



Equalizzazione Istogramma

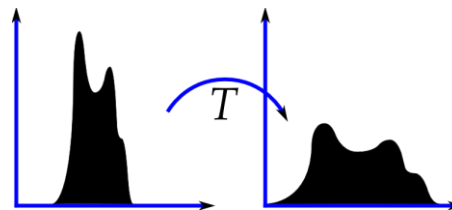
Implementazione e risultati in modalità
parallela con CUDA e OpenMP

Alessandro Lemmo

Introduzione

Cos'è l'equalizzazione

- L'equalizzazione dell'istogramma è un metodo per migliorare la qualità di un immagine con un'esposizione non ottimale ovvero dove sfondo e primo piano sono entrambi troppo chiari o troppo scuri.
- Un istogramma rappresenta in modo grafico la distribuzione tonale di un'immagine. In pratica, traccia il numero di pixel presenti per ogni singolo valore tonale.
- L'equalizzazione consiste nel ridistribuire tale grafico su un range più ampio



- In base alla tipologia di immagine si usano tecniche di equalizzazione leggermente diverse.

Introduzione

Aspetti teorici

- Per un immagine in scala di grigi la formula da applicare è la seguente:

$$h(v) = \text{round} \left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L - 1) \right)$$

- Per un immagine a colori RGB si hanno due possibilità
 - Applicare la formula sopra a tutti e tre i canali separatamente
 - Convertire l'immagine in YCbCr e applicare la formula solo alla luminanza Y

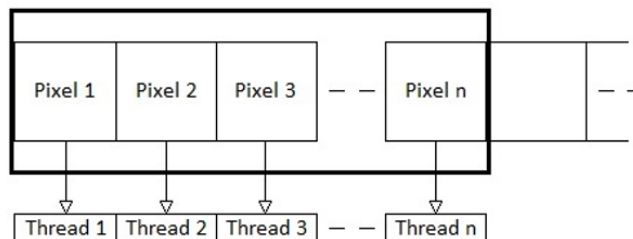
$$\begin{cases} Y = R * 0.299000 + G * 0.587000 + B * 0.114000 \\ Cb = R * -0.168736 + G * -0.331264 + B * 0.5 + 128 \\ Cr = R * 0.5 + G * -0.418688 + B * -0.081312 + 128 \end{cases}$$

$$\begin{cases} R = Y + 1.4075 * (V - 128) \\ G = Y - 0.3455 * (U - 128) - (0.7169 * (V - 128)) \\ B = Y + 1.7790 * (U - 128) \end{cases}$$

Implementazione CUDA

Primo kernel

- Il lavoro è stato suddiviso in tre fasi:
 - Conversione RGB \rightarrow YCbCr
 - Equalizzazione canale Y
 - Riconversione YCbCr \rightarrow RGB
- Si implementano tre kernel, uno per fase, in modo da parallelizzare le tre operazioni
- Nella prima fase ogni thread lavora sul singolo pixel in accordo con il seguente schema



```

01. int idx = blockIdx.x * blockDim.x + threadIdx.x
02.
03. for(int i = idx; i < width * height; i += blockDim.x * gridDim.x)
04.
05.     index = i * 3;
06.     int r = ptr_image[index + 0];
07.     int g = ptr_image[index + 1];
08.     int b = ptr_image[index + 2];
09.
10.     //conversion to YCbCr
11.     int Y = ...
12.     int Cb = ...
13.     int Cr = ...
14.
15.     ptr_image[index + 0] = Y
16.     ptr_image[index + 1] = Cb
17.     ptr_image[index + 2] = Cr
18.
19.     //update the histogram of Y channel
20.     histogram[Y] += 1;
    
```

Implementazione CUDA

Secondo e terzo kernel

- Il secondo kernel si occupa dell'equalizzazione del canale Y prendendo in input la funzione di distribuzione cumulativa

```
01. int idx = blockIdx.x * blockDim.x + threadIdx.x;  
02.  
03. for(int i = idx; i < 256; i += blockDim.x * gridDim.x)  
04.  
05.     eq_histogram[i] = (cdf[i] - cdf[0] / (width * height - 1)) * 255;
```

