

Parallel histogram equalization

Macaluso Alberto

Matricola: 7115684

alberto.macaluso@edu.unifi.it

<https://github.com/albe9/parallel-histogram-equalization>

Abstract

Nell'elaborato è stato trattato un algoritmo in grado di effettuare l'histogram equalization di immagini in scala di grigi. Inizialmente verrà descritto l'algoritmo nella sua forma base (sequenziale) e in seguito saranno illustrate due diverse implementazioni parallele basate sull'architettura CUDA. Infine verranno mostrate un'analisi e un confronto delle varie versioni andando ad evidenziarne le caratteristiche principali.

1. Introduzione

L'histogram equalization è una tecnica di elaborazione delle immagini utilizzata per migliorare il contrasto in un'immagine. Consiste nel riassegnare la distribuzione dei livelli di grigio in modo uniforme sull'intero intervallo disponibile, producendo un'immagine con un contrasto migliore e una maggiore vivacità dei dettagli. Questa tecnica è particolarmente utile quando l'istogramma dell'immagine originale è concentrato su una piccola gamma di valori di grigio.

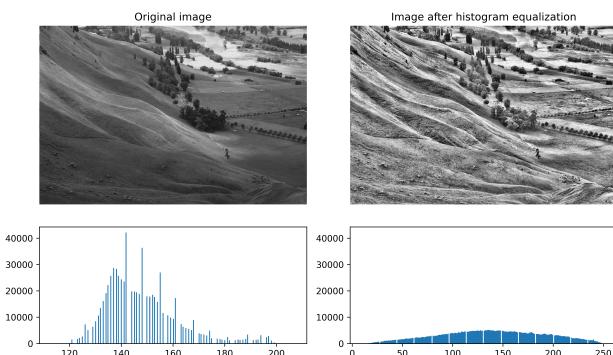


Figure 1: Esempio di histogram equalization

Ad ogni pixel dell'immagine originale viene applicata una trasformazione calcolata in modo tale da generare una nuova immagine con un'istogramma "piatto".

1.1. Implementazione

Inizialmente ogni immagine viene caricata in memoria tramite l'utilizzo della libreria STB Image[1], in seguito sono estratti i valori dei pixel su cui verranno eseguite delle operazioni preliminari (1.3). A questo punto avviene l'histogram equalization vera e propria.

L'implementazione dell'algoritmo[2] per effettuare l'equalizzazione può essere riassunta nei seguenti passaggi:

1. Calcolo dell'istogramma dell'immagine in ingresso, $H(i)$ dove $i = 0, 1, 2, \dots, L - 1$ e L è il numero di livelli di intensità (tipicamente 256 per immagini a 8 bit).
2. Normalizzazione dell'istogramma per ottenere la funzione di densità di probabilità (PDF), $P(i) = \frac{H(i)}{MN}$ dove M è il numero di righe e N è il numero di colonne dell'immagine.
3. Calcolo della funzione di distribuzione cumulativa (CDF), $C(i) = \sum_{j=0}^i P(j)$.
4. Mappatura di ogni intensità del pixel r in una nuova intensità s usando la formula:

$$s = (L - 1) \cdot C(r)$$

5. Infine, sostituzione di ogni intensità del pixel r nell'immagine originale con la sua corrispondente nuova intensità s .

Questi step assicurano che l'istogramma dell'immagine in uscita sia approssimativamente uniforme, portando ad un contrasto migliorato. Infine, utilizzando la libreria STB Image writer[1], le nuove immagini vengono salvate in memoria.

1.2. CLAHE

Nell'elaborato è stata utilizzata una versione più avanzata dell'histogram equalization base appena descritta, ossia Contrast Limited Adaptive Histogram Equalization[3].

CLAHE è una variante dell'histogram equalization che limita l'amplificazione del contrasto in aree dell'immagine con grande differenza di intensità. A differenza dell'histogram equalization standard, CLAHE divide l'immagine in regioni più piccole e applica l'equalizzazione dell'istogramma a ciascuna di queste regioni in modo adattivo.

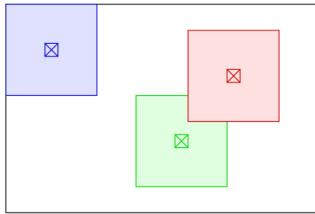


Figure 2: Applicazione dell'equalizzazione nelle regioni (Fonte: Wikipedia[3])

Inoltre, per evitare l'aumento del rumore nelle aree uniformi, CLAHE utilizza un meccanismo di limitazione del contrasto. Questo consiste nel ridistribuire uniformemente i valori dell'istogramma che superano un certo limite (clip). Ciò impedisce che il contrasto venga amplificato eccessivamente, mantenendo una migliore qualità dell'immagine finale.

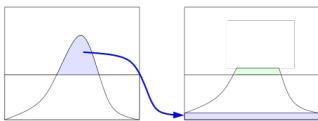


Figure 3: Redistribuzione dei valori oltre il limite (Fonte: Wikipedia[3])

1.3. Gestione dei bordi

Per poter applicare l'histogram equalization in modo adattivo è stato necessario gestire i pixel presenti nei bordi delle immagini. Per questo motivo è stata utilizzata una tecnica di border mirroring che consiste nello specchiare i valori dei pixel lungo i bordi delle immagini. L'offset per effettuare il mirroring è dato dal raggio delle regioni su cui verrà in seguito applicata l'equalizzazione dell'istogramma. In questo modo, per costruzione, ogni pixel avrà sempre un intorno sufficiente a contenere tale finestra.

