МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«**Вятский государственный университет**»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Отчет по лабораторной работе №4

по дисциплине «Параллельное программирование»

Вариант 10

Выполнил студент группы ИВТ-32 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Караваев П.А./

Проверил преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Чистяков Г.А./

Киров 2019

Задание на лабораторную работу

1. Изучить основные принципы работы с интерфейсом MPI, освоить механизм передачи сообщений между процессами.
2. Выделить в полученной в ходе первой лабораторной работы реализации алгоритма фрагменты кода, выполнение которых может быть разнесено на несколько процессоров.
3. Реализовать многопоточную версию алгоритма с помощью языка C++ и библиотеки MPICH, используя при этом предлагаемые интерфейсом MPI механизмы и виртуальные топологии (в случае при применимости).
4. Показать корректность полученной реализации путем осуществления тестирования на построенном в ходе первой лабораторной работы наборе тестов
5. Провести доказательную оценку эффективности MPI-реализации алгоритма.

Словесное описание процесса выделения распараллеливаемых фрагментов

В реализованном алгоритме есть лишь одна процедура, распараллеливание которой может принести выигрыш в скорости нахождения решения поставленной задачи. Данный участок кода отвечает за порождение новых состояний игрового поля за счет сдвига пустой фишки во всех возможных направлениях.

Таким образом, основной процесс отвечает за взаимодействие с пользователем через консоль, а непосредственно решение задачи выполняется несколькими процессами. Количество порождаемых состояний может варьироваться от 2 до 4 включительно, в зависимости от местоположения пустой фишки.

Логичным способом распараллеливания является одновременное выполнение сдвига фишка во всех возможных направлениях и запись новых порожденных состояний в дерево.

Каждый процесс порождает новое состояние из изначального путем сдвига пустой фишки в одну из сторон, а затем в каждом процессе происходит поиск решения задачи на основе порожденного состояния.

Как только в одном из процессов было найдено решение, об этом информируются другие процессы и поиск решения завершается.

Исходное состояние сгенерированного игрового поля передается другим процессам из основного с помощью функции MPI\_Bcast.

Процессы сигнализируют о нахождении ответа с помощью функций асинхронной передачи MPI\_Isend, приём с помощью функции асинхронного приёма MPI\_Irecv.

Для синхронизации процессов используются функция MPI\_Barrier и механизм сообщений.

Схема взаимодействия процессов

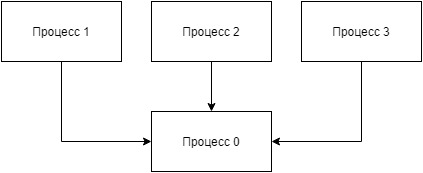


Рисунок 1 - Схема взаимодействия процессов

Листинг программы

#include <mpi.h>

#include <conio.h>

#include <iostream>

#include <cmath>

#include <vector>

#include "Map.h"

#include "BinTree.h"

#include "State.h"

#include <chrono>

#include <mutex>

using namespace std;

#define TESTS 10000

using namespace std;

bool check(Map\*);

void printMap(Map\*);

//Генератор игрового поля

Map\* generateMap(int lines, int cols) {

int len = lines \* cols;

Map\* map = new Map(lines, cols);

for (int i = 0; i < len; ++i)

{

map->map[i] = i + 1;

}

map->map[len - 1] = 0;

int i = 0;

int shift\_pos;

srand(time(0));

while (i <= len\*20) {

//Находим пустую клетку

int zero = map->find(0);

shift\_pos = rand() % 4;

switch (shift\_pos) {

case 0:

//Если свехру есть квадрат

if (zero / map->getCols() != 0) {

map = map->shift(shift\_pos);

i++;

}

continue;

case 1:

//Если справа есть квадрат

if (zero % map->getCols() != map->getCols() - 1) {

map = map->shift(shift\_pos);

i++;

}

continue;

case 2:

//Если снизу есть квадрат

if (zero / map->getCols() != map->getLines() - 1) {

map = map->shift(shift\_pos);

i++;

}

continue;

case 3:

//Если слева есть квадрат

if (zero % map->getCols() != 0) {

map = map->shift(shift\_pos);

i++;

}

continue;

}

}

return map;

}

// Функция для вывода матрицы на экран

void printMap(Map\* map) {

cout << endl;

for (int i = 0; i < map->lines; ++i) {

for (int j = 0; j < map->cols; ++j) {

cout << map->map[i \* map->cols + j] << '\t';

}

cout << endl;

}

cout << endl;