- La cdf (Cumulative Distribution Function) è calcolata a partire dall'istogramma sommando in modo cumulativo il numero di pixel presenti per ogni valore tonale.
- Il terzo kernel effettua la riconversione in modo analogo a quanto fatto nel primo kernel



Implementazione OpenMP

Conversione canali

- Logica di funzionamento analoga quella dell'implementazione CUDA
 - Tre fasi parallelizzate tramite funzionalità OpenMP
- La clausola default con la condizione shared specifica che tutte le variabili nel blocco seguente saranno condivise
- La clausola schedule specifica come le iterazioni del ciclo sono suddivise tra i thread
 - Specificando la condizione static le iterazioni vengono suddivise in blocchi di dimensioni uguali, con un blocco per ogni thread

```
01. #pragma omp parallel num_threads(32) default(shared)
02.     #pragma omp for schedule(static)
03.
04.     for each rows i in the image
05.         for each cols j in the image
06.
07.             int R = image(i,j).R;
08.             int G = image(i,j).G;
09.             int B = image(i,j).B;
10.
11.             //conversion to YCbCr
12.             ...
13.
14.             histogram[Y]++;
15.
16.             image(i,j).R = Y;
17.             image(i,j).G = Cb;
18.             image(i,j).B = Cr;
```

Implementazione OpenMP

Equalizzazione istogramma e riconversione

- Il secondo blocco parallelizzato è quello relativo all'equalizzazione del canale Y
- Equalizzazione effettuata tramite la funzione di distribuzione cumulativa

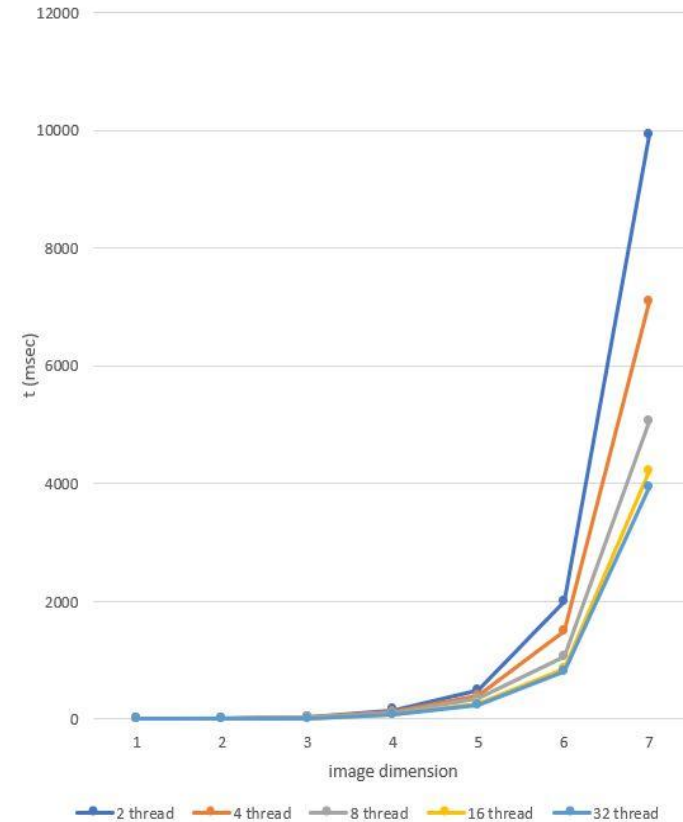
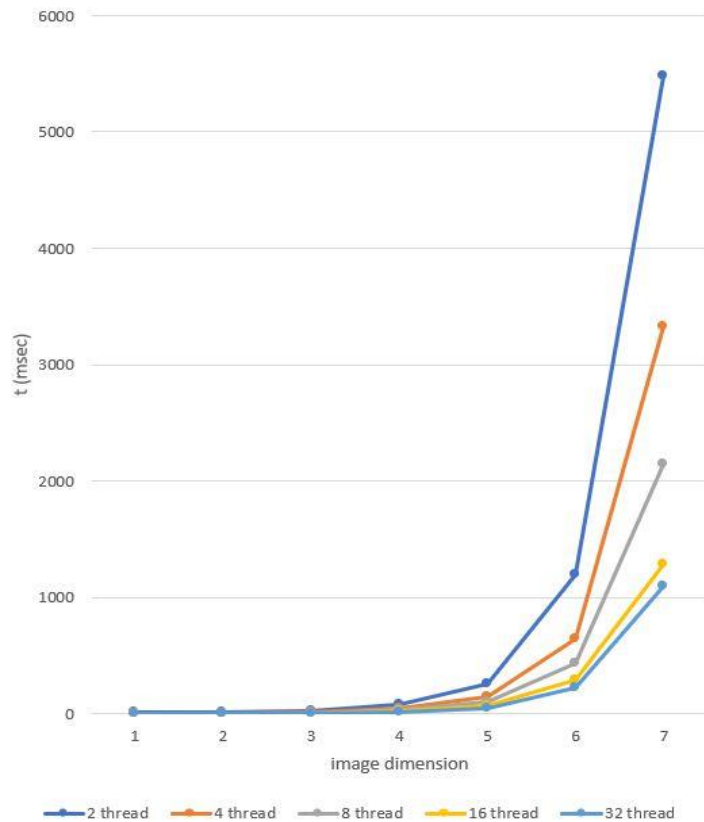
```
01. #pragma omp parallel for
02.
03.     for( int i = 1; i < 256; i ++ )
04.
05.         eq_histogram[i] = ((cdf[i] - cdf[0]) / (cols * rows - 1) * 255)
```

