UNIVERSITÀ
DEGLI STUDI
FIRENZE

Pattern Recognition

Gabriele Boddi

UNIVERSITÀ DEGLI STUDI FIRENZE

Indice

UNIVERSITÀ DEGLI STUDI FIRENZE

Introduzione

- ***Pattern Recognition***: metodologia che permette di studiare la similarità di un insieme di dati rispetto a determinati *parametri di ricerca*.
- viene utilizzato in numerosi ambiti dove è necessaria l'analisi dei dati.
- Per i suoi calcoli viene utilizzata la misura matematica ***SAD*** (*Sum of Absolute Differences*).

UNIVERSITÀ DEGLI STUDI FIRENZE

Componenti Informatiche utilizzate

- *Array* di **k** elementi per il calcolo delle misure SAD.
- 2 *Matrici*, di dimensioni variabili, rappresentanti rispettivamente l'insieme di dati che si vuole studiare e la porzione di valori che fungono da parametro di ricerca (detto anche *query*).

Descrizione della misura SAD

- Siano dati in input le codifiche matematiche, sotto forma di matrici, di un *template* ed un'*immagine* che vogliamo confrontare:

Template	Search image
2 5 5	2 7 5 8 6
4 0 7	1 7 4 2 7
7 5 9	8 4 6 8 5

UNIVERSITÀ DEGLI STUDI FIRENZE

Descrizione della misura SAD

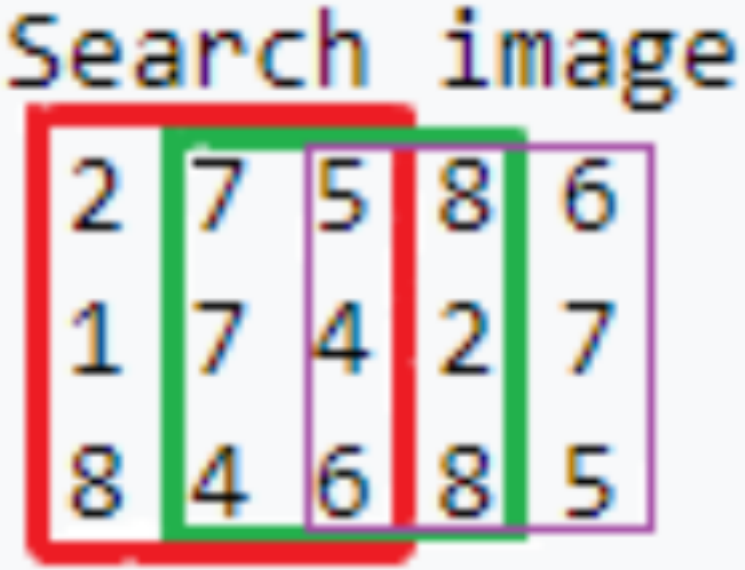
- Ciò che vogliamo fare è suddividere l'immagine in *Regioni* per effettuare uno studio suddiviso in più parti.
- La loro cardinalità viene data dalla seguente formula:

$$regions = cols - querycols + 1 \tag{1}$$

dove *cols* sono le colonne della matrice rappresentante l'immagine e *querycols* le colonne che rappresentano il template.

Descrizione della misura SAD

- L'immagine quindi sarà divisa in 3 Regioni: il lato *sinistro*, il lato *centrale* ed il lato *destro*.



Descrizione della misura SAD

- Si definiscono 3 nuove matrici ottenute dalle differenze tra i valori di ogni regione con quelli del template (in valore assoluto).

Left	Center	Right
0 2 0	5 0 3	3 3 1
3 7 3	3 4 5	0 2 0
1 1 3	3 1 1	1 3 4

- Si procede infine a ricavare le somme degli elementi al loro interno (rispettivamente 20, 25 e 17) che costituiscono le **similarità**: da questi valori si ricava che la parte destra dell'immagine è la più simile al template, poiché contiene il valore più basso.

UNIVERSITÀ DEGLI STUDI FIRENZE

Descrizione dell'Algoritmo

- Il seguente programma è stato implementato in **C++**, utilizzando le direttive **OpenMP** per realizzare la sua versione parallela.
- Il codice si divide in due classi: una che serve per estrapolare dati da *file* (con la possibilità di generarli anche in maniera casuale nel caso dei parametri di ricerca), e quella principale utilizzata per il funzionamento del programma.
- Di seguito verrà mostrata la struttura di quest'ultima (per vedere il funzionamento della prima classe si può consultare il file *Report.pdf* presente insieme a questa presentazione).

