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

Практические приемы построения многопоточных приложений

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

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 многопотчная обработка данных);

В пункте выше мы рассматриваем две ситуации: количество троек меньше количества потоков и количество троек больше или равное количеству потоков. В первой ситуации мы проводим расчеты только на одном потоке, в то время как в другой мы используем все N потоков.

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

3.4. 4 Тест

Входные данные:	Выходные данные:
64 26 43 59 47 43 10 54 1 62 94 46 82 92 88 86 23 74 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 …	((10, 54, 1), (28, 12, 4), (24, 60, 3)) ((29, 10, 5), (46, 28, 14), (13, 38, 19)) ((29, 10, 5), (46, 28, 14), (91, 12, 6)) ((29, 10, 5), (13, 38, 19), (91, 12, 6)) ((46, 28, 14), (13, 38, 19), (91, 12, 6)) ((1, 1, 96), (15, 15, 59), (74, 74, 27))

3.5. 5 Тест

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

В последних трех тестах используется большое количество случайно сгенерированных векторов.

Приложение 1. Исходный код с комментариями

```
//
//  main.cpp
//  task3
//
//  Created by Oleg Koptev on 08.11.2020.
//

#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
using namespace std;

#define NUM_THREADS 4

/**
 * 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 it they are coplanar
 */
struct Triplet {
    Point3D triplet[3];
    bool bCoplanar;
};

/**
 * Information for a thread
 */
typedef struct {
    long start_index;
    long end_index;
    vector<Triplet>* triplets;
} pthrData;

/**
 * Function for a thread, that counts coplanar property for needed amount of triplets
 */
void* threadFunc(void* thread_data) {
    pthrData *data = (pthrData*) thread_data;
    auto start = next(data->triplets->begin(), data->start_index);
    auto end = next(data->triplets->begin(), data->end_index);

    for (auto i = start; i != end; ++i) {
        Triplet* triplet = &(*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;
        }
    }

    return NULL;
};

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

//  1. Check for correct amount of arguments
    if (argc != 3) {
        cout << "Wrong amount of arguments\n";
        return 1;
    }

//  2. Read all vectors from input file
    vector<Point3D> vectors;
    float x, y, z;
    ifstream input (argv[1]);
    if (input.is_open()) {
        while (input >> x >> y >> z) {
            vectors.push_back(Point3D(x, y, z));
        }
        input.close();
    } else {
        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;
    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
    if (amountOfTriplets < NUM_THREADS) {
        pthread_t thread;
        pthrData threadData;
        threadData.triplets = &triplets;
        threadData.start_index = 0;
```

```
        threadData.end_index = amountOfTriplets;
        pthread_create(&thread, NULL, threadFunc, &threadData);
        pthread_join(thread, NULL);
    } else {
        pthread_t* threads = (pthread_t*) malloc(NUM_THREADS *
sizeof(pthread_t));
        pthrData* threadData = (pthrData*) malloc(NUM_THREADS *
sizeof(pthrData));
        long shift = ceil((float)amountOfTriplets / (float)NUM_THREADS);
        for (int i = 0; i < NUM_THREADS; i++) {
            threadData[i].triplets = &triplets;
            threadData[i].start_index = i * shift;
            if (i < NUM_THREADS - 1) {
                threadData[i].end_index = (i + 1) * shift;
            } else {
                threadData[i].end_index = amountOfTriplets;
            }

            pthread_create(&(threads[i]), NULL, threadFunc, &threadData[i]);
        }
        for(int i = 0; i < NUM_THREADS; i++)
            pthread_join(threads[i], NULL);
    }

//  5. Write coplanar triplets
    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";
    }
    ofstream output (argv[2]);
    if (output.is_open()) {
        output << outputString.str();
    } else {
        cout << "Error when writing to output file\n";
    }


    return 0;
}
```