}

/////////////////////////////////////////////////////////ПАРАЛЛЕЛЬНЫЙ НОВЫЙ/////////////////////////////////////////////////////////

vector<State\*> resultP2;

vector<State\*> thread\_func2(Map\* map, State\* min, BinTree\* close, BinTree\* open, int flag\_active\_arr[4], int rank) {

vector<State\*> lol;

int flag\_solution[4] = { 0, 0, 0, 0 };

MPI\_Request request[4];

MPI\_Request request0;

MPI\_Request request1;

MPI\_Request request2;

MPI\_Request request3;

int flag\_request[4] = { 0, 0, 0, 0 };

MPI\_Status status0;

MPI\_Status status1;

MPI\_Status status2;

MPI\_Status status3;

if (flag\_active\_arr[0] == 1)

MPI\_Irecv(&flag\_solution[0], 1, MPI\_INT, 0, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &request0);

if (flag\_active\_arr[1] == 1)

MPI\_Irecv(&flag\_solution[1], 1, MPI\_INT, 1, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &request1);

if (flag\_active\_arr[2] == 1)

MPI\_Irecv(&flag\_solution[2], 1, MPI\_INT, 2, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &request2);

if (flag\_active\_arr[3] == 1)

MPI\_Irecv(&flag\_solution[3], 1, MPI\_INT, 3, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &request3);

//Пока нет состояния в котором расстояние равно 0 (признак упорядоченности)

while ((flag\_request[0] == 0) && (flag\_request[1] == 0) && (flag\_request[2] == 0) && (flag\_request[3] == 0))

{

if (flag\_active\_arr[0] == 1)

MPI\_Test(&request0, &flag\_request[0], &status0);

if (flag\_active\_arr[1] == 1)

MPI\_Test(&request1, &flag\_request[1], &status1);

if (flag\_active\_arr[2] == 1)

MPI\_Test(&request2, &flag\_request[2], &status2);

if (flag\_active\_arr[3] == 1)

MPI\_Test(&request3, &flag\_request[3], &status3);

if ((flag\_request[0] == 0) && (flag\_request[1] == 0) && (flag\_request[2] == 0) && (flag\_request[3] == 0))

{

min = open->min();

close->add(min);

open->del(min);

//Находим пустую клетку

int zero = min->getMap()->find(0);

//Если сверху есть квадрат

if (zero / map->getCols() != 0) {

State\* s = new State(min->getMap()->shift(0), min);

if ((open->find(s) == NULL) && (close->find(s) == NULL)) {

open->add(s);

}

}

//Если справа есть квадрат

if (zero % map->getCols() != map->getCols() - 1) {

State\* s = new State(min->getMap()->shift(1), min);

if ((open->find(s) == NULL) && (close->find(s) == NULL)) {

open->add(s);

}

}

//Если снизу есть квадрат

if (zero / map->getCols() != map->getLines() - 1) {

State\* s = new State(min->getMap()->shift(2), min);

if ((open->find(s) == NULL) && (close->find(s) == NULL)) {

open->add(s);

}

}

//Если слева есть квадрат

if (zero % map->getCols() != 0) {

State\* s = new State(min->getMap()->shift(3), min);

if ((close->find(s) == NULL) && (open->find(s) == NULL)) {

open->add(s);

}

}

if (min->getCost() == 0)

{

flag\_solution[rank] = 1;

if (flag\_active\_arr[0] == 1)

MPI\_Isend(&flag\_solution[rank], 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &request0);

if (flag\_active\_arr[1] == 1)

MPI\_Isend(&flag\_solution[rank], 1, MPI\_INT, 1, 1, MPI\_COMM\_WORLD, &request1);

if (flag\_active\_arr[2] == 1)

MPI\_Isend(&flag\_solution[rank], 1, MPI\_INT, 2, 2, MPI\_COMM\_WORLD, &request2);

if (flag\_active\_arr[3] == 1)

MPI\_Isend(&flag\_solution[rank], 1, MPI\_INT, 3, 3, MPI\_COMM\_WORLD, &request3);

flag\_solution[rank] = 1;

}

}

}

return lol;

}

vector<State\*> aPar2(Map\* map, int rank) {

int flag\_active\_arr[4] = { 0,0,0,0 };

//Создаем открытый и закрытый список для состояний

BinTree\* open = NULL;

BinTree\* close = NULL;

BinTree\* closekek = new BinTree(new State(map, NULL));

State\* min = closekek->min();

//Находим пустую клетку

int zero = min->getMap()->find(0);

int index = 0;

//Если сверху есть квадрат

if (zero / map->getCols() != 0) {

flag\_active\_arr[0] = 1;

if (rank == 0)

{

open = new BinTree(new State(min->getMap()->shift(0), NULL));

close = new BinTree();

