Threshold-Free Cluster Enhancement

Luigi Giugliano[1], Marco Mecchia[1]

[1]Università degli studi di Salerno

5 maggio 2016

Overview

Codice

Overview

Codice

Codice

Andremo ora a spiegare il codice prodotto per il plugin TFCE.

I file principali che compongono il plugin sono:

- Tfce.cpp
- Utilities.cpp

Tfce è il core del plugin, dove avviene il calcolo degli score.

Utilities invece contiene tutti le funzioni di supporto per l'esecuzione del plugin stesso.

L' unica funzione che viene esposta dal file **Tfce.h** è:

```
1 #ifndef TFCE_H
2     #define TFCE_H
3     #include <float.h>
4     float * tfce_score(float * map, int dim_x, int
          dim_y, int dim_z, float E, float H, float dh);
5 #endif //TFCE_H
```

Le funzioni che espone **Utilities.h** sono:

```
1  #ifndef UTILITIES_H
2      #define UTILITIES_H
3
4      #include <float.h>
5      #include <stdio.h>
6
7      void findMinMax(float *map, int n, float *min,
           float *max, float * range);
8
9      int confront(float a, float b, char operation);
10
11     int * getBinaryVector(float * map, int n, int
           (*confront)(float, float), float value, int *
               numOfElementsMatching);
```

```
1     float * fromBinaryToRealVector(float * map, int n,
          int * binaryVector);
2
3     float * fill0(int n);
4
5     void apply_function(float * vector, int n, float (*
          operation) (float a, float b), float argument);
6
7     int linearIndexFromCoordinate(int x, int y, int z,
          int max_x, int max_y);
8
9     void coordinatesFromLinearIndex(int index, int
          max_x, int max_y, int * x, int * y, int * z);
10
11    float * copyAndConvertIntVector(int * vector, int n);
12
13 #endif //UTILITIES_H
```

Andremo ora a vedere l'implementazione della funzione **tfice_score**:

```
 1 findMinMax(map, n, &minData, &maxData, &rangeData);
 2 precision = rangeData/dh;
 3 if (precision > 200) {
 4     increment = rangeData/200;
 5 } else{
 6     increment = rangeData/precision;
 7 }
 8 steps = ceil((maxData - minData) / (increment));
 9 #pragma omp parallel for
10 for (i = 0; i < steps; i++) {
11     computeTfceIteration(minData + i*increment, map,
           n, dim_x, dim_y, dim_z, E, H, dh, toReturn);
12 }
13 return toReturn;
```

Funzione **computeTfceIteration**:

```
1  int * indexMatchingData = getBinaryVector(map, n,
       moreThan, h, &numOfElementsMatching);
2  clustered_map = find_clusters_3D(indexMatchingData,
       dim_x, dim_y, dim_z, n, &num_clusters);
3  extent_map = new int[n];
4  for (j = 0; j < n; ++j){
5       extent_map[j] = 0;
6  }
7  delete [] indexMatchingData;
8  for (i = 1; i <= num_clusters; ++i) {
9       numOfElementsMatching = 0;
10      for (j = 0; j < n; ++j){
11          if(clustered_map[j] == i){
12              numOfElementsMatching++; }}
13      for (j = 0; j < n; ++j) {
14          if(clustered_map[j] == i)
15              extent_map[j] = numOfElementsMatching; }}
```

```
1  clustered_map_float =
       copyAndConvertIntVector(extent_map, n);
2  apply_function(clustered_map_float, n, elevate, E);
3  apply_function(clustered_map_float, n, multiply, pow(h,
       H));
4  apply_function(clustered_map_float, n, multiply, dh);
5  for (i = 0; i < n; ++i) {
6  #pragma omp atomic
7      toReturn[i] += (clustered_map_float[i]);
8  }
9  delete[] clustered_map_float;
10 delete[] clustered_map;
11 delete[] extent_map;
```

Funzione **getBinaryVector**:

```
1  int * getBinaryVector(float * map, int n, int
       (*confront)(float, float), float value, int *
       numOfElementsMatching){
2      int * binaryVector = new int [n];
3      (*numOfElementsMatching) = 0;
4      int i;
5      for (i = 0; i < n; ++i) {
6          if(confront(map[i],value)){
7              binaryVector[i] = 1;
8              (*numOfElementsMatching)++;
9          }
10         else
11             binaryVector[i] = 0;
12         }
13     return binaryVector;
14 }
```

Funzione **find_cluster_3D**:

```
1 int * find_clusters_3D(int * binaryVector, int dim_x,
      int dim_y, int dim_z, int n, int * num_clusters)
```

questo metodo preso in input una **mappa binaria in 3D ma linearizzata**, le sue tre dimensioni, il numero totale voxel della mappa e il puntatore ad un intero che indica il numero attuale di cluster trovati.

Cercando i cluster all'interno della mappa 3D utilizzando la **26-connectivity**

Restituisce un'altra mappa in cui al posto degli uno viene sostituito l'identificativo del cluster.

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 3 & 3 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 2 & 2 & 2 \\ 1 & 0 & 0 & 2 & 2 \end{bmatrix}$$

In questo piccolo esempio in 2D viene mostrato il funzionamento della nostra funzione.

E' stato utilizzata la specifica OpenMP per rendere il calcolo degli score più velocemente.

OpenMP (Open Multiprocessing) è un API multipiattaforma per la creazione di applicazioni parallele su sistemi a memoria condivisa.

Il comando:
#pragma omp parallel for
viene utilizzato per rendere un for parallelo.

Il comando:
#pragma omp atomic
invece viene utilizzato per rendere un istruzione atomica.

Abbiamo deciso di utilizzare, OMP perché l'effort per utilizzarlo è praticamente nullo, e le prestazioni sono ottime.

Inoltre essendo che l'implementazione dei *Thread* in *C* cambia tra Windows e Linux, si sarebbe reso necessario modificare il codice per renderlo funzionante su entrambe le piattaforme.