

НИУ ВШЭ
Факультет Компьютерных Наук
Сделано Коптевым Олегом
Станиславовичем
Из группы БПИ197

Разработка многопоточных приложений с использованием OpenMP

Вариант 10.

Найти все возможные тройки компланарных векторов.

Входные данные: множество не равных между собой векторов
 (x, y, z) , где x, y, z – числа.

Оглавление

1. Алгоритм	3
1.1 Суть алгоритма	3
2. Компиляция и сборка	3
3. Пример работы	3
3.1. 1 Тест	3
3.2. 2 Тест	3
3.3. 3 Тест	4
3.4. 4 Тест	5
3.5. 5 Тест	6
Приложение 1. Исходный код с комментариями	7

1. Алгоритм

Определение принадлежности трех векторов к одной плоскости включает в себя вычисление смешанного произведения этих векторов. Именно эта задача распределяется на потоки, так как все остальные элементы программы не требуют сложных арифметических операций.

Для решения задачи используется итерационный параллелизм.

1.1 Суть алгоритма

- 1) Считывание всех векторов из входного файла;
- 2) Составить все возможные тройки с помощью вложенных циклов и сохранить их в массив;
- 3) Для каждой тройки вычислить значение свойства компланарности (именно здесь и реализуется многопоточная обработка данных);

Для реализации многопоточности используется инструкция `#pragma omp parallel for`, которая является указателем библиотеки OpenMP для особой обработки цикла. Эквивалентно записи `#pragma omp parallel { #pragma omp for for(...) {...} }`. Библиотека OpenMP автоматически распределяет каждую итерацию одному из доступных потоков, и таким образом происходит параллельное вычисление свойств компланарности для каждой тройки.

- 4) Записать компланарные тройки векторов в выходной файл.

2. Компиляция и сборка

Программа была скомпилирована в среде Xcode на платформе macOS. Запуск программы выполняется командой с аргументами `./app <inputFileName> <outputFileName>`.

3. Пример работы

Ниже представлены несколько вариантов входных данных и соответствующие результаты исполнения:

3.1. 1 Тест

Входные данные:	Выходные данные:
0 0 0	

3.2. 2 Тест

Входные данные:	Выходные данные:
-----------------	------------------

0 0 1 0 1 1 0 1 0	((0, 0, 1), (0, 1, 1), (0, 1, 0))
-------------------------	-----------------------------------

3.3. 3 Тест

Входные данные:	Выходные данные:
13 99 69	((38, 2, 24), (72, 63, 21), (40, 10, 22))
38 2 24	((38, 2, 24), (96, 44, 78), (63, 89, 78))
88 69 9	((28, 75, 94), (19, 57, 70), (25, 12, 28))
91 10 32	((27, 81, 40), (19, 57, 70), (29, 87, 60))
62 15 85	((27, 81, 40), (36, 73, 30), (99, 92, 10))
30 51 1	((29, 11, 4), (43, 62, 37), (38, 17, 7))
31 37 37	((93, 29, 93), (95, 43, 95), (73, 36, 73))
92 13 12	((93, 29, 93), (95, 43, 95), (71, 46, 71))
28 75 94	((93, 29, 93), (73, 36, 73), (71, 46, 71))
81 74 80	((62, 52, 72), (42, 76, 8), (30, 1, 59))
27 81 40	((69, 56, 44), (75, 5, 65), (41, 37, 25))
29 11 4	((77, 86, 60), (68, 68, 51), (43, 62, 37))
28 88 65	((77, 86, 60), (68, 68, 51), (95, 22, 53))
93 29 93	((77, 86, 60), (43, 62, 37), (95, 22, 53))
42 85 25	((3, 44, 93), (63, 89, 78), (31, 65, 86))
54 49 90	((95, 43, 95), (73, 36, 73), (71, 46, 71))
85 1 94	((62, 14, 38), (40, 78, 59), (75, 89, 82))
25 79 7	((62, 14, 38), (40, 78, 59), (61, 83, 72))
4 17 30	((62, 14, 38), (75, 89, 82), (61, 83, 72))
52 65 54	((75, 5, 65), (87, 3, 74), (48, 86, 83))
53 72 89	((94, 85, 68), (58, 57, 54), (62, 72, 87))
79 97 64	((40, 78, 59), (75, 89, 82), (61, 83, 72))
72 1 10	((99, 0, 24), (57, 0, 47), (53, 0, 48))
68 81 67	((41, 25, 79), (23, 1, 7), (84, 23, 81))
62 52 72	((52, 71, 17), (59, 57, 24), (40, 10, 22))
54 59 23	((71, 14, 42), (23, 1, 7), (25, 12, 28))
72 63 21	((68, 68, 51), (43, 62, 37), (95, 22, 53))
41 25 94	((23, 1, 7), (48, 80, 91), (17, 23, 27))
69 56 44	((0, 51, 13), (0, 44, 86), (0, 30, 17))
77 86 60	
45 95 29	
...	

