МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федерально автономное бюджетное образовательное учреждение высшего образования

«Севастопольский государственный университет»

кафедра Информационных систем

Куркчи Ариф Эрнестович

Институт информационных технологий и управления в технических системах

курс 3 группа ИС/б-31-о

09.03.02 Информационные системы и технологии (уровень бакалавриата)

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Теория распределенных систем и параллельных вычислений»

на тему «Исследование возможностей формирования виртуальных топологий вычислительных кластеров»

Отметка о зачете \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_

(дата)

Руководитель практикума

ассистент   Н. В. Балясный

(должность) (подпись) (инициалы, фамилия)

Севастополь 2016

1. Цель работы

Исследовать возможности, предоставляемые MPI по формированию виртуальных топологий.

2. Постановка задачи

Вариант №1

Необходимо реализовать алгоритм перемножения матриц ленточным способом с распределением столбцов.

3. Текст программы

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 1 | #include <iostream> | | 2 | #include <mpi.h> | | 3 | #include <fstream> | | 4 | #include <iomanip> | | 5 |  | | 6 | using namespace std; | | 7 |  | | 8 | MPI\_Comm graph; | | 9 | *int* processRank, processCount; | | 10 | *double* \*\*aMatrix, \*\*bMatrix, \*\*cMatrix, \*row, \*column, \*tmpColumn, \*result; | | 11 |  | | 12 | *int* nextProcess() { | | 13 | *int* rankToNextProcess; | | 14 | MPI\_Graph\_neighbors(graph, processRank, 1, &rankToNextProcess); | | 15 | return rankToNextProcess; | | 16 | } | | 17 |  | | 18 | *void* input(*double* \*\**matrix*, *int* *n*, string *path*) { | | 19 | ifstream in(path); | | 20 | for (*int* i = 0; i < n; i++) { | | 21 | for (*int* j = 0; j < n; j++) { | | 22 | in >> matrix[i][j]; | | 23 | } | | 24 | } | | 25 | } | | 26 |  | | 27 | *void* initBuffers(*double* \**rowsBuf*, *double* \**columnsBuf*) { | | 28 | *int* k = 0; | | 29 | for (*int* i = 0; i < processCount; i++) { | | 30 | for (*int* j = 0; j < processCount; j++) { | | 31 | rowsBuf[k] = aMatrix[i][j]; | | 32 | columnsBuf[k++] = bMatrix[j][i]; | | 33 | } | | 34 | } | | 35 | } | | 36 |  | | 37 | *void* createGraph() { | | 38 | *int* n = processCount; | | 39 | *int* \*index = new *int*[processCount]; | | 40 | *int* \*edges = new *int*[processCount]; | | 41 | for (*int* i = 1; i <= processCount; i++) { | | 42 | index[i - 1] = i; | | 43 | edges[i - 1] = i % processCount; | | 44 | } | | 45 | MPI\_Barrier(MPI\_COMM\_WORLD); | | 46 | MPI\_Graph\_create(MPI\_COMM\_WORLD, n, index, edges, 0, &graph); | | 47 | MPI\_Comm\_size(graph, &processCount); | | 48 | MPI\_Comm\_rank(graph, &processRank); | | 49 | delete[] index; | | 50 | delete[] edges; | | 51 | } | | 52 |  | | 53 | *void* initRowAndColumn(*double* \**rowsBuf*, *double* \**columnsBuf*, *double* \**row*, *double* \**column*) { | | 54 | MPI\_Barrier(graph); | | 55 | MPI\_Scatter(rowsBuf, processCount, MPI\_DOUBLE, row, processCount, MPI\_DOUBLE, 0, graph); | | 56 | MPI\_Barrier(graph); | | 57 | MPI\_Scatter(columnsBuf, processCount, MPI\_DOUBLE, column, processCount, MPI\_DOUBLE, 0, graph); | | 58 | } | | 59 |  | | 60 | *void* iteration(*double* \**row*, *double* \**column*, *double* \**result*, *int* *index*) { | | 61 | result[index] = 0; | | 62 | for (*int* i = 0; i < processCount; result[index] += row[i] \* column[i], i++); | | 63 | } | | 64 |  | | 65 | *void* swapColumns(*double* \**column*, *double* \**tmpColumn*) { | | 66 | MPI\_Status status; | | 67 | *int* next; | | 68 | next = !processRank ? processCount - 1 : processRank - 1; | | 69 | if (!