Министерство науки и высшего образования Российской Федерации
Севастопольский государственный университет
Кафедра ИС

Отчет
по лабораторной работе №2
«Исследование коллективного типа передачи данных, групп и коммуникаторов в MPI»
по дисциплине
«ТЕОРИЯ РАСПРЕДЕЛЕННЫХ СИСТЕМ И ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ»

Выполнил студент группы ИС/б-17-2-о
Горбенко К. Н.
Проверил
Дрозин А. Ю.

Севастополь
2020

1 ЦЕЛЬ РАБОТЫ

Исследовать способы обмена данными между процессами в режиме широковещания или группового обмена с использованием MPI-функций.

2 ПОСТАНОВКА ЗАДАЧИ

Вариант №1. Реализовать блочный алгоритм распределенного параллельного перемножения матриц A и B с размерами $(8 * 5)$ и $(5 * 3)$ соответственно. Вид распределяемых между процессами блоков представлен на рисунке 1.

Рисунок 1 – Перемножение матриц

3 ХОД РАБОТЫ

Исходный код программы представлен на листинге:

```
1 #include <iostream>
2 #include <fstream>
3 #include <mpi.h>
4 #include <iomanip>
5
6 using namespace std;
7
8 int len_for_node(int size, int all) {
9     if (size == 1) {
10         return all;
11     }
12     if (!(all % size)) {
13         return all / size;
14     }
15     return 0;
16 }
17
18 void master() {
19     int size;
20     MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
21      if (size == 1) {
22          cout << "Can‘t run without slaves! Just buy some..." << endl;
23          return;
24      }
25      ifstream is("input.txt");
26      int n, k, m;
27      is >> n >> k >> m;
28      int *init_data = new int[4];
29      init_data[0] = n;
30      init_data[1] = k;
31      init_data[2] = m;
32
33      int len = len_for_node(size - 1, n);
34      init_data[3] = len;
35      if (!len) {
36          cout << "Can‘t split lines by processes" << endl;
37          exit(418);
38      }
39
40      MPI_Barrier(MPI_COMM_WORLD);
41      MPI_Bcast(init_data, 4, MPI_INTEGER, 0, MPI_COMM_WORLD);
42      delete[] init_data;
43
44      int *a = new int[n * k],
45              *b = new int[k * m];
46      for (int i = 0; i < n * k; i++) {
47          is >> a[i];
48      }
49      for (int i = 0; i < k * m; i++) {
50          is >> b[i];
51      }
52
53      int *empty = new int[len * k];
54      int *buf = new int[(len + n) * k];
55      memcpy(buf + (len * k), a, n * k * sizeof(int));
56
57      MPI_Barrier(MPI_COMM_WORLD);
58      MPI_Scatter(buf, len * k, MPI_INTEGER, empty, len * k, MPI_INTEGER, 0,
            MPI_COMM_WORLD);
59      delete[] empty;
60      delete[] buf;
61      delete[] a;
62
63      MPI_Barrier(MPI_COMM_WORLD);
64      MPI_Bcast(b, k * m, MPI_INTEGER, 0, MPI_COMM_WORLD);
65      delete[] b;
66
67      int *c = new int[(len + n) * m];
```

```
68      empty = new int[len * m];
69      MPI_Barrier(MPI_COMM_WORLD);
70      MPI_Gather(empty, len * m, MPI_INTEGER, c, len * m, MPI_INTEGER, 0,
            MPI_COMM_WORLD);
71
72      for (int i = 0; i < n; i++) {
73          for (int j = 0; j < m; j++) {
74              cout << setw(5) << c[len * m + i * m + j] << " ";
75          }
76          cout << endl;
77      }
78      delete[] c;
79  }
80
81  void slave() {
82      int *buf = new int[4];
83
84      MPI_Barrier(MPI_COMM_WORLD);
85      MPI_Bcast(buf, 4, MPI_INTEGER, 0, MPI_COMM_WORLD);
86      int k = buf[1],
87              m = buf[2],
88              len = buf[3];
89      delete[] buf;
90
91      int empty[0];
92      int *a = new int[len * k];
93
94      MPI_Barrier(MPI_COMM_WORLD);
95      MPI_Scatter(empty, 0, MPI_INTEGER, a, len * k, MPI_INTEGER, 0,
            MPI_COMM_WORLD);
96
97      int *b = new int[k * m];
98
99      MPI_Barrier(MPI_COMM_WORLD);
100     MPI_Bcast(b, k * m, MPI_INTEGER, 0, MPI_COMM_WORLD);
101
102     int *c = new int[len * m];
103     for (int i = 0; i < len; i++) {
104         for (int j = 0; j < m; j++) {
105             c[i * m + j] = 0;
106             for (int y = 0; y < k; y++) {
107                 c[i * m + j] += a[i * k + y] * b[y * m + j];
108             }
109         }
110     }
111     delete[] a;
112     delete[] b;
113
```

```
114     MPI_Barrier(MPI_COMM_WORLD);
115     MPI_Gather(c, len * m, MPI_INTEGER, empty, 0, MPI_INTEGER, 0,
            MPI_COMM_WORLD);
116     delete[] c;
117 }
118
119 int main(int argc, char **argv) {
120     int rank;
121
122     MPI_Init(&argc, &argv);
123     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
124
125     rank ? slave() : master();
126
127     MPI_Finalize();
128     return 0;
129 }
```

ВЫВОДЫ

В ходе выполнения лабораторной работы были исследованы способы обмена данными между процессами в режиме широковещания или группового обмена с использованием MPI-функций. Написана программа умножения двух матриц произвольного размера, распределяющая расчет строк динамически между кратным количеством процессов-рабочих.