МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«**Вятский государственный университет**»

**(«ВятГУ»)**

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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Щесняк Д. С./

Проверил доцент кафедры ЭВМ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Чистяков Г.А./

Киров 2017

1. Задание на лабораторную работу

Познакомиться со стандартом MPI, получить навыки реализации многопоточных приложений с применением MPI.

1. Изучить основные принципы создания приложений с использованием библиотеки MPI.
2. Выделить в полученной в ходе первой лабораторной работы реализации алгоритма фрагменты кода, выполнение которых может быть разнесено на несколько процессорных ядер.
3. Реализовать многопоточную версию алгоритма с помощью языка С++ и библиотеки MPI, использую при этом необходимые примитивы синхронизации.
4. Показать корректность полученной реализации путем осуществления на построенном в ходе первой лабораторной работы наборе тестов.
5. Провести доказательную оценку эффективность MPI-реализации алгоритма.
6. Разработка параллельного алгоритма

Основная идея распараллеливания при помощи MPI заключается в том, что поиск решения головоломки идет с двух сторон: от собранной комбинации к заданной и от заданной к собранной. На каждом шаге программы обмениваются текущими вершинами и проверяют, могут ли они добраться до заданной вершины, если могут, то составляется путь, который и будет являться решением головоломки.

1. Исходный код параллельного алгоритма

Исходный код параллельного алгоритма представлен на рисунке 1

|  |
| --- |
| #include <mpi.h>  #include "binTree.h"  #include <vector>  #include <set>  #include <iostream>  void t(State\* min, BinTree\* open, BinTree\* close, std::multiset<State\*, compare>\* hashes, int i, Map\* target = NULL) {  State\* s;    if (target) s = new State(min->getMap()->shift(i), min, target);  else s = new State(min->getMap()->shift(i), min);  for (std::multiset<State\*, compare>::iterator iter = hashes->find(s); iter != hashes->end(); iter++) {  if (cmp((\*iter), s)) {  return;  }  }  open->add(s);  hashes->insert(s);  }  std::vector<Map\*> aMPI(Map\* map) {  Map\* one = ones(map->lines, map->cols);    int rank;  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  std::multiset<State\*, compare> hashes;  if (rank == 0) {  BinTree\* open = new BinTree();  BinTree\* close = new BinTree(new State(map, NULL, one));    State\* min = close->min();  for (; ; min = open->min(), close->add(min), open->del(min))  {  MPI\_Status stat;  int\* m = new int[map->lines \* map->cols];  int\* sendM = min->getMap()->map;  MPI\_Send(sendM, map->lines \* map->cols, MPI\_INT, 1, 1, MPI\_COMM\_WORLD);  MPI\_Recv(m, map->lines \* map->cols, MPI\_INT, 1, 1, MPI\_COMM\_WORLD, &stat);  Map\* resMap = new Map(map->lines, map->cols, m);  State\* resState = new State(resMap, NULL);  bool f = false;  bool resF = false;  for (std::multiset<State\*, compare>::iterator iter = hashes.find(resState); iter != hashes.end(); iter++) {  if (cmp(resState, (\*iter))) {  min = (\*iter);  f = true;  break;  }  }  MPI\_Send(&f, 1, MPI\_C\_BOOL, 1, 1, MPI\_COMM\_WORLD);  MPI\_Recv(&resF, 1, MPI\_C\_BOOL, 1, 