- Il terzo blocco di riconversione in RGB è analogo al primo

Risultati

Variazione numero di thread

- Confronto tra i tempi delle due versioni al variare del numero di thread (OpenMP dx, CUDA sx)

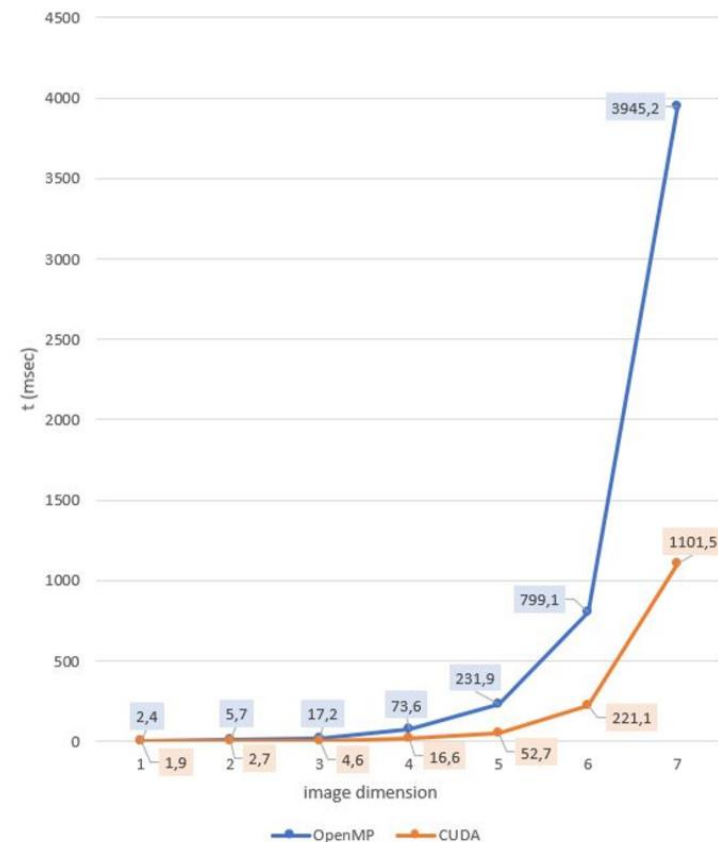


Risultati

Variazione dimensione immagine

- Confronto tra i tempi delle due versioni al variare delle dimensioni dell'immagine equalizzata

Image Dimension	OpenMP (msec)	CUDA (msec)
309 x 201	2.4	1.9
497 x 302	5.7	2.7
900 x 599	17.2	4.6
2000 x 1499	73.6	16.6
4000 x 2670	231.9	52.7
8000 x 6000	799.1	221.1
25816 x 8935	3945.2	1101.5



