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 № 1
MPI, Пересылка данных

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

Переслать вектор, размерности M, N процессам, используя различные виды связи между процессами. Элементы вектора задаются произвольно. Элементы вектора пересылаемого и принятого вектора, а также время выполнения должны быть выведены на экран.

№ варианта	M	N	Функция
9	7	5	MPI_Scatter

2. Текст программы

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define M 7
#define root 0

static int gsize, myid;
void initArrayNew(int*a, int m, int k) {
	int h = m - M; //количество требуемых пустотных данных
	int kolvo = 0; //количество уже вписанных пустотных данных
	int inc = 0; //счетчик для присваивания данных
	for (int i = m - 1; i >= 0; i--) {
		if ((i%k == 0) && (kolvo < h)) { //если у нас кратный К элемент, то его надо
бы сделать пустотным
//если у нас еще нехватка пустотных, то
			kolvo++;
			a[i] = 11111;//создаем пустотный элемент
		}
		else {
			a[i] = M - inc;
			inc++;
		}
	}
}
void printArray(int*a, int m) {
	int i;
	printf("[ ");
	for (i = 0; i < m; i++) {
		if (a[i] != 11111) {
			printf("%d ", a[i]);
		}
		else {
			printf("NOPE ");
		}
	}

	printf("]\n");
	}

int main(int argc, char *argv[]) {
	int *rbuf;
	int *sendbuf;
	int namelen;
	double startwtime = 0.0, endwtime;
	char processor_name[MPI_MAX_PROCESSOR_NAME];
	int k, razmer, l = 0;
	MPI_Init(&argc, &argv);
	MPI_Comm_size(MPI_COMM_WORLD, &gsize);
	MPI_Comm_rank(MPI_COMM_WORLD, &myid);
```

```
MPI_Get_processor_name(processor_name, &namelen);
//printf("Process %d on %s\n", myid, processor_name);
fflush(stdout);
//gsize = 9;
if (myid == 0) {
	startwtime = MPI_Wtime();
}
if (M < gsize) {
	razmer = gsize;
	k = 1;
}
else if (M == gsize) {
	razmer = gsize;
	k = 1;
}
else if (M > gsize) {
	if (gsize == 1) {//случай когда у нас 1 процесс
		k = M;//всю память ему предоставляем
		razmer = k;
	}
	else{//если у нас несколько процессов
		l = M / gsize;
		k = l + 1;//количество ячеек на каждый процесс
	}
	if (M%gsize != 0) {//суммарный размер буфера
		razmer = gsize*k;//M + gsize - l
	}

}
rbuf = (int *)malloc(razmer * sizeof(int));
if (myid == root) {//0
	sendbuf = (int *)malloc(razmer * sizeof(int));
	initArrayNew(sendbuf, razmer, k);
	printf("Massiv: ");
	printArray(sendbuf, razmer);
}
MPI_Scatter(sendbuf, k, MPI_INT, rbuf, k, MPI_INT, root, MPI_COMM_WORLD);
//printf("id = %d, sendbuf = %d\n", myid, sendbuf[myid]);
//printf("rbuf = %d\n", rbuf[0]);
printf("Process %d on %s:", myid, processor_name);
printArray(rbuf, k);
fflush(stdout);

if (myid == 0) {
	endwtime = MPI_Wtime();
	fprintf(stderr, "wall clock time = %f\n", endwtime - startwtime);
	fflush(stdout);
}
MPI_Finalize();
return 0;
}
```

3. Результат работы программы

```
C:\Users\Admin\Documents\Visual Studio 2015\Projects\MPITest\Debug>mpiexec.exe -n 1  MPITest.exe
Massiv: [ 1 2 3 4 5 6 7 ]
Process 0 on Danila.MYDNS:[ 1 2 3 4 5 6 7 ]
wall clock time = 0.000086

C:\Users\Admin\Documents\Visual Studio 2015\Projects\MPITest\Debug>mpiexec.exe -n 6  MPITest.exe
Massiv: [ 1 2 NOPE 3 NOPE 4 NOPE 5 NOPE 6 NOPE 7 ]
Process 0 on Danila.MYDNS:[ 1 2 ]
Process 1 on Danila.MYDNS:[ NOPE 3 ]
Process 4 on Danila.MYDNS:[ NOPE 6 ]
Process 2 on Danila.MYDNS:[ NOPE 4 ]
Process 5 on Danila.MYDNS:[ NOPE 7 ]
Process 3 on Danila.MYDNS:[ NOPE 5 ]
wall clock time = 0.000438

C:\Users\Admin\Documents\Visual Studio 2015\Projects\MPITest\Debug>mpiexec.exe -n 7  MPITest.exe
Massiv: [ 1 2 3 4 5 6 7 ]
Process 0 on Danila.MYDNS:[ 1 ]
Process 1 on Danila.MYDNS:[ 2 ]
Process 4 on Danila.MYDNS:[ 5 ]
Process 2 on Danila.MYDNS:[ 3 ]
Process 6 on Danila.MYDNS:[ 7 ]
Process 3 on Danila.MYDNS:[ 4 ]
Process 5 on Danila.MYDNS:[ 6 ]
wall clock time = 0.000450

C:\Users\Admin\Documents\Visual Studio 2015\Projects\MPITest\Debug>mpiexec.exe -n 12  MPITest.exe
Massiv: [ 1 2 3 4 5 6 7 NOPE NOPE NOPE NOPE NOPE ]
Process 0 on Danila.MYDNS:[ 1 ]
Process 8 on Danila.MYDNS:[ NOPE ]
Process 1 on Danila.MYDNS:[ 2 ]
Process 2 on Danila.MYDNS:[ 3 ]
Process 4 on Danila.MYDNS:[ 5 ]
Process 9 on Danila.MYDNS:[ NOPE ]
Process 3 on Danila.MYDNS:[ 4 ]
Process 5 on Danila.MYDNS:[ 6 ]
Process 10 on Danila.MYDNS:[ NOPE ]
Process 6 on Danila.MYDNS:[ 7 ]
Process 11 on Danila.MYDNS:[ NOPE ]
Process 7 on Danila.MYDNS:[ NOPE ]
wall clock time = 0.000577
```

Рисунок 1. Результат работы программы.

В ходе работы программы формируются данные относительного того, сколько процессов пользователь ввел для работы MPI. Существует три варианта:

1) Если количество процессов меньше, чем размер массива, то данные дополнятся «пустышками» (NOPE) до размера массива по формуле: количество процессов * количество данных для передачи на один процесс. После на экран выводятся данные.

2) Если количество процессов равно размеру массива. Тогда данные распределяются по одному на каждый процесс, затем передаются и выводятся на экран.

3) Если количество процессов больше размера массива, то данные дополняются «пустышками» (NOPE) пока размер массива не будет равен

количеству процессов. Далее данные передаются процессам и выводятся на экран.