && (k == 0))) {/\*Если второй случай\*/

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

initArrayNew(sendbuf[i], i);

printArray(sendbuf[i], M);

fflush(stdout);

}

}

if (l == 0) {

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

initArray2(sendbuf[i]);

printArray(sendbuf[i], M);

fflush(stdout);

}

}

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

scounts[i] \*= M;

displs[i] \*= M;

}

printf("\tProcess %d on %s distribution of array\n", myid, processor\_name);

}

fflush(stdout);

int MPI\_Barrier1(MPI\_COMM\_WORLD);

rbuf = (short int\*)malloc(s \* sizeof(short int));

MPI\_Scatterv(sendbuf, scounts, displs, MPI\_SHORT, rbuf, s, MPI\_SHORT, root, MPI\_COMM\_WORLD);

for (int i = 0; i < kolvo\_str; ++i) {

printf("\t");

printArray(&rbuf[i \* M], M);

}

fflush(stdout);

int MPI\_Barrier3(MPI\_COMM\_WORLD);

for (int i = 0; i < kolvo\_str; ++i) {

sort(&rbuf[i \* M]);

fflush(stdout);

}

fflush(stdout);

int MPI\_Barrier4(MPI\_COMM\_WORLD);

for (int i = 0; i < kolvo\_str; ++i) {

printf("New\_m:");

printArray(&rbuf[i \* M], M);

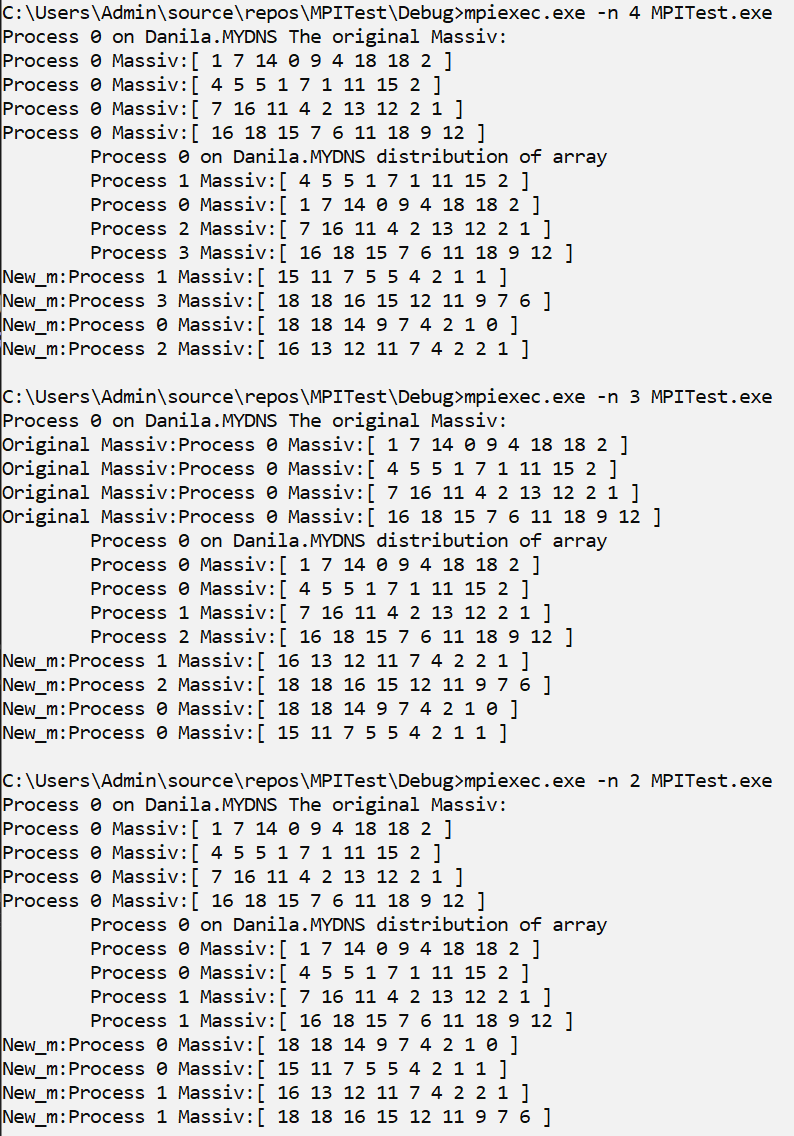
}

MPI\_Finalize();

return 0;

}

1. **Результат работы программы**



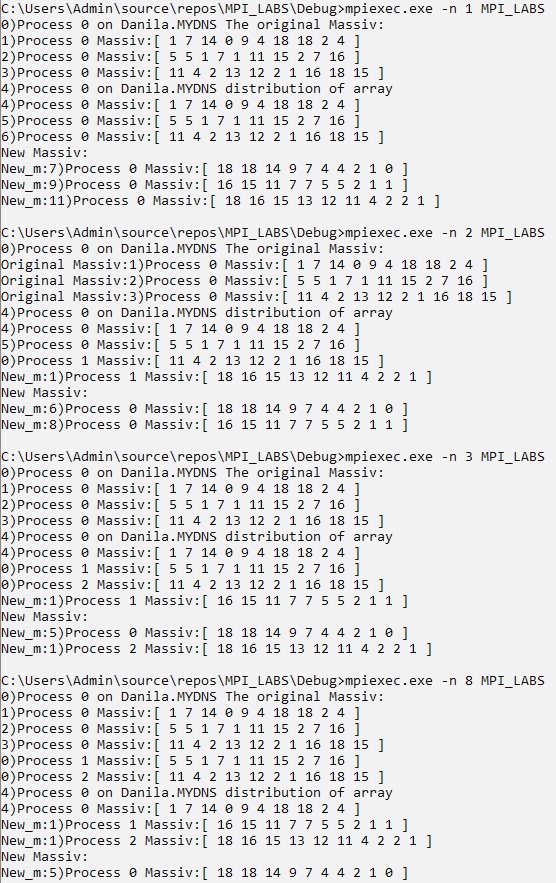
****

Рисунок 1. Результат работы программы