3.4. 4 Тест

Входные данные:	Выходные данные:
64 26 43	((10, 54, 1), (28, 12, 4), (24, 60, 3))
59 47 43	((29, 10, 5), (46, 28, 14), (13, 38, 19))
10 54 1	((29, 10, 5), (46, 28, 14), (91, 12, 6))
62 94 46	((29, 10, 5), (13, 38, 19), (91, 12, 6))
82 92 88	((46, 28, 14), (13, 38, 19), (91, 12, 6))
86 23 74	((1, 1, 96), (15, 15, 59), (74, 74, 27))
10 92 23	
15 56 91	
53 25 1	
8 87 25	
41 92 5	
81 29 19	
29 10 5	
48 89 50	
96 77 73	
18 88 80	
59 62 36	
46 45 44	
71 29 53	
33 27 36	
50 26 0	
24 60 59	
35 16 71	
75 1 59	
7 4 10	
9 66 30	
41 99 74	
73 91 65	
69 46 55	
83 31 73	
28 12 4	
46 28 14	
25 14 11	
5 88 8	
82 35 63	
95 6 58	
78 93 62	
31 51 94	
46 19 79	
48 95 98	
1 1 96	
91 27 81	
44 58 27	
73 50 94	
7 53 21	
15 41 6	
46 50 9	
69 20 15	
81 11 16	
84 13 42	
25 99 93	
22 31 77	
65 81 59	
26 19 9	
...	

3.5. 5 Тест

Входные данные:	Выходные данные:
94 64 64	((26, 21, 49), (74, 7, 91), (51, 28, 84))
89 40 33	((59, 68, 54), (7, 49, 45), (97, 24, 6))
80 11 65	((59, 86, 29), (12, 18, 27), (27, 40, 40))
99 32 81	((0, 70, 35), (70, 4, 13), (70, 38, 30))
13 46 29	((65, 49, 21), (3, 49, 21), (98, 35, 15))
67 2 99	((12, 31, 0), (69, 26, 0), (99, 86, 0))
5 4 74	((12, 31, 0), (69, 26, 0), (92, 84, 0))
75 5 68	((12, 31, 0), (69, 26, 0), (99, 62, 0))
48 83 71	((12, 31, 0), (99, 86, 0), (92, 84, 0))
89 62 88	((12, 31, 0), (99, 86, 0), (99, 62, 0))
98 47 50	((12, 31, 0), (92, 84, 0), (99, 62, 0))
43 79 41	((69, 26, 0), (99, 86, 0), (92, 84, 0))
95 77 5	((69, 26, 0), (99, 86, 0), (99, 62, 0))
88 39 5	((69, 26, 0), (92, 84, 0), (99, 62, 0))
77 60 31	((12, 18, 27), (63, 38, 57), (29, 6, 9))
26 21 49	((99, 86, 0), (92, 84, 0), (99, 62, 0))
36 2 39	((13, 66, 77), (67, 94, 11), (40, 80, 44))
4 23 46	((13, 66, 77), (8, 11, 77), (4, 3, 41))
97 83 3	
33 83 22	
49 39 20	
39 30 36	
37 80 45	
8 62 12	
19 68 94	
59 25 72	
84 56 29	
9 15 58	
0 41 9	
11 76 15	
95 33 43	
86 58 88	
44 74 41	
15 0 24	
74 7 91	
59 68 54	
59 86 29	
8 12 73	
0 70 35	
96 39 53	
77 42 78	
62 49 75	
...	

