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)

ПРОИЗВОДНЫЕ ТИПЫ ДАННЫХ В MPI

Лабораторная работа №3

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 функций MPI для конструирования производных типов данных и передачи на их основе выбранных частей двумерного массива.

Индивидуальное задание по варианту №:30: в квадратной матрице с нечетным числом строк и столбцов выбрать треугольную часть матрицы с вершиной в центре матрицы и основанием в первой строке.

1 Ход работы

1. Для индивидуального задания выбрать наиболее подходящий способ конструирования производного типа для выборки части массива.

В качестве способа конструирования производного типа данных был выбран индексный способ, так как он позволяет строить тип, в котором промежутки между блоками исходного типа могут иметь нерегулярный характер. Таким образом в качестве функции создания производного типа использовалась функция MPI_Type_indexed().

2. Определить аргументы функции конструирования, их представление. Составить алгоритмы вычисления аргументов конструктора производного типа (векторный тип – размер блока и расстояние между блоками; индексный тип – массивы размеров блоков и расстояний каждого блока от начала массива).

Аргументами функции MPI_Type_indexed() являются:

• count — целочисленное значение количества блоков в производном типе данных. Значение было вычислено как половина количества строк матрицы плюс один;

• array_of_blocklengths[] - целочисленный массив, содержащий количество элементов в каждом блоке. По мере прохождения строк матрицы количество элементов в текущем блоке уменьшается на 2, по сравнению с предыдущим блоком. Иными словами, в зависимости от рассматриваемой строки, количество элементов блока на 2*i (слева и справа по строке) меньше исходного количества элементов в строке, где i — номер текущей строки. Поэтому формула была определена как: columns — (i*2);

• array_of_displacements[] - целочисленный массив, содержащий значения смещения (типа данных oldtype) каждого блока, относительно начала производного типа данных. Индекс начала блока, относительно рассматриваемой строки есть номер этой строки. Но так как мы имеем

матрицу, получившаяся формула выглядит следующий образом: i*columns + i. Упрощая, получаем: i*(columns+1);

• oldtype — MPI_Datatype тип исходных данных. Для матрицы с действительными значениями был выбран тип MPI_FLOAT;

• newtype — новый определяемый тип данных, названный MPI_TRIANGLE.

3. Организовать в программе две матрицы и массив по размерам упакованного представления данных, для размещения в нем выбранных данных.

В классе существует указатель на массив матрицы, который инициализируется в конструкторе нулями по размеру переданного количества строк и столбцов. Второй массив создаётся перед приёмом массива матрицы производного типа данных в нулевом процессе. Для процесса с номером 1 необходимости в создании нового массива нету, так как используется инициализированный в конструкторе нулями массив класса.

4. Составить программу с двумя процессами и использованием производного типа для передачи части исходного массива в другой процесс. Матрицы инициировать нулевыми значениями элементов. В нулевом процессе заполнить одну матрицу целыми числами, начиная с 1. Распечатать ее с указанием ранга процесса. В другом процессе обнулить эту матрицу.

В нулевом процессе в конструкторе матрица заполняется вещественными числами алгоритмом «Вихрь Мерсенна» и выводится в файл output.txt с указанием ранга процесса. Как было сказано выше, в другом процессе матрица инициализируется нулями.

5. Объявить имя производного типа данных. Записать MPI функцию конструирования производного типа, а также функции регистрации и освобождения производного типа.

В методе createDatatype() в каждом процессе объявляется производный тип данных MPI_TRIANGLE с помощью функции MPI_Type_indexed(), далее он регистрируется с помощью функции MPI_Type_commit(). Высвобождение типа осуществляется в функции MPI_Type_free().

6. Используя сконструированный производный тип данных, передать данные из нулевого процесса в матрицу другого процесса. Еще раз послать матрицу с производным типом другому процессу.

В методе selectTriangle() осуществляется передача матрицы типа MPI_TRIANGLE на процесс с рангом 1 два раза. Далее в процессе 1 буфер с рангом 1 был принят и выведен в файл как элемент типа MPI_TRIANGLE.

7. В другом процессе принять посланные данные в массив по размеру выбранных данных базового типа.

