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 №3

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

Задание на лабораторную работу

Познакомиться со стандартом OpenMP, получить навыки реализации многопоточных SPMD-приложений с применением OpenMP.

1. Изучить основные принципы создания приложений с использованием библиотеки OpenMP, рассмотреть базовый набор директив компилятора.
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 OpenMP, 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 OpenMP-реализации алгоритма.

Выделение областей для распараллеливания

Для распараллеливания были использованы parallel\_section и section в основном цикле поиска возможного хода. Это позволяет выполнять поиск сразу по 4 сторонам параллельно.

Листинг

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

}

}Результаты тестирования

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Size | Time Serial | Time Parallel | Serial/Parallel | OMP | Serial/OMP |
| 2x3 | 0,062540 | 0,619025 | 0,1 | 0,43 | 0,15 |
| 3x3 | 1,457634 | 0.651726 | 2,24 | 0,44 | 3,31 |
| 3x4 | 3,155734 | 0,812236 | 3,88 | 0,32 | 9,8 |
| 4x4 | 1,637676 | 0,749886 | 2,18 | 0,55 | 2,98 |
| 4x5 | 2,005796 | 0,820880 | 2,44 | 1,35 | 1,49 |
| 5x5 | 16,969815 | 1,983010 | 8,4 | 2,235 | 7,6 |

Вывод

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

В результате было выяснено, что на небольших данных OMP реализация немного быстрее реализации на потоках, на больших же немного медленнее, в среднем реализация на OMP быстрее последовательного в 4,2 раза. Использовался компилятор GCC.