}

}

//Если справа есть квадрат

if (zero % map->getCols() != map->getCols() - 1) {

flag\_active\_arr[1] = 1;

if (rank == 1)

{

open = new BinTree(new State(min->getMap()->shift(1), NULL));

close = new BinTree();

}

}

//Если снизу есть квадрат

if (zero / map->getCols() != map->getLines() - 1) {

flag\_active\_arr[2] = 1;

if (rank == 2)

{

open = new BinTree(new State(min->getMap()->shift(2), NULL));

close = new BinTree();

}

}

//Если слева есть квадрат

if (zero % map->getCols() != 0) {

flag\_active\_arr[3] = 1;

if (rank == 3)

{

open = new BinTree(new State(min->getMap()->shift(3), NULL));

close = new BinTree();

}

}

//MPI\_Allgather(&flag\_active\_arr[rank], 1, MPI\_INT, flag\_active\_arr, 1, MPI\_INT, MPI\_COMM\_WORLD);

if (flag\_active\_arr[rank] == true)

{

resultP2 = thread\_func2(open->min()->getMap(), open->min(), close, open, flag\_active\_arr, rank);

}

//else

//resultP2 = thread\_func2(NULL, NULL, NULL, NULL, flag\_active\_arr, rank);

return resultP2;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int main(int argc, char\*\* argv)

{

int rank, size;

int lines = 0;

int cols = 0;

Map\* map;

vector<State\*> ans;

long double tParNew = 0;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

do {

if (rank == 0)

{

system("cls");

cout << "Enter field sizes: " << endl;

cin >> lines >> cols;

}

MPI\_Barrier(MPI\_COMM\_WORLD);

MPI\_Bcast(&lines, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&cols, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (rank == 0)

{

cin.clear();

if (cin.good() == false || cols < 2 || lines < 2) {

system("cls");

cout << "The size of the field is wrong" << endl;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

\_getch();

}

}

} while (cols \* lines <= 4);

int i = 0;

for (i; i < TESTS; i++) {

srand(i);

map = generateMap(lines, cols);

if (rank == 0)

{

cout << "\n" << "-----------------------------------------------------";

cout << "\n" << "Case #" << i + 1 << ": ";

printMap(map);

}

//Решение новым параллельным

//MPI\_Barrier(MPI\_COMM\_WORLD);

//MPI\_Bcast(&(map->lines), 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

//MPI\_Bcast(&(map->cols), 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(map->map, lines \* cols, MPI\_INT, 0, MPI\_COMM\_WORLD);

chrono::high\_resolution\_clock::time\_point t11 = chrono::high\_resolution\_clock::now();

ans = aPar2(map, rank);

chrono::high\_resolution\_clock::time\_point t22 = chrono::high\_resolution\_clock::now();

MPI\_Barrier(MPI\_COMM\_WORLD);

long double duration2 = chrono::duration\_cast<chrono::milliseconds>(t22 - t11).count();

if (rank == 0)

cout << "\n" << "Time of NEW PARALLEL = " << duration2;

tParNew += duration2;

}

if (rank == 0)

{

cout << "\n" << "--------------" << endl;

cout << "Average time NEW PARALLEL = " << (tParNew / (double)TESTS) << endl;

cout << "--------------" << endl;

system("pause");

}

MPI\_Barrier(MPI\_COMM\_WORLD);

MPI\_Finalize();

return 0;

}

Результаты тестирования

Таблица 1 – Сравнение алгоритмов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размерность головоломки | Среднее время решения выборки из 1000 головоломок | | | |
| Последовательный | Параллельный | OpenMP | MPI |
| 2х3 | 0,062540 | 0,619025 | 0,43 | 0,02 |
| 3х3 | 1,457634 | 0,651726 | 0,44 | 0,88 |
| 3х4 | 3,155734 | 0,812236 | 0,32 | 1,23 |
| 4х4 | 1,637676 | 0,749886 | 0,55 | 1,09 |
| 4х5 | 2,005796 | 0,820880 | 1,35 | 1,42 |
| 5х5 | 16,969815 | 1,983010 | 2,235 | 4,8 |

Вывод

В ходе выполнения лабораторной работы были изучены принципы создания приложения с использованием MPI. Были рассмотрены механизмы передачи сообщений между процессами.

Были выделены участки кода, выполнение которых может быть разнесено на несколько процессоров. На их основе был разработан вариант параллельного алгоритма с использованием MPI на языке C++.

Разработанный алгоритм хорошо показал себя во время тестирования. Почти на всех размерностях исходного игрового поля он превосходит последовательную версию алгоритма, но проигрывает по времени всем остальным реализациям.