(processRank % 2)) { | | 70 | MPI\_Barrier(graph); | | 71 | MPI\_Send(column, processCount, MPI\_DOUBLE, next, 0, graph); | | 72 | } | | 73 | MPI\_Barrier(graph); | | 74 | MPI\_Recv(tmpColumn, processCount, MPI\_DOUBLE, nextProcess(), MPI\_ANY\_TAG, graph, &status); | | 75 | if (processRank % 2) { | | 76 | MPI\_Barrier(graph); | | 77 | MPI\_Send(column, processCount, MPI\_DOUBLE, next, 0, graph); | | 78 | } | | 79 | } | | 80 |  | | 81 | *void* mult(*double* \**row*, *double* \**column*, *double* \**result*, *double* \**tmpColumn*) { | | 82 | *int* index = processRank; | | 83 | for (*int* i = 0; i < processCount; i++) { | | 84 | iteration(row, column, result, index); | | 85 | if (i != processCount - 1) { | | 86 | swapColumns(column, tmpColumn); | | 87 | for (*int* j = 0; j < processCount; column[j] = tmpColumn[j], j++); | | 88 | } | | 89 | index = (index + 1) % processCount; | | 90 | } | | 91 | } | | 92 |  | | 93 | *void* collect() { | | 94 | *double* \*tmpBuf; | | 95 | if (processRank == 0) { | | 96 | tmpBuf = new *double*[processCount \* processCount]; | | 97 | } else { | | 98 | tmpBuf = new *double*[1]; | | 99 | } | | 100 | MPI\_Barrier(graph); | | 101 | MPI\_Gather(result, processCount, MPI\_DOUBLE, tmpBuf, processCount, MPI\_DOUBLE, 0, graph); | | 102 | if (processRank == 0) { | | 103 | *int* k = 0; | | 104 | for (*int* i = 0; i < processCount; i++) { | | 105 | for (*int* j = 0; j < processCount; cMatrix[i][j++] = tmpBuf[k++]); | | 106 | } | | 107 | } | | 108 | } | | 109 |  | | 110 | *void* print(*double* \*\**matrix*) { | | 111 | for (*int* i = 0; i < processCount; i++) { | | 112 | for (*int* j = 0; j < processCount; cout << setw(3) << matrix[i][j++] << " "); | | 113 | cout << endl; | | 114 | } | | 115 | } | | 116 |  | | 117 | *void* master() { | | 118 | aMatrix = new *double* \*[processCount]; | | 119 | bMatrix = new *double* \*[processCount]; | | 120 | cMatrix = new *double* \*[processCount]; | | 121 | for (*int* i = 0; i < processCount; i++) { | | 122 | aMatrix[i] = new *double*[processCount]; | | 123 | bMatrix[i] = new *double*[processCount]; | | 124 | cMatrix[i] = new *double*[processCount]; | | 125 | } | | 126 | input(aMatrix, processCount, "a.txt"); | | 127 | input(bMatrix, processCount, "b.txt"); | | 128 | result = new *double*[processCount]; | | 129 | row = new *double*[processCount]; | | 130 | column = new *double*[processCount]; | | 131 | tmpColumn = new *double*[processCount]; | | 132 | *double* \*rowsBuf = new *double*[processCount \* processCount]; | | 133 | *double* \*columnsBuf = new *double*[processCount \* processCount]; | | 134 | initBuffers(rowsBuf, columnsBuf); | | 135 | initRowAndColumn(rowsBuf, columnsBuf, row, column); | | 136 | mult(row, column, result, tmpColumn); | | 137 | collect(); | | 138 |  | | 139 | cout << "\tA\*B = C" << endl << endl; | | 140 | print(aMatrix); | | 141 | cout << "\*" << endl; | | 142 | print(bMatrix); | | 143 | cout << "===============" << endl; | | 144 | print(cMatrix); | | 145 |  | | 146 | delete aMatrix; | | 147 | delete bMatrix; | | 148 | delete cMatrix; | | 149 | delete result; | | 150 | delete row; | | 151 | delete column; | | 152 | delete tmpColumn; | | 153 | delete[] rowsBuf; | | 154 | delete[] columnsBuf; | | 155 | } | | 156 |  | | 157 | *void* slave() { | | 158 | result = new *double*[processCount]; | | 159 | row = new *double*[processCount]; | | 160 | column = new *double*[processCount]; | | 161 | tmpColumn = new *double*[processCount]; | | 162 | *double* \*rowsBuf = new *double*[1]; | | 163 | *double* \*columnsBuf = new *double*[1]; | | 164 | initRowAndColumn(rowsBuf, columnsBuf, row, column); | | 165 | mult(row, column, result, tmpColumn); | | 166 | collect(); | | 167 | delete[] result; | | 168 | delete[] row; | | 169 | delete[] column; | | 170 | delete[] tmpColumn; | | 171 | delete[] rowsBuf; | | 172 | delete[] columnsBuf; | | 173 | } | | 174 |  | | 175 | *int* main(*int* *argc*, *char* \*\**argv*) { | | 176 | MPI\_Init(&argc, &argv); | | 177 | MPI\_Comm\_rank(MPI\_COMM\_WORLD, &processRank); | | 178 | MPI\_Comm\_size(MPI\_COMM\_WORLD, &processCount); | | 179 | createGraph(); | | 180 | processRank == 0 ? master() : slave(); | | 181 | MPI\_Finalize(); | | 182 | return 0; | | 183 | } | |  |

4. Тестовый пример

На рисунке 4.1 представлен тестовый пример.

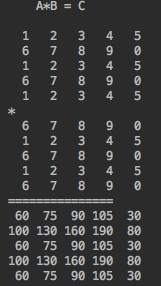


Рисунок 4.1 – результат работы программы

Вывод

В ходе лабораторной работы были исследованы возможности, предоставляемые MPI по формированию виртуальных топологий.