В последних трех тестах используется большое количество случайно сгенерированных векторов.

Приложение 1. Исходный код с комментариями

```
//
// main.cpp
// task4
//
// Created by Oleg Koptev on 29.11.2020.
//

#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <omp.h>

/**
 * Definition of 3 space lengths
 */
class Point3D {
private:
    float x_;
    float y_;
    float z_;
public:
    Point3D() { x_ = 0; y_ = 0; z_ = 0; }
    Point3D(float x, float y, float z) { x_ = x; y_ = y; z_ = z; }
    float x() { return x_; }
    float y() { return y_; }
    float z() { return z_; }
};

/**
 * Structure containing 3 vectors and boolean value if they are coplanar
 */
struct Triplet {
    Point3D triplet[3];
    bool bCoplanar;
};

int main(int argc, const char * argv[]) {

    // 1. Check for correct amount of arguments
    if (argc != 3) {
        std::cout << "Wrong amount of arguments. Please enter \"<input_file>
<output_file>\"\\n";
        return 1;
    }

    // 2. Read all vectors from input file
    std::vector<Point3D> vectors;
    float x, y, z;
    std::ifstream input (argv[1]);
    if (input.is_open()) {
        while (input >> x >> y >> z) {
            vectors.push_back(Point3D(x, y, z));
        }
        input.close();
    } else {
        std::cout << "Couldn't open the input file\\n";
        return 1;
    }

    // 3. Form an array of all possible triplets
```

```

    long amountOfPoints = vectors.size();
    long amountOfTriplets = amountOfPoints * (amountOfPoints - 1) *
(amountOfPoints - 2) / 6;
    std::vector<Triplet> triplets(amountOfTriplets);
    int tripletIndex = 0;
    for (auto i = vectors.begin(); i != vectors.end(); ++i) {
        for (auto j = next(i, 1); j != vectors.end(); ++j) {
            for (auto k = next(j, 1); k != vectors.end(); ++k) {
                struct Triplet triplet;
                triplet.triplet[0] = *i;
                triplet.triplet[1] = *j;
                triplet.triplet[2] = *k;
                triplet.bCoplanar = false;

                triplets[tripletIndex] = triplet;
                tripletIndex++;
            }
        }
    }

    // 4. Calculate coplanar property for all triplets
#pragma omp parallel for
    for (size_t i = 0; i < triplets.size(); i++) {
        Triplet* triplet = &*(triplets.begin()+i);
        float value =
            triplet->triplet[0].x() *
            (triplet->triplet[1].y() * triplet->triplet[2].z() -
            triplet->triplet[1].z() * triplet->triplet[2].y()) +
            triplet->triplet[0].y() *
            (triplet->triplet[1].z() * triplet->triplet[2].x() -
            triplet->triplet[1].x() * triplet->triplet[2].z()) +
            triplet->triplet[0].z() *
            (triplet->triplet[1].x() * triplet->triplet[2].y() -
            triplet->triplet[1].y() * triplet->triplet[2].x());
        if (value == 0) {
            triplet->bCoplanar = true;
        }
    }

    // 5. Write coplanar triplets
    std::stringstream outputString;
    for (auto i = triplets.begin(); i != triplets.end(); ++i) {
        Triplet triplet = *i;
        if (!triplet.bCoplanar) { continue; }
        outputString << "("
            << triplet.triplet[0].x() << ", " << triplet.triplet[0].y() << ",
" << triplet.triplet[0].z() << "), ("
            << triplet.triplet[1].x() << ", " << triplet.triplet[1].y() << ",
" << triplet.triplet[1].z() << "), ("
            << triplet.triplet[2].x() << ", " << triplet.triplet[2].y() << ",
" << triplet.triplet[2].z() << ")\n";
    }
    std::ofstream output (argv[2]);
    if (output.is_open()) {
        output << outputString.str();
    } else {
        std::cout << "Error when writing to output file\n";
    }

    return 0;
}

```