1, MPI\_COMM\_WORLD, &stat);  if (f || resF) break;  int zero = min->getMap()->find(0);  bool v[4] = { false, false, false, false };  if (zero / map->getCols() != 0) v[0] = true;  if (zero % map->getCols() != map->getCols() - 1) v[1] = true;  if (zero / map->getCols() != map->getLines() - 1) v[2] = true;  if (zero % map->getCols() != 0) v[3] = true;  for (int i = 0; i < 4; i++) {  if (v[i]) t(min, open, close, &hashes, i);  }  }  std::vector<Map\*> vect;  State\* s = min;  do  {  vect.push\_back(s->getMap());  s = s->getParent();  } while (s != NULL);  std::vector<Map\*> solution;  for (int i = (vect.size() - 1); i >= 0; i--) {  solution.push\_back(vect[i]);  }  MPI\_Status stat;  int size;  MPI\_Recv(&size, 1, MPI\_INT, 1, 1, MPI\_COMM\_WORLD, &stat);  int\* m = new int[size];  MPI\_Recv(m, size, MPI\_INT, 1, 1, MPI\_COMM\_WORLD, &stat);  int step = (map->cols \* map->lines);  for (int i = 0; i < size ; i+= step) {  Map\* elem = new Map(map->lines, map->cols, &m[i]);  solution.push\_back(elem);  }  return solution;  }  else {  BinTree\* open = new BinTree();  State\* min = new State(one, NULL, map);  for (; ; min = open->min(), /\*close->add(min),\*/ open->del(min))  {  MPI\_Status stat;  int\* m = new int[map->lines \* map->cols];  int\* sendM = min->getMap()->map;  MPI\_Send(sendM, map->lines \* map->cols, MPI\_INT, 0, 1, MPI\_COMM\_WORLD);  MPI\_Recv(m, map->lines \* map->cols, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, &stat);  Map\* resMap = new Map(map->lines, map->cols, m);  State\* resState = new State(resMap, NULL);  bool f = false;  bool resF = false;  for (std::multiset<State\*, compare>::iterator iter = hashes.find(resState); iter != hashes.end(); iter++) {  if (cmp(resState, (\*iter))) {  min = (\*iter);  f = true;  break;  }  }  MPI\_Send(&f, 1, MPI\_C\_BOOL, 0, 1, MPI\_COMM\_WORLD);  MPI\_Recv(&resF, 1, MPI\_C\_BOOL, 0, 1, MPI\_COMM\_WORLD, &stat);  if (f || resF) break;  int zero = min->getMap()->find(0);  bool v[4] = { false, false, false, false };  if (zero / map->getCols() != 0) v[0] = true;  if (zero % map->getCols() != map->getCols() - 1) v[1] = true;  if (zero / map->getCols() != map->getLines() - 1) v[2] = true;  if (zero % map->getCols() != 0) v[3] = true;  for (int i = 0; i < 4; i++) {  if (v[i]) t(min, open, NULL, &hashes, i, resMap);  }  }  std::vector<Map\*> solution;  State\* s = min;  do  {  solution.push\_back(s->getMap());  s = s->getParent();  } while (s != NULL);  int size = solution.size() \* map->cols \* map->lines;  MPI\_Send(&size, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD);  int\* m = new int[size];  unsigned i = 0;  for (auto elem : solution) {  for (int j = 0; j < (map->cols \* map->lines); j++) {  m[i] = elem->map[j];  i++;  }  }  MPI\_Send(m, size, MPI\_INT, 0, 1, MPI\_COMM\_WORLD);  }  return std::vector<Map\*>();  } |

Рисунок 1 – Исходный код параллельного алгоритма

1. Тестирование

Данные тесты проводились на ЭВМ под управлением 64-х разрядной операционной системы Windows 10, с 4 Гб оперативной памяти, с 4-х ядерным процессором Intel N3530 c частотой 2.2 ГГц.

Тестирование параллельного и линейного алгоритма представлено в таблице 1. Экранные формы выполнения программы представлены на рисунке 2.

