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 |  |  |  | В.А. Седов |
|  |  |  | 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 |
| 16 | 10 | 3 | Длинное целое | По убыванию |

1. **Текст программы**

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <iostream>

#include <ctime>

#include <mpi.h>

using namespace std;

int command = 0;

#define root 0

static int gsize, myid;

#define N 3

#define M 10

void initArrayNew(long 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 initArrayDouble(long int\* a) {

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

{

a[i] = rand() % 20;

}

}

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

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

command++;

printf("[ ");

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

if (a[i] != 11111) {

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

}

else {

printf("NOPE ");

}

}

printf("]\n");

}

void sort(long 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

long int temp = a[j];

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

a[j + 1] = temp;

}

}

}

}

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

{

int kolvo\_str;

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

long int sendbuf[N][M];

long int\* rbuf;

int stride, \* displs, \* scounts; //размерность для каждого процесса (если M кратно N) //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 первый 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);

//gsize = 2;

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

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

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];//s + scounts[gsize - 1];//M

/\*if (myid == 0) {

if (s == N) {

printf("Distribution successful\n");

fflush(stdout);

}

else {

printf("Distribution failed\n");

fflush(stdout);

MPI\_Finalize();//заменить

return 0;

}

}\*/

}

else if ((l > 0) && (k == 0)) { //когда количество ячеек меньше количества процессов

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

scounts[i] = 1; //раздаем всем по 1

displs[i] = i;

}

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

scounts[i] = 0; //раздаем всем по 1

displs[i] = displs[N-1];

}\*/

s = M; //N gsize

}

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

stride = k;

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

scounts[i] = k;

displs[i] = i \* k;

}

s = k \* gsize \* M;

}

if (myid < N) {

kolvo\_str = scounts[myid];

}

else {

kolvo\_str = 0;

}

//printf("id %d kolvo\_str %d:\n", myid, kolvo\_str);

fflush(stdout);

//int MPI\_BARRIER(MPI\_COMM\_WORLD);

if (myid == 0) {

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

fflush(stdout);

command++;

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

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

initArrayDouble(sendbuf[i]);

printf("Original Massiv:");

printArray(sendbuf[i], M);

fflush(stdout);

}

}

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

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) {

initArrayDouble(sendbuf[i]);

printArray(sendbuf[i], M);

fflush(stdout);

}

}

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

scounts[i] \*= M;

displs[i] \*= M;

}

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

}

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

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

fflush(stdout);

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

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

//printf("id %d kolvo\_str %d:\n", myid, kolvo\_str);

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

}

fflush(stdout);

int MPI\_BARRIER2(MPI\_COMM\_WORLD);

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

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

}

fflush(stdout);

int MPI\_BARRIER3(MPI\_COMM\_WORLD);

if (myid == 0) {

printf("New Massiv:\n");

}

int MPI\_BARRIER4(MPI\_COMM\_WORLD);

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

printf("New\_m:");

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

command++;

}

MPI\_Finalize();

return 0;

}

1. **Результат работы программы**

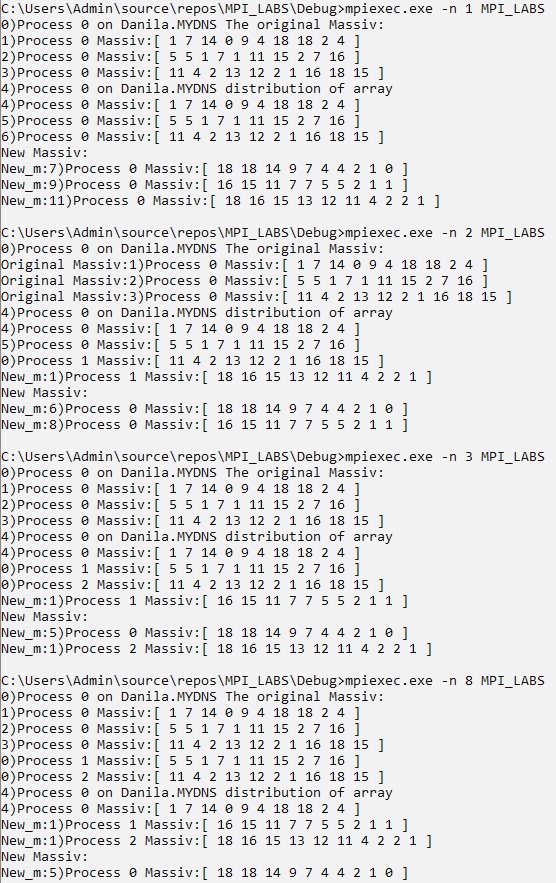
****

Рисунок 1. Результат работы программы