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 №2

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

Необходимо реализовать алгоритм из лабораторной работы №1 в виде параллельного алгоритма, который позволяет найти разрешающую последовательность ходов в пятнашках произвольной размерности.

1. Представить последовательный алгоритм из лабораторной работы №1 в виде параллельного алгоритма для решения поставленной задачи.
2. Реализовать алгоритм с помощью языка С++, используя потоки.
3. Построить набор тестовых примеров и сравнить эффективность работы разработанного параллельного алгоритма с последовательным алгоритмом из лабораторной работы №1.

Описание алгоритма

Были составлены два варианта параллельных алгоритмов на основе алгоритма из лабораторной работы №1.

* Параллельный 1: предполагает параллельное вычисление всех соседних состояний из состояния, являющегося текущим. В остальном его работа ничем не отличается от работы алгоритма из лабораторной работы №1.
* Параллельный 2: предполагает нахождение из начального состояния всех соседних состояний. Данные состояния становятся начальными для каждого из N потоков, каждый из которых реализует работу последовательного алгоритма из лабораторной работы №1. Таким образом, обработка ветвей дерева состояний игрового поля производится параллельно. Как только один из потоков найдет разрешающую последовательность ходов, все потоки завершают свою работу и выдается ответ.

Листинг программ

1. #include <conio.h>
2. #include <iostream>
3. #include <cstdlib>
4. #include <cmath>
5. #include <ctime>
6. #include <vector>
7. #include "Map.h"
8. #include "BinTree.h"
9. #include "State.h"
10. #include <ctime>
11. #include <chrono>
12. #include "mingw.mutex.h"
13. #include "mingw.thread.h"
14. #define TESTS 300
15. using namespace std;
16. bool check(Map\*);
17. void printMap(Map\*);
18. //Генератор игрового поля
19. Map\* generateMap(int lines, int cols) {
20. int len = lines \* cols;
21. Map\* map = new Map(lines, cols);
22. for (int i = 0; i < len; ++i)
23. {
24. map->map[i] = i + 1;
25. }
26. map->map[len - 1] = 0;
27. int i = 0;
28. int shift\_pos;
29. srand(time(0));
30. while (i <= len \* 20) {
31. //Находим пустую клетку
32. int zero = map->find(0);
33. shift\_pos = rand() % 4;
34. switch (shift\_pos) {
35. case 0:
36. //Если свехру есть квадрат
37. if (zero / map->getCols() != 0) {
38. map = map->shift(shift\_pos);
39. i++;
40. }
41. continue;
42. case 1:
43. //Если справа есть квадрат
44. if (zero % map->getCols() != map->getCols() - 1) {
45. map = map->shift(shift\_pos);
46. i++;
47. }
48. continue;
49. case 2:
50. //Если снизу есть квадрат
51. if (zero / map->getCols() != map->getLines() - 1) {
52. map = map->shift(shift\_pos);
53. i++;
54. }
55. continue;
56. case 3:
57. //Если слева есть квадрат
58. if (zero % map->getCols() != 0) {
59. map = map->shift(shift\_pos);
60. i++;
61. }
62. continue;
63. }
64. }
65. return map;
66. }
67. // Функция для вывода матрицы на экран
68. void printMap(Map\* map) {
69. cout << endl;
70. for (int i = 0; i < map->lines; ++i) {
71. for (int j = 0; j < map->cols; ++j) {
72. cout << map->map[i\*map->cols + j] << '\t';
73. }
74. cout << endl;
75. }
76. cout << endl;
77. }
78. /////////////////////////////////////////////////////////ПОСЛЕДОВАТЕЛЬНЫЙ АЛГОРИТМ/////////////////////////////////////////////////////////
79. //Функция поиска решения головоломки
80. vector<State\*> a(Map\* map) {
81. //Создаем открытый и закрытый список для состояний
82. BinTree open = BinTree(new State(map, NULL));
83. BinTree close = BinTree();
84. State\* min = open.min();
85. close.add(min);
86. open.del(min);
87. //Пока нет состояния в котором расстояние равно 0 (признак упорядоченности)
88. for (; min->getCost() != 0; min = open.min(), close.add(min), open.del(min))
89. {
90. //Находим пустую клетку
91. int zero = min->getMap()->find(0);
92. //Если свехру есть квадрат
93. if (zero / map->getCols() != 0) {
94. State\* s = new State(min->getMap()->shift(0), min);
95. if ((open.find(s) == NULL) && (close.find(s) == NULL)) {
96. open.add(s);
97. }
98. }
99. //Если справа есть квадрат
100. if (zero % map->getCols() != map->getCols() - 1) {
101. State\* s = new State(min->getMap()->shift(1), min);
102. if ((open.find(s) == NULL) && (close.find(s) == NULL)) {
103. open.add(s);
104. }
105. }
106. //Если снизу есть квадрат
107. if (zero / map->getCols() != map->getLines() - 1) {
108. State\* s = new State(min->getMap()->shift(2), min);
109. if ((open.find(s) == NULL) && (close.find(s) == NULL)) {
110. open.add(s);
111. }
112. }
113. //Если слева есть квадрат
114. if (zero % map->getCols() != 0) {
115. State\* s = new State(min->getMap()->shift(3), min);
116. if ((close.find(s) == NULL) && (open.find(s) == NULL)) {
117. open.add(s);
118. }
119. }
120. }
121. vector<State\*> solution;
122. State\* s = min;
123. do
124. {
125. solution.push\_back(s);
126. s = s->getParent();
127. } while (s != NULL);
128. return solution;
129. }
130. /////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
131. /////////////////////////////////////////////////////////ПАРАЛЛЕЛЬНЫЙ НОВЫЙ/////////////////////////////////////////////////////////
132. std::mutex flag\_mutex;
133. bool flag\_solution = false;
134. vector<State\*> resultP2;
135. mutex resultP2\_mutex;
136. vector<State\*> thread\_func2(Map\* map, State\* min, BinTree\* close, BinTree\* open) {
137. vector<State\*> lol;
138. //Пока нет состояния в котором расстояние равно 0 (признак упорядоченности)
139. for (; min->getCost() != 0; min = open->min(), close->add(min), open->del(min))
140. {
141. //Находим пустую клетку
142. int zero = min->getMap()->find(0);