Таблица 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тест | Линейное время (сек) | Параллельное время (сек) | Ускорение |
| 1 | |  |  |  |  | | --- | --- | --- | --- | | 13 | 11 | 10 | 9 | | 8 | 6 | 5 | 12 | | 2 | 4 | 3 | 14 | | 7 | 15 | 1 |  | | 0.533 | 0.426 | 1.25 |
| 2 | |  |  |  |  | | --- | --- | --- | --- | | 14 | 6 | 9 | 12 | | 2 | 5 | 11 | 15 | | 1 | 3 | 8 | 10 | | 13 | 7 | 4 |  | | 0.399 | 0.31 | 1.29 |
| 3 | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 4 | 13 | 6 | 11 | 8 | | 24 | 3 | 22 | 23 | 7 | | 2 | 10 | 16 | 21 | 18 | | 14 | 15 | 17 | 9 | 1 | | 12 | 20 | 5 | 19 |  | | 40.527 | 38.447 | 1.054 |
| 4 | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 16 | 1 | 18 | 11 | 3 | | 12 | 20 | 9 | 22 | 2 | | 19 | 21 | 5 | 17 | 13 | | 7 | 15 | 4 | 14 | 23 | | 6 | 24 | 10 | 8 |  | | 8.725 | 7.023 | 1.24 |
| 5 | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 5 | 12 | 11 | 18 | 2 | | 15 | 6 | 4 | 17 | 7 | | 16 | 1 | 22 | 21 | 19 | | 14 | 3 | 23 | 24 | 10 | | 13 | 8 | 20 | 9 |  | | 7.567 | 6.136 | 1.23 |
| 6 | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 6 | 18 | 5 | 10 | 2 | | 14 | 8 | 22 | 17 | 7 | | 20 | 11 | 19 | 13 | 4 | | 21 | 3 | 1 | 24 | 15 | | 23 | 9 | 16 | 12 |  | | 7.904 | 6.977 | 1.13 |
| 7 | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 17 | 1 | 2 | 6 | 21 | | 13 | 18 | 7 | 4 | 16 | | 15 | 14 | 10 | 20 | 24 | | 8 | 19 | 5 | 22 | 3 | | 12 | 11 | 23 | 9 |  | | 2.398 | 1.85 | 1.29 |
| 8 | |  |  |  |  | | --- | --- | --- | --- | | 12 | 6 | 13 | 11 | | 15 | 9 | 8 | 2 | | 3 | 14 | 7 | 5 | | 4 | 10 | 1 |  | | 0.389 | 0.286 | 1.36 |
| 9 | |  |  |  | | --- | --- | --- | | 3 | 10 | 2 | | 7 | 6 | 4 | | 1 | 5 | 8 | | 11 | 9 |  | | 1.332 | 1.155 | 1.15 |
| 10 | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 18 | 15 | 1 | 8 | 9 | | 6 | 19 | 13 | 2 | 11 | | 5 | 16 | 3 | 10 | 12 | | 4 | 17 | 7 | 14 |  | | 5.354 | 4.376 | 1.22 |

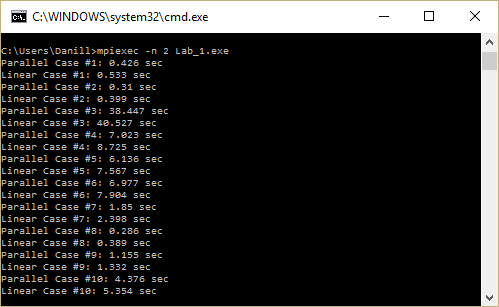


Рисунок 2 – Время прохождения тестов

1. Вывод

В ходе выполнения лабораторной работы был изучен стандарт MPI, его применение и функции. На основе знаний, полученных в ходе лекционного материала по MPI, был разработан параллельный алгоритм поиска решения игры в 15.

Параллельная реализация не принесла существенного прироста в скорости, но если к полученному алгоритму применить другие библиотеки для распараллеливания, то возможно получить существенно ускорение поиска решения. Например, благодаря MPI реализации алгоритма, появилась возможность искать решение головоломки на 2-х компьютерах. Если использовать для распараллеливания openMP, то теоретическое ускорение на одном компьютере будет равно 1.5, а итоговое ускорение 1.5 \* 1.5 = 2.25.