Также в процессе с рангом 1 происходит приём посланных данных в массив по размеру выбранных данных базового типа с тегом 2, значение которого вычисляется ещё на этапе создания производного типа.

8. В другом процессе распечатать полученную матрицу и вектор выбранных элементов.

После приёма буфера в этом же процессе осуществляется печать вектора в выходной файл. Матрица, полученная в процессе 1 в виде буфера с тегом 1, ранее уже была напечатана (в пункте 6).

9. Организовать обратную передачу в нулевой процесс вектора выбранных элементов базового типа данных. В нулевом процессе принять эти данные в другую матрицу с использованием производного типа данных.

Далее организуется обратная передача в нулевой процесс вектора выбранных элементов базового типа из процесса с рангом 1. В нулевом процессе массив принимается с производным типом MPI-TRIANGLE.

10. Вывести полученную другую матрицу с указанием ранга процесса.

И наконец, в процессе с рангом 0 выводится полученная матрица типа MPI_TRIANGLE.

2 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

В результате запуска программы на двух процессах, листинг которой представлен в приложении А.1, мы получили результат для процесса 1, который представлен на рисунке 2.1. Как видно из рисунка, в результате передачи матрицы типа MPI_TRIANGLE, на процессе 1 было получено два буфера: матрица типа MPI_TRIANGLE и вектор по размеру выбранных данных базового типа. В свою очередь для процесса 0 была выведена исходная матрица и матрица, полученная от процесса с рангом 1 производного типа данных MPI_TRIANGLE, как можно увидеть на рисунке 2.2.

```
1:
Number of rows: 9
Number of columns: 9
  -24.1189   -33.8995   -25.8726   -13.7168    8.46913   -8.43538    61.3709    30.4728   -19.7908
         0    6.52841   -32.2236    96.2096    59.9735    82.9267     25.257    46.7295          0
         0          0    43.0155    81.5076   -56.7588    94.0392    60.4467          0          0
         0          0          0   -21.1326    19.5765   -71.3125          0          0          0
         0          0          0          0   -97.3185          0          0          0          0
         0          0          0          0          0          0          0          0          0
         0          0          0          0          0          0          0          0          0
         0          0          0          0          0          0          0          0          0
         0          0          0          0          0          0          0          0          0

1:
Number of rows: 9
Number of columns: 9
  -24.1189   -33.8995   -25.8726   -13.7168    8.46913   -8.43538    61.3709    30.4728   -19.7908
   6.52841   -32.2236    96.2096    59.9735    82.9267     25.257    46.7295    43.0155    81.5076
  -56.7588    94.0392    60.4467   -21.1326    19.5765   -71.3125   -97.3185          0          0
         0          0          0          0          0          0          0          0          0
         0          0          0          0          0          0          0          0          0
         0          0          0          0          0          0          0          0          0
         0          0          0          0          0          0          0          0          0
         0          0          0          0          0          0          0          0          0
         0          0          0          0          0          0          0          0          0

Processor name: hiuluzinsan
Number of processes: 3
```

Рисунок 2.1. - Полученная матрица и массив данных на процессе с рангом 1

```
0: INITIAL RANDOM MATRIX
Number of rows: 9
Number of columns: 9
  -24.1189   -33.8995   -25.8726   -13.7168    8.46913   -8.43538    61.3709    30.4728   -19.7908
  -77.7209    6.52841   -32.2236    96.2096    59.9735    82.9267     25.257    46.7295   -26.2674
  -7.05916    7.41842    43.0155    81.5076   -56.7588    94.0392    60.4467   -11.0606   -29.7883
  -11.6511     82.083    36.1528   -21.1326    19.5765   -71.3125   -48.5629   -15.3211   -76.8636
   17.9032    97.0145   -20.4842    86.965   -97.3185    83.2711   -41.3583   -69.0957     22.659
   29.4747   -58.8548    65.9431    5.1806    21.2057    39.9695   -81.6927   -43.5719   -29.4678
  -99.5207   -32.9876    29.8206   -77.656    28.9737   -81.8492    88.0134   -87.5749   -63.0977
  -27.1768   -77.3149    86.4082   -72.925   -27.9007    60.8712    98.2437    57.8585    56.2073
  -6.20261   -51.1873    58.7702   -64.871   -95.2677    12.4759    6.49805    24.9874   -81.2642

0:  --------------------------------
0:
Number of rows: 9
Number of columns: 9
  -24.1189   -33.8995   -25.8726   -13.7168    8.46913   -8.43538    61.3709    30.4728   -19.7908
         0    6.52841   -32.2236    96.2096    59.9735    82.9267     25.257    46.7295          0
         0          0    43.0155    81.5076   -56.7588    94.0392    60.4467          0          0
         0          0          0   -21.1326    19.5765   -71.3125          0          0          0
         0          0          0          0   -97.3185          0          0          0          0
         0          0          0          0          0          0          0          0          0
         0          0          0          0          0          0          0          0          0
         0          0          0          0          0          0          0          0          0
         0          0          0          0          0          0          0          0          0
```

