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

Ассистент кафедры АСУ

_____ П.Д. Тихонов

«____»_____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.

Окончательные результаты работы параллельной программы выводятся в зависимости от переданных параметров, задающих количество строк и столбцов матрицы. Вывод осуществляется в файл output.txt, содержимое которого можно посмотреть на рисунке 1.1. Листинг программы представлен в приложении А.1. Результаты работы функции отображения для каждого процесса представлены на рисунке 1.2.

```
1
2   Processor name: cluster
3   Number of processes: 4
4
5   0: INITIAL RANDOM MATRIX
6   Number of rows: 14
7   Number of columns: 4
8     -53.3292   -0.63662    83.2952   -66.4203
9     -94.8369   -97.3451    83.7319   -85.0817
10    -68.8101   -89.0165    62.9939   -35.5925
11    -74.1928    34.1792    7.98651   -54.5685
12    -33.9355    6.72819    97.0509    74.0125
13     49.9717   -96.4518   -97.6992   -19.2637
14    -26.8721    59.7615    20.8299   -87.7208
15    -26.5477   -35.2121   -22.5885     24.035
16    -55.2793    10.5269    -28.924   -98.6288
17    -48.5941    -84.343   -20.4227   -45.2144
18    -60.5124   -46.1291   -27.4324   -87.0172
19    -7.94513   -21.8313    69.9494    99.9711
20     20.4689   -63.5101   -9.24525    81.0814
21     -93.473   -76.5756    27.1987    38.0792
22
23  0:  --------------------------------
24  Number of rows: 14
25  Number of columns: 4
26    -66.4203    83.2952   -0.63662   -53.3292
27    -85.0817    83.7319   -97.3451   -94.8369
28    -35.5925    62.9939   -89.0165   -68.8101
29    -54.5685    7.98651    34.1792   -74.1928
30     74.0125    97.0509    6.72819   -33.9355
31    -19.2637   -97.6992   -96.4518    49.9717
32    -87.7208    20.8299    59.7615   -26.8721
33     24.035   -22.5885   -35.2121   -26.5477
34    -98.6288    -28.924    10.5269   -55.2793
35    -45.2144   -20.4227    -84.343   -48.5941
36    -87.0172   -27.4324   -46.1291   -60.5124
37     99.9711    69.9494   -21.8313   -7.94513
38     81.0814   -9.24525   -63.5101    20.4689
39     38.0792    27.1987   -76.5756    -93.473
40
41  0: Time of calculation: 1.390650e-309
42  0:  --------------------------------
43
```

Рисунок 1.1 — Результат работы параллельной программы, выполненной на 4 процессах

```
laa4302@asu.local@cluster:~/labs/labs> mpirun -np 1 lab1

0:
  -54.6886     -50.22    75.0365   -42.7621
  -5.19571   -16.2705   -23.9465    -37.768
   58.5248   -34.3938   -34.8738   -58.3839
  -52.3707    55.7391   -87.3596    99.5997
   92.2294    69.5923   -51.7169    37.5033
    46.181   -35.7273    47.0077   -55.7999
  -63.1292     9.2407    38.8177   -93.5318
   20.8113    38.9805    52.1895   -58.9167
  -49.3246    19.8304   -49.7252   -2.69147
   2.27873     72.146    -14.717    46.9098
  -70.3272    98.6412    77.1288   -31.8018
   60.1705    31.7019    31.6212    38.4416
   99.9141   -80.9275   -35.3564    -34.615
   95.1189   -23.8406    19.9689   -79.9969
laa4302@asu.local@cluster:~/labs/labs>
```

```
laa4302@asu.local@cluster:~/labs/labs> mpirun -np 4 lab1

0:
  -66.4203    83.2952   -0.63662   -53.3292
  -85.0817    83.7319   -97.3451   -94.8369
  -35.5925    62.9939   -89.0165   -68.8101
  -54.5685    7.98651    34.1792   -74.1928

1:
   74.0125    97.0509    6.72819   -33.9355
  -19.2637   -97.6992   -96.4518    49.9717
  -87.7208    20.8299    59.7615   -26.8721
    24.035   -22.5885   -35.2121   -26.5477

2:
  -98.6288    -28.924    10.5269   -55.2793
  -45.2144   -20.4227    -84.343   -48.5941
  -87.0172   -27.4324   -46.1291   -60.5124

3:
   99.9711    69.9494   -21.8313   -7.94513
   81.0814   -9.24525   -63.5101    20.4689
   38.0792    27.1987   -76.5756    -93.473
```

Рисунок 1.2 — Результат работы функции отражения для каждого процесса

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 — Листинг класса матрицы, осуществляющей отображение элементов строк матрицы в параллельных процессах

```
#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;
    }
};
```