143. //Если свехру есть квадрат
144. if (zero / map->getCols() != 0) {
145. State\* s = new State(min->getMap()->shift(0), min);
146. if ((open->find(s) == NULL) && (close->find(s) == NULL)) {
147. open->add(s);
148. }
149. }
150. //Если справа есть квадрат
151. if (zero % map->getCols() != map->getCols() - 1) {
152. State\* s = new State(min->getMap()->shift(1), min);
153. if ((open->find(s) == NULL) && (close->find(s) == NULL)) {
154. open->add(s);
155. }
156. }
157. //Если снизу есть квадрат
158. if (zero / map->getCols() != map->getLines() - 1) {
159. State\* s = new State(min->getMap()->shift(2), min);
160. if ((open->find(s) == NULL) && (close->find(s) == NULL)) {
161. open->add(s);
162. }
163. }
164. //Если слева есть квадрат
165. if (zero % map->getCols() != 0) {
166. State\* s = new State(min->getMap()->shift(3), min);
167. if ((close->find(s) == NULL) && (open->find(s) == NULL)) {
168. open->add(s);
169. }
170. }
171. flag\_mutex.lock();
172. if (flag\_solution == true)
173. {
174. flag\_mutex.unlock();
175. return lol;
176. }
177. flag\_mutex.unlock();
178. }
179. flag\_mutex.lock();
180. flag\_solution = true;
181. flag\_mutex.unlock();
182. State\* s = min;
183. vector <State\*> solution;
184. do
185. {
186. solution.push\_back(s);
187. s = s->getParent();
188. } while (s != NULL);
190. resultP2\_mutex.lock();
191. resultP2 = solution;
192. resultP2\_mutex.unlock();
193. return lol;
194. }
195. vector<State\*> aPar2(Map\* map) {
196. //Создаем открытый и закрытый список для состояний
197. BinTree\* open = new BinTree();
198. BinTree\* close = new BinTree(new State(map, NULL));
199. State\* min = close->min();
200. vector<std::thread> threads;
201. vector<BinTree\*> open\_branch;
202. vector<BinTree\*> close\_branch;
203. //Находим пустую клетку
204. int zero = min->getMap()->find(0);
205. int index = 0;
206. //Если свехру есть квадрат
207. if (zero / map->getCols() != 0) {
208. open\_branch.emplace\_back(new BinTree(new State(min->getMap()->shift(0), NULL)));
209. close\_branch.emplace\_back(new BinTree());
210. }
211. //Если справа есть квадрат
212. if (zero % map->getCols() != map->getCols() - 1) {
213. open\_branch.emplace\_back(new BinTree(new State(min->getMap()->shift(1), NULL)));
214. close\_branch.emplace\_back(new BinTree());
215. }
216. //Если снизу есть квадрат
217. if (zero / map->getCols() != map->getLines() - 1) {
218. open\_branch.emplace\_back(new BinTree(new State(min->getMap()->shift(2), NULL)));
219. close\_branch.emplace\_back(new BinTree());
220. }
221. //Если слева есть квадрат
222. if (zero % map->getCols() != 0) {
223. open\_branch.emplace\_back(new BinTree(new State(min->getMap()->shift(3), NULL)));
224. close\_branch.emplace\_back(new BinTree());
225. }
226. for (int i = 0; abs(i) < open\_branch.size()-1; i++) {
227. threads.emplace\_back(thread\_func2, open\_branch[i]->min()->getMap(), open\_branch[i]->min(), close\_branch[i], open\_branch[i]);
228. }
229. for (auto &thread\_ : threads) {
230. thread\_.join();
231. }
232. return resultP2;
234. }
235. //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
236. int main(int argc, char const \*argv[]) {
237. int lines, cols;
238. Map\* map;
239. do {
240. system("cls");
241. cout << "Enter field sizes: " << endl;
242. cin >> lines >> cols;
243. cin.clear();
244. if (cin.good() == false || cols < 2 || lines < 2) {
245. system("cls");
246. cout << "The size of the field is wrong" << endl;
247. cin.ignore(numeric\_limits<streamsize>::max(), '\n');
248. \_getch();
249. }
250. } while (cols\*lines <= 4);
251. auto tSer = 0;
252. auto tParNew = 0;
253. vector<State\*> ans;
254. for (int i = 0; i < TESTS; i++) {
255. srand(i);
256. map = generateMap(lines, cols);
257. cout << "\n" << "-----------------------------------------------------";
258. cout << "\n" << "Case #" << i + 1 << ": ";
259. printMap(map);
260. //Решение последовательным
261. // clock\_t time = clock();
262. std::chrono::high\_resolution\_clock::time\_point t1 = std::chrono::high\_resolution\_clock::now();
263. ans = a(map);
264. std::chrono::high\_resolution\_clock::time\_point t2 = std::chrono::high\_resolution\_clock::now();
265. // time = clock() - time;
266. // cout << "\n" << "Solution:";
267. // for (int k = ans.size() - 1; k > 0; k--) {
268. // printMap(ans[k]->getMap());
269. // }
270. auto duration = std::chrono::duration\_cast<std::chrono::milliseconds>( t2 - t1 ).count();
271. cout << "\n" << "Time of SERIAL = " << duration;
272. // printMap(ans[0]->getMap());
273. tSer += duration;
274. //Решение новым параллельным
275. // time = clock();
276. std::chrono::high\_resolution\_clock::time\_point t11 = std::chrono::high\_resolution\_clock::now();
277. ans = aPar2(map);
278. std::chrono::high\_resolution\_clock::time\_point t22 = std::chrono::high\_resolution\_clock::now();
279. // time = clock() - time;
280. cout << "\n" << "Solution:";
281. // for (int k = ans.size() - 1; k > 0; k--) {
282. // printMap(ans[k]->getMap());
283. // }
284. auto duration2 = std::chrono::duration\_cast<std::chrono::milliseconds>( t22 - t11 ).count();
285. cout << "\n" << "Time of NEW PARALLEL = " << duration2;
286. // printMap(ans[0]->getMap());
287. tParNew += duration2;
288. }
289. //Начиная с 4 на 4 можно старый параллельный не считать, 1000 выборок
290. cout << "\n" << "--------------" << endl;
291. cout << "Average time SERIAL = " << tSer / TESTS << endl;
292. cout << "Average time NEW PARALLEL = " << tParNew / TESTS << endl;
293. cout << "SERIAL / NEW PARALLEL = " << (double)(tSer / tParNew) << endl;
294. cout << "--------------" << endl;
295. system("pause");
296. return 0;