UNIVERSITÀ DEGLI STUDI FIRENZE

Descrizione dell'Algoritmo

Costruttore della classe principale (di entrambe le versioni):

```
GetElements g;
cout << "Loading Dataset..." << endl;
elements = g.get_elements_from_file(rows,cols,"blogData_train.csv");
cout << "Dataset Loaded." << endl << endl;

cout << "Loading Query..." << endl;
query = g.get_random_query(queryrows,querycols,rows,cols,regions);
cout << "Query Loaded." << endl << endl;

if(queryrows>=cols || queryrows!=rows){
throw std::runtime_error("Error: query parameters are incorrect!");
}

if(regions == -1) regions = cols-querycols+1;

if(regions<2 || regions>cols){
std::runtime_error("Error: the value of the Regions is incorrect!");
}
```

UNIVERSITÀ DEGLI STUDI FIRENZE

Descrizione dell'Algoritmo

Funzione di calcolo SAD (versione ***Sequenziale***):

```
void Pr::calculate_sad()
{
    initialize_sad();

    for(int k=0; k<regions; k++)
    {
        sad[k] = sum(k);
    }
}
```

UNIVERSITÀ DEGLI STUDI FIRENZE

Descrizione dell'Algoritmo

Funzione di calcolo SAD (versione ***Parallela***):

```
void Pr_parallel::calculate_sad()
{
    initialize_sad();

    #pragma omp parallel for num_threads(omp_get_max_threads())
    for(int k=0; k<regions; k++)
    {
        sad[k] = sum(k);
    }
}
```

UNIVERSITÀ DEGLI STUDI FIRENZE

Descrizione dell'Algoritmo

Funzione di inizializzazione SAD (versione ***Sequenziale***):

```
void Pr::initialize_sad()
{
    sad = new double[regions];

    for(int i=0; i<regions; i++)
    {
        sad[i] = 0;
    }
}
```

Descrizione dell'Algoritmo

Funzione di inizializzazione SAD (versione ***Parallela***):

```
void Pr_parallel::initialize_sad()
{
    sad = new double[regions];

    #pragma omp parallel for num_threads(omp_get_max_threads())
    for(int i=0; i<regions; i++)
    {
        sad[i] = 0;
    }
}
```

UNIVERSITÀ DEGLI STUDI FIRENZE

Descrizione dell'Algoritmo

Funzione di somma (versione ***Sequenziale***):

```
double Pr::sum(int k)
{
    double sum = 0;

    for(int i=0; i<rows; i++)
    {
        for(int j=0; j<querycols; j++)
        {
            sum += abs(elements[i][j+k]-query[i][j]);
        }
    }

    return sum;
}
```

Descrizione dell'Algoritmo

Funzione di somma (versione ***Parallela***):

```
double Pr_parallel::sum(int k)
{
    double sum = 0;

    for(int i=0; i<rows; i++)
    {
        #pragma omp simd reduction(+:sum)
        for(int j=0; j<querycols; j++)
        {
            sum += abs(elements[i][j+k]-query[i][j]);
        }
    }

    return sum;
}
```

Descrizione dell'Algoritmo

Funzione di restituzione della regione più *simile* (per entrambe le versioni):

```
int Pr::find_min_sad()
{
    double min_value = sad[0];
    int index = 0;

    for(int i=1; i<regions; i++)
    {
        if(sad[i]<min_value)
        {
            min_value = sad[i];
            index = i;
        }
    }

    return index;
}
```

Risultati di Esecuzione

- Nella seguente tabella è possibile vedere i tempi di esecuzione sequenziali e paralleli, su varie suddivisioni in regioni, di un *Dataset* da 281 righe e 60021 colonne.
- E' stata utilizzata una *query* definita con valori casuali per semplicità di esecuzione dei vari test.
- Come sarà possibile vedere dai risultati, mantenendo una suddivisione in Regioni non superiore alle 10 unità si raggiunge uno *Speedup Superlineare*.
- Architettura usata: *CPU Intel(R) Core(TM) i3-3217U, RAM 4 GB ddr3, Processori Logici: 4, Processori Fisici: 2*.

Risultati di Esecuzione

Regioni	Tempo Seq.	Tempo Par.	Speedup
2	0.588862	0.298821	1.971
3	0.977397	0.335793	2.911
4	1.19542	0.391792	3.051
5	1.4571	0.653596	2.23
6	1.92481	0.654602	2.94
7	1.91282	0.71156	2.688
8	2.18664	1.01016	2.164
9	2.62541	1.14934	2.284
10	2.83424	1.59339	1.778

UNIVERSITÀ DEGLI STUDI FIRENZE

Conclusioni

Con la versione implementata in OpenMP, questo programma raggiunge una grande ottimizzazione in termini di Speedup, specialmente con una suddivisione in poche regioni.
Ad ogni modo, le prestazioni restano comunque ottime sfruttando la parallelizzazione.