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 №2

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 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.
6. Выделение областей для распараллеливания

Основной цикл проверки состояний распараллелить невозможно, так как есть зависимость по данным между итерациями. Внутри самого цикла распараллелить возможно только проверку наличия состояний в списках. Данные проверки можно распараллелить при помощи директивы pragma omp sections. Максимально количество секций – 4, соответственно максимальное количество потоков – 4, но не все потоки будут задействованы, так как фишку невозможно сдвинуть в некоторые позиции. Для поля 4х4 среднее количество потоков будет равно:

Для поля 5х5:

1. Исходный код параллельного алгоритма

Исходный код параллельного алгоритма представлен на рисунке 1

|  |
| --- |
| //Алгоритм поиска решения головоломки  std::vector<State\*> a(Map\* map) {  BinTree open = BinTree();  BinTree close = BinTree(new State(map, NULL));  State\* min = close.min();  for(;min->getCost() != 0; min = open.min(), close.add(min), open.del(min))  {  //Поиск пустой клетки  int zero = min->getMap()->find(0);  State\* states[4] = { 0, 0, 0, 0 };  #pragma omp parallel sections  {  #pragma omp section  {  //Проверка элемента сверху  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);  states[0] = s;  }  }  }  #pragma omp section  {  //Проверка элемента справа  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);  states[1] = s;  }  }  }  #pragma omp section  {  //Проверка элемента снизу  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);  states[2] = s;  }  }  }  #pragma omp section  {  //Проверка элемента слева  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);  states[3] = s;  }  }  }  }  //Обновление списка вершин для проверки  for (int i = 0; i < 4; i++)  {  if (states[i]) {  open.add(states[i]);  }  }  }  std::vector<State\*> solution;    State\* s = min;  do  {  solution.push\_back(s);  s = s->getParent();  } while (s != NULL);  return solution;  } |

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 параллельной и линейной программы представлены на рисунках 2 и 3.

Таблица 1

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тест | Линейное время (сек) | Параллельное время (сек) |
| 1 | |  |  |  |  | | --- | --- | --- | --- | | 13 | 11 | 10 | 9 | | 8 | 6 | 5 | 12 | | 2 | 4 | 3 | 14 | | 7 | 15 | 1 |  | | 7.001 | 4.707 |
| 2 | |  |  |  |  | | --- | --- | --- | --- | | 14 | 6 | 9 | 12 | | 2 | 5 | 11 | 15 | | 1 | 3 | 8 | 10 | | 13 | 7 | 4 |  | | 3.58 | 2.318 |
| 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 |  | | 475.311 | 304.868 |
| 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 |  | | 32.783 | 20.439 |
| 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 |  | | 29.047 | 18.57 |
| 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 |  | | 17.439 | 11.012 |
| 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 |  | | 1.331 | 0.812 |
| 8 | |  |  |  |  | | --- | --- | --- | --- | | 12 | 6 | 13 | 11 | | 15 | 9 | 8 | 2 | | 3 | 14 | 7 | 5 | | 4 | 10 | 1 |  | | 0.877 | 0.567 |
| 9 | |  |  |  | | --- | --- | --- | | 3 | 10 | 2 | | 7 | 6 | 4 | | 1 | 5 | 8 | | 11 | 9 |  | | 0.424 | 0,287 |
| 10 | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 18 | 15 | 1 | 8 | 9 | | 6 | 19 | 13 | 2 | 11 | | 5 | 16 | 3 | 10 | 12 | | 4 | 17 | 7 | 14 |  | | 2.595 | 1.642 |

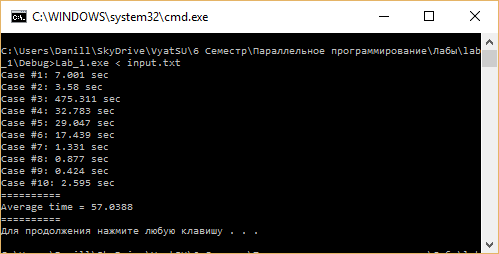


Рисунок 2 – Время работы линейной программы

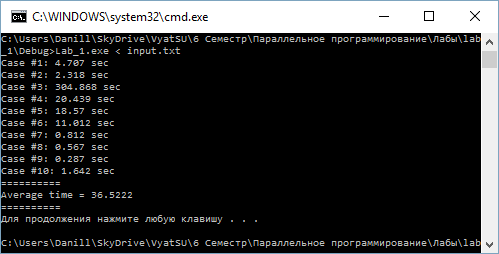


Рисунок 3 – Время работы параллельной программы

1. 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.

Параллельная реализация не принесла существенного прироста в скорости, она ускорила поиск ответа примерно в 1.5 раза, это обуславливается небольшим средним количеством активных потоков и временем задержки, связанным со созданием и управлением потоками.