Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

КОЛЛЕКТИВНЫЕ ФУНКЦИИ В MPI. ОБРАБОТКА МАССИВОВ

Лабораторная работа №2

по дисциплине

«Параллельное программирование»

Студент гр. 430-2

_____А.А. Лузинсан

«____»_____2023 г.

Руководитель

Доцент кафедры АСУ

_____ С.М. Алфёров

«____»_____2023 г.

Томск 2023

Оглавление

Введение

Цель работы: освоить применение коллективных функций MPI для рассылки и сборки с фрагментов массивов и параллельной их обработки по заданному алгоритму.

Индивидуальное задание по варианту №:23: переписать элементы каждой строки матрицы в обратном порядке.

1 Ход работы

1. Для предложенного алгоритма составить и отладить последовательную программу обработки числовых массивов индивидуального задания. Использовать динамическое выделение памяти для массивов. Входные массивы заполнить случайными числами. Алгоритм обработки оформить внешней функцией.

Задача заключается в том, чтобы отразить позиции элементов по строкам. Для этого используется функция из стандартной библиотеки std::swap. Сам массив выделяется динамическим способом с помощью оператора new. Входной массив был заполнен рандомными числами посредством алгоритма «Вихрь Мерсенна».

2 Для параллельной обработки определить размер порции массива для каждого процесса и смещение порции от начала полного массива.

Так как в задаче фигурируют матрицы, то разделение задач будет осуществляться по строкам. Для этого вычисляется целое количество обрабатываемых строк одним процессом count и остаточное количество обрабатываемых строк rest, распределяемое по процессам. Далее в root процессе ранее инициализированные массивы rcounts и displs заполняются, в соответствии с распределяемыми буферами. Массив rcounts содержит длины отправляемых буферов данных, тогда как массив displs содержит индексы начала частей буфера.

3 В каждом процессе выделить память для размещения порции массива. Функцией MPI_Scatter или MPI_Scatterv распределить исходный массив(ы) на число процессов, выбранных при запуске программы.

Далее выделяется буфер памяти получаемого массива данных и посредством функции MPI_Scatterv процесс root рассылает соответствующие части массива всем процессам коммуникатора.

5 В каждом процессе выполнить обработку части массива составленной

внешней функцией и разместить результаты в массиве порции или в выходных переменных.

После получения части массива всеми процессами посредством той же самой функции MPI_Scatterv, вызывается метод обработки массива reflect, выполняющий индивидуальное задание.

6 Собрать в главном процессе окончательные результаты (функции MPI_Gather(v) или MPI_Reduce.

Обработав массив, вызывается функция MPI_Gatherv, которая на всех процессах посылает root процессу свой буфер данных, который объединяет части буферов в единый буфер.

7 Вывести окончательные результаты. Для больших выходных массивов достаточно вывести первый и последний элементы.

Окончательные результаты работы параллельной программы выводятся в зависимости от переданного параметра. В данном случае, вывод происходит в файл output.txt, содержимое которого можно посмотреть на рисунке 1.1. Листинг программы представлен в приложении А.1.

labs > labs > output.txt

```
 1
 2   Processor name: cluster
 3   Number of processes: 8
 4
 5   0: INITIAL RANDOM MATRIX
 6   Number of rows: 14
 7   Number of columns: 4
 8   |  -89.9336    78.4108    -19.053   -65.7975
 9   |  -34.2165   -70.7073     93.283   -45.7598
10   |  |59.7741   -18.9527    64.8198   -99.0048
11   |  -96.0457   -53.5152    -67.738   -57.9113
12   |  |13.2769    57.8662   -49.8891      85.87
13   |  -25.8091    18.8373   -40.5093   -89.2307
14   |  -37.3924    11.7564   -15.3928   -6.72411
15   |  |-41.176   -13.4395    24.1318     11.008
16   |  |77.2283    25.3852    -37.347    21.1898
17   |  |38.3678    56.9085    -84.102     87.639
18   |  -15.7846   -14.2578   -30.2808   -59.9605
19   |  |15.9136    70.0936    38.2818   -68.4514
20   |  -1.23096    44.3953   -22.2965   -9.94228
21   |  | 54.133    81.7608    54.3028    89.7422
22
23   0:  --------------------------------
24   Number of rows: 14
25   Number of columns: 4
26   |  -65.7975    -19.053    78.4108   -89.9336
27   |  -45.7598     93.283   -70.7073   -34.2165
28   |  -99.0048    64.8198   -18.9527    59.7741
29   |  -57.9113    -67.738   -53.5152   -96.0457
30   |  | |85.87   -49.8891    57.8662    13.2769
31   |  -89.2307   -40.5093    18.8373   -25.8091
32   |  -6.72411   -15.3928    11.7564   -37.3924
33   |  | 11.008    24.1318   -13.4395    -41.176
34   |  |21.1898    -37.347    25.3852    77.2283
35   |  | 87.639    -84.102    56.9085    38.3678
36   |  -59.9605   -30.2808   -14.2578   -15.7846
37   |  -68.4514    38.2818    70.0936    15.9136
38   |  -9.94228   -22.2965    44.3953   -1.23096
39   |  |89.7422    54.3028    81.7608     54.133
40
41   0: Time of calculation: 1.390650e-309
42   0:  --------------------------------
```

