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

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 1742		Д.В. Коробков
	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
9	3	10	Без знаковый целый	По убыванию

2. Текст программы

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <iostream>
#include <ctime>
#include <mpi.h>
using namespace std;
int command = 0;
static int gsize, myid;
#define N 3
#define M 10
void initArrayNew(unsigned int* a, int id) {
	for (int i = 0; i < M; i++) {
		if (id >= gsize - 1) { //если у нас кратный K элемент, то его надо бы сделать пустотны
			a[i] = 11111;//создаем пустотный элемент
		}
		else {
			a[i] = rand() % 20;
		}
	}
}
void initArrayDouble(unsigned int* a) {
	for (int i = 0; i < M; i++)
	{
		a[i] = rand() % 20;
	}
}
void printArray(unsigned 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(unsigned 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
                                unsigned 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
        unsigned int sendbuf[N][M];
        unsigned 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 = (unsigned int*)malloc(s * sizeof(unsigned int));
		MPI_Scatterv(sendbuf, scounts, displs, MPI_UNSIGNED, rbuf, s, MPI_UNSIGNED, 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;
}
```

3. Результат работы программы

```
C:\Users\Admin\source\repos\MPI_LABS\Debug>mpiexec.exe -n 1 MPI_LABS
0)Process 0 on Danila.MYDNS The original Massiv:
1)Process 0 Massiv:[ 1 7 14 0 9 4 18 18 2 4 ]
2)Process 0 Massiv:[ 5 5 1 7 1 11 15 2 7 16 ]
3)Process 0 Massiv:[ 11 4 2 13 12 2 1 16 18 15 ]
4)Process 0 on Danila.MYDNS distribution of array
4)Process 0 Massiv:[ 1 7 14 0 9 4 18 18 2 4 ]
5)Process 0 Massiv:[ 5 5 1 7 1 11 15 2 7 16 ]
6)Process 0 Massiv:[ 11 4 2 13 12 2 1 16 18 15 ]
New Massiv:
New_m:7)Process 0 Massiv:[ 18 18 14 9 7 4 4 2 1 0 ]
New_m:9)Process 0 Massiv:[ 16 15 11 7 7 5 5 2 1 1 ]
New_m:11)Process 0 Massiv:[ 18 16 15 13 12 11 4 2 2 1 ]

C:\Users\Admin\source\repos\MPI_LABS\Debug>mpiexec.exe -n 2 MPI_LABS
0)Process 0 on Danila.MYDNS The original Massiv:
Original Massiv:1)Process 0 Massiv:[ 1 7 14 0 9 4 18 18 2 4 ]
Original Massiv:2)Process 0 Massiv:[ 5 5 1 7 1 11 15 2 7 16 ]
Original Massiv:3)Process 0 Massiv:[ 11 4 2 13 12 2 1 16 18 15 ]
4)Process 0 on Danila.MYDNS distribution of array
4)Process 0 Massiv:[ 1 7 14 0 9 4 18 18 2 4 ]
5)Process 0 Massiv:[ 5 5 1 7 1 11 15 2 7 16 ]
0)Process 1 Massiv:[ 11 4 2 13 12 2 1 16 18 15 ]
New_m:1)Process 1 Massiv:[ 18 16 15 13 12 11 4 2 2 1 ]
New Massiv:
New_m:6)Process 0 Massiv:[ 18 18 14 9 7 4 4 2 1 0 ]
New_m:8)Process 0 Massiv:[ 16 15 11 7 7 5 5 2 1 1 ]

C:\Users\Admin\source\repos\MPI_LABS\Debug>mpiexec.exe -n 3 MPI_LABS
0)Process 0 on Danila.MYDNS The original Massiv:
1)Process 0 Massiv:[ 1 7 14 0 9 4 18 18 2 4 ]
2)Process 0 Massiv:[ 5 5 1 7 1 11 15 2 7 16 ]
3)Process 0 Massiv:[ 11 4 2 13 12 2 1 16 18 15 ]
4)Process 0 on Danila.MYDNS distribution of array
4)Process 0 Massiv:[ 1 7 14 0 9 4 18 18 2 4 ]
0)Process 1 Massiv:[ 5 5 1 7 1 11 15 2 7 16 ]
0)Process 2 Massiv:[ 11 4 2 13 12 2 1 16 18 15 ]
New_m:1)Process 1 Massiv:[ 16 15 11 7 7 5 5 2 1 1 ]
New Massiv:
New_m:5)Process 0 Massiv:[ 18 18 14 9 7 4 4 2 1 0 ]
New_m:1)Process 2 Massiv:[ 18 16 15 13 12 11 4 2 2 1 ]

C:\Users\Admin\source\repos\MPI_LABS\Debug>mpiexec.exe -n 8 MPI_LABS
0)Process 0 on Danila.MYDNS The original Massiv:
1)Process 0 Massiv:[ 1 7 14 0 9 4 18 18 2 4 ]
2)Process 0 Massiv:[ 5 5 1 7 1 11 15 2 7 16 ]
3)Process 0 Massiv:[ 11 4 2 13 12 2 1 16 18 15 ]
0)Process 1 Massiv:[ 5 5 1 7 1 11 15 2 7 16 ]
0)Process 2 Massiv:[ 11 4 2 13 12 2 1 16 18 15 ]
4)Process 0 on Danila.MYDNS distribution of array
4)Process 0 Massiv:[ 1 7 14 0 9 4 18 18 2 4 ]
New_m:1)Process 1 Massiv:[ 16 15 11 7 7 5 5 2 1 1 ]
New_m:1)Process 2 Massiv:[ 18 16 15 13 12 11 4 2 2 1 ]
New Massiv:
New_m:5)Process 0 Massiv:[ 18 18 14 9 7 4 4 2 1 0 ]
```

Рисунок 1. Результат работы программы.

В ходе работы программы главный процесс инициализирует массив типа данных unsigned int размерности [N][M]. Далее главный процесс рассчитывает, как правильно передать строки массива всем процессам. После вычислений передает каждой своей ветви процесса определённый за ними строки этого массива. Каждый процесс, получив свою строку – сортирует ее с помощью функции сортировки. Далее выводиться на экран отсортированный массив.

На рисунке 1 продемонстрированы 3 примера:

1) когда один процесс, то есть все строки массива остаются за главным процессом. Отсортированный массив выводится на экран главным процессом;

2) когда два процесса, то есть строки массива распределяются между процессами. Отсортированный массив выводится на экран совместно всеми процессами;

3) когда у нас 8 процессов, то есть процессов больше, чем количество строк в массиве. Главный процесс правильно передаст строки. Отсортированный массив будет верным и полным, в соответствии с изначальным массивом.