Рисунок 2.2 — Исходная матрица и матрица производного типа данных процесса с рангом 0

Заключение

В результате выполнения лабораторной работы я освоила применение функций MPI для конструирования производных типов данных и передачи на их основе выбранных частей двумерного массива.

Приложение А
(обязательное)
Листинг программы

Листинг А.1 — Листинг класса матрицы с методом создания производного типа

```
#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;
   MPI_Datatype MPI_TRIANGLE;
   int len_type = 0;

public:
   Matrix(int argc, char *argv[],
            int _rows=5, int _columns=5,
            std::string filename =
      "/home/luzinsan/TUSUR_learn/4 курс/7_semester/ПП/labs/lr3/output.txt",
      MPI_Comm comm = MPI_COMM_WORLD)
      : Process(argc, argv, comm)
   {
      fout.open(filename, std::ios::app);
      rows = _rows;
      columns = _columns;
      data = new T[rows * columns]{0};
      if (PID==Process::INIT)
      {
         Communicator::printInfo("", fout);
```

```
        fillRandom();
        Process::printInfo("INITIAL RANDOM MATRIX", fout);
        fout << *this; fflush(NULL);
        Process::printInfo("\t-------------------------------", fout);
    }
    createDatatype();
    startwtime=MPI_Wtime();
}

~Matrix() {
    fout.close();
    delete data;
    MPI_Type_free(&MPI_TRIANGLE);
}

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);
}

void printInfo(std::string accompanying_message = "",
                std::ostream &out = std::cout, T* buf=NULL, int length=0)
{
    Process::printInfo("\n", out);
    if (buf)
        for(int i = 0; i < length/columns; i++)
        {
            for(int j = 0; j < columns; j++)
                out << std::setw(10) << buf[i*columns + j] << " ";
            out << "\n"; fflush(NULL);
        }
    fflush(NULL);
}

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] << " ";
fflush(NULL);}
            out << "\n";
        }
        fflush(NULL);
        return out;
    }

private:
    void createDatatype()
    {
        int length = (rows>>1) + 1;
        int blocklens[length], indices[length];
        for (int i=0; i<length; i++)
        {
            blocklens[i] = columns - (i<<1);
            len_type += blocklens[i];
            indices[i]   = i*(columns + 1);
        }
        MPI_Type_indexed(length, blocklens, indices,
                    MPI_FLOAT, &MPI_TRIANGLE);
        MPI_Type_commit(&MPI_TRIANGLE);
    }
public:
    void selectTriangle()
    {
        switch(PID){
            case Process::INIT:
                send(data, 1, MPI_TRIANGLE, 1, 1);
                send(data, 1, MPI_TRIANGLE, 1, 2);
                delete[] data;
                data = new T[rows * columns]{0};
                receive(data, 1, MPI_TRIANGLE);
                Process::printInfo("", fout);
                fout << *this; fflush(NULL);
                break;
            case 1:
                receive(data, Process::INIT, MPI_TRIANGLE, 1, 1);
```

```
            Process::printInfo("", fout);
            fout << *this; fflush(NULL);
            delete[] data;
            data = new T[rows * columns]{0};
            receive(data, Process::INIT, MPI_FLOAT, len_type, 2);
            Process::printInfo("", fout);
            fout << *this; fflush(NULL);
            send(data, Process::INIT, MPI_FLOAT, len_type);
            break;
        }
    }
};
```