Рисунок 1.1 — Результат работы параллельной программы, выполненной на 8 процессах

Заключение

В результате выполнения лабораторной работы я освоила применение коллективных функций MPI для рассылки и сборки фрагментов массивов и параллельной их обработки по заданному алгоритму.

Приложение А
(обязательное)
Листинг программы

Листинг А.1

```
#pragma once
#include "Process.h"
#include <fstream>
#include <iomanip>
#include <iostream>

#include <random>
#include <algorithm>
#include <iterator>
#include <vector>

template<typename T>
class Matrix: public Process
{
private:
   int rows, columns;
   T* data = NULL;
   double startwtime, endwtime;
   std::ofstream fout;
public:
   Matrix(int argc, char *argv[],
      int _rows=5, int _columns=5,
      MPI_Comm comm = MPI_COMM_WORLD,
      std::string filename = "output.txt")
      : Process(argc, argv, comm)
   {
      fout.open(filename, std::ios::out);
      rows = _rows;
      columns = _columns;
      if (PID==Process::INIT)
      {
         Communicator::printInfo("", fout);
         data = new T[rows * columns];
         fillRandom();
         Process::printInfo("INITIAL RANDOM MATRIX", fout);
         fout << *this;
         Process::printInfo("\t------------------------------", fout);
```

```
        }
        startwtime=MPI_Wtime();
    }
    ~Matrix() { fout.close(); delete data;}

    void fillRandom(T min=-100.0, T max=100.0)
    {
        std::random_device rnd_device;
        std::mt19937 mersenne_engine {rnd_device()};
        std::uniform_real_distribution<T> dist {min, max};

        auto gen = [&dist, &mersenne_engine]()
                {return dist(mersenne_engine);};

        std::generate(data, data + rows * columns, gen);
    }
private:
    static T* reflect(T* array, int len, int cols)
    {
        for(int i = 0; i < len / cols; ++i)
            for(int j = 0; j < cols/2; ++j)
                std::swap(array[i*cols + j],
                        array[i*cols + (cols -1-j)]);
        return array;
    }
public:

    void scatterVec()
    {
        int count = rows / numprocs;
        int rest = rows % numprocs;
        int *displs = new int[numprocs],
            *rcounts = new int[numprocs];
        if (PID==Process::INIT)
            for(int i = 0; i < numprocs; i++)
            {
                rcounts[i] = i < rest ? columns * (count+1) : columns*count;
                displs[i] = displs[i-1] + rcounts[i-1];
            }
        int length = PID < rest ? columns * (count+1) : columns*count;
        int startIndex = PID * length + (PID >= rest ? rest : 0);
        T *partOfArray = new T[length];
```

```
        MPI_Scatterv(data, rcounts, displs, MPI_FLOAT,
                partOfArray, length, MPI_FLOAT, Process::INIT, comm);

        reflect(partOfArray, length, columns);

        MPI_Gatherv(partOfArray, length, MPI_FLOAT,
                data, rcounts, displs, MPI_FLOAT, Process::INIT, comm);
        delete rcounts, displs, partOfArray;

        if (PID == Process::INIT)
        {
            fout << *this;
            std::stringstream str;
            str << std::scientific << "Time of calculation: " << endwtime - startwtime;
            Process::printInfo(str.str(), fout);
            Process::printInfo("\t------------------------------", fout);
            fflush(NULL);
            str.clear();
            str.str("");
        }
    }

    friend std::ostream& operator<<(std::ostream& out, const Matrix<T>& matrix)
    {
        out << "\nNumber of rows: " << matrix.rows
            << "\nNumber of columns: " << matrix.columns
            << "\n";
        for (int i = 0; i < matrix.rows; ++i)
        {
            for (int j = 0; j < matrix.columns; ++j)
                out << std::setw(10)<<matrix.data[i * matrix.columns + j] << " ";
            out << "\n";
        }
        fflush(NULL);
        return out;
    }


};
```