2. Implementazione in CUDA

Oltre alle operazioni preliminari in comune con la versione sequenziale, l'implementazione in CUDA ha richiesto l'allocazione di due regioni di memoria relative all'immagine in ingresso e a quella in uscita. Su queste è avvenuta la copia dei valori dei pixel rispettivamente da CPU a GPU prima di eseguire il kernel e da GPU a CPU

una volta terminato. In seguito sono state definite le dimensioni dei blocchi (16x16) e in base a queste, la dimensione della griglia, scelta in modo tale da poter coprire tutta l'immagine.

Il kernel è stato progettato con l'idea di associare ad ogni pixel dell'immagine un thread che andrà ad effettuare l'equalizzazione considerando una finestra centrata su di esso (con un raggio predefinito).

Una volta calcolato l'istogramma e ridistribuiti i valori oltre il limite impostato (come precedentemente spiegato in 1.2) se ne ricava la CDF. Infine mappando tramite di essa l'intensità iniziale del pixel si ottiene la sua nuova intensità, che verrà quindi salvata nell'immagine in uscita.

L'equalizzazione di una data regione è completamente svincolata da tutte le altre, di conseguenza ogni thread può operare senza il bisogno di alcun tipo di meccanismo di sincronizzazione.

2.1. Utilizzo della shared memory

Per il calcolo degli istogrammi è necessario che ogni thread legga i valori di tutti i pixel relativi alla propria finestra e questo comporta numerosi accessi alla memoria globale. Poiché le regioni dei pixel vicini sono in parte sovrapposte, molte letture dalla memoria globale potrebbero essere evitate, andando così a migliorare le performance. Per questo motivo è stata implementata una seconda versione che sfrutta la shared memory.

Il kernel è stato diviso in due fasi:

- Prima Fase: Caricamento dei dati nella shared memory.

In questa fase i threads si occupano di caricare i valori dei pixels dalla memoria globale a quella condivisa.

- Image
- Border mirroring
- Thread Block
- Thread Block index

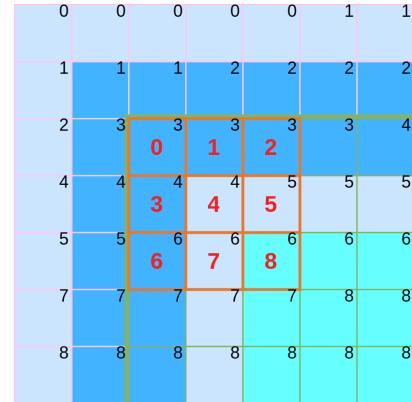


Figure 4: Assegnazione dei pixel da caricare per ogni thread

Per costruzione il numero di pixels necessari è sempre superiore al numero di threads presenti in un blocco, di conseguenza ogni thread dovrà caricare più di un dato come è possibile osservare nello schema 4. Nell'esempio i blocchi e le regioni su cui equalizzare hanno dimensioni rispettivamente 3x3 e 5x5 e nell'angolo di ogni pixel è indicato l'indice del thread che andrà a caricarlo in memoria.

Conclusa questa fase, avviene una sincronizzazione dei threads relativi allo stesso blocco così da essere sicuri che prima di proseguire, tutti i dati siano stati caricati nella memoria condivisa.

- Seconda Fase: Histogram equalization.

In questa fase i threads effettuano le stesse operazioni della versione descritta nella sezione 2, con la differenza che tutti i threads appartenenti allo stesso blocco andranno ad accedere alla stessa memoria condivisa invece di quella globale.

2.2. Profiling

Per verificare il flusso di esecuzione ed il corretto utilizzo dell'hardware è stata condotta una profilazione tramite il tool Nvidia Nsight[4].

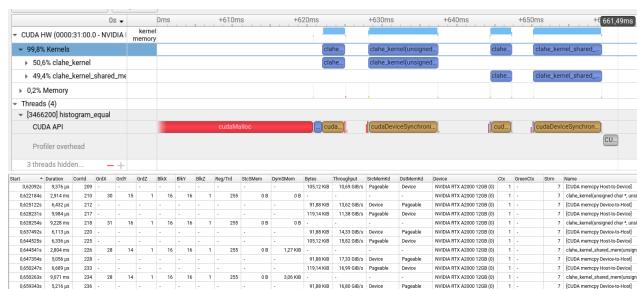


Figure 5

L'esempio (5) è relativo all'esecuzione di due kernel prima per la versione base e in seguito per quella che sfrutta la memoria condivisa. Tra la prima e la seconda esecuzione di ogni versione è stato aumentato il raggio delle regioni su cui effettuare l'equalizzazione.

Il tool ha reso possibile osservare anche l'occupancy teorica della GPU che è risultata essere del 16-17% circa. Questo è dovuto all'elevato utilizzo dei registri (255 per thread) causato dal calcolo dell'istogramma delle regioni. Sono stati provati diversi approcci quali la memorizzazione dell'istogramma in shared memory o memoria globale o anche l'utilizzo di una flag in compilazione per limitare l'utilizzo dei registri. Tutti questi tentativi hanno effettivamente portato alla riduzione del numero di registri e ad un'occupancy del 100%, ma le performance complessive sono risultate essere nettamente inferiori. Probabilmente, a causa della natura del problema, non è possibile aumentare

ulteriormente l'occupancy e la velocità di esecuzione senza andare a riformulare completamente l'algoritmo.

3. Risultati

Tutti i benchmarks sono stati condotti sotto gli stessi parametri di valutazione:

- Hardware