Risultati SpeedUp

- Realizzata una versione sequenziale per confrontare lo SpeedUp delle due versioni calcolato con la seguente formula

$$S_p = \frac{T_s}{T_p}$$

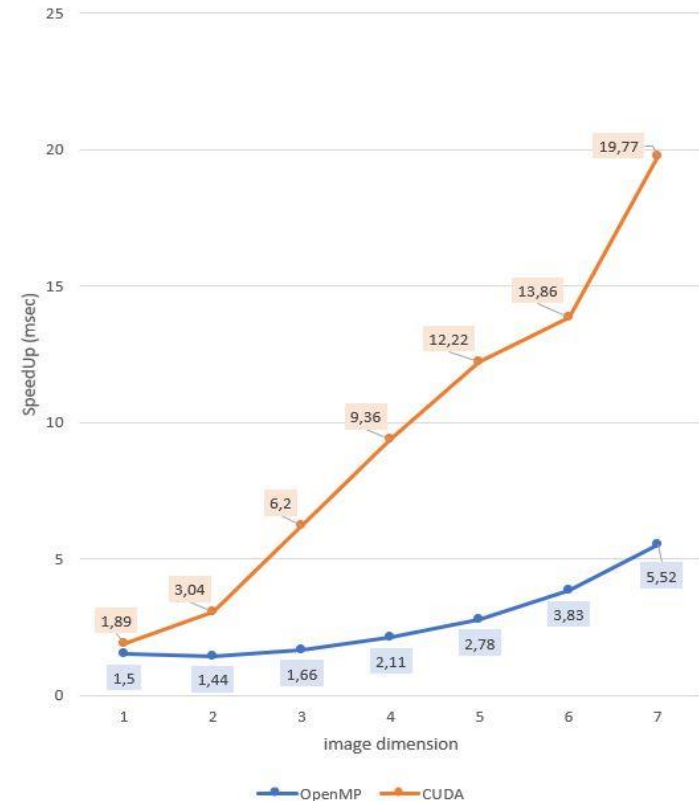
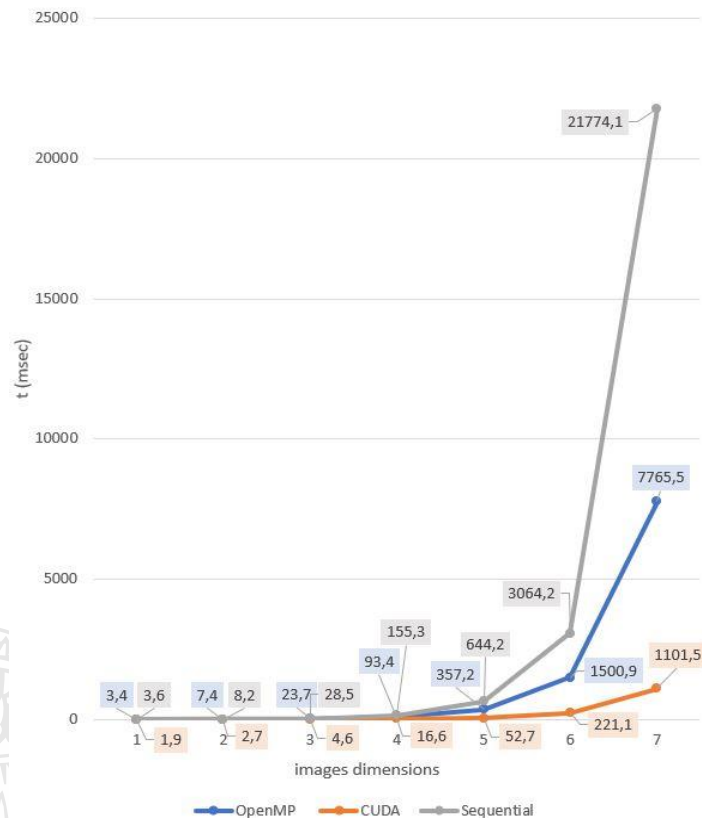


Image Dimension	SpeedUp OpenMP (msec)	SpeedUp CUDA (msec)
309 x 201	1.06	1.89
497 x 302	1.11	3.04
900 x 599	1.2	6.2
2000 x 1499	1.66	9.36
4000 x 2670	1.8	12.22
8000 x 6000	2.04	13.86
25816 x 8935	2.8	19.77

Esempi

- Risultati ottenuti con un immagine sovraesposta e con una sottoesposta