}Результаты тестирования

Тестирование проводилось при следующей конфигурации ЭВМ:

* AMD A8-7410 2.2 GHz (4 ядра)
* 8 Gb ОЗУ (DDR3L)
* Количество перемешиваний собранного игрового поля для получения входного игрового поля равнялось: кол-во элементов поля \* 20.

Таблица 1 – Сравнение алгоритмов

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность головоломки | Среднее время решения выборки из 5 головоломок | | |
| Последовательный | Параллельный 1 | Параллельный 2 |
| 2х3 | 0,0002 | 0,0642 | 0,0028 |
| 3х3 | 0,0032 | 2,1782 | 0,0142 |
| 3х4 | 0,3384 | 7,6808 | 0,0222 |
| 4х4 | 1,1784 | 16,4664 | 0,7264 |
| 4х5 | 3,3778 | 34,1576 | 0,9358 |
| 5х5 | 24,2352 | 104,3425 | 1,9843 |

Затем был использован компилятор GCC с этой же конфигурацией ЭВМ:

Таблица 2 – Сравнение алгоритмов

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность головоломки | Последовательный | Параллельный | Последовательный/Параллельный |
| 2x3 | 0,062540 | 0,619025 | 0,1 |
| 3x3 | 1,457634 | 0.651726 | 2,24 |
| 3x4 | 3,155734 | 0,812236 | 3,88 |
| 4x4 | 1,637676 | 0,749886 | 2,18 |
| 4x5 | 2,005796 | 0,820880 | 2,44 |
| 5x5 | 16,969815 | 1,983010 | 8,4 |

Вывод

В ходе выполнения лабораторной работы были разработаны 2 варианта параллельного алгоритма.

Первый параллельный алгоритм показал свою полную несостоятельность, связанную с малым объемом параллельных вычислений. Он оказался даже хуже последовательного из-за того. Второй вариант параллельного алгоритма же показал свою эффективность, так как рассматривается не самое лучшее, собранное состояние, а также и 3 других на первом этапе. В итоге получается так, что одновременно работают 4 последовательных алгоритма, что и даёт лучший результат. При использовании компилятора GCC можно наблюдать, что параллельный алгоритм в среднем в 2,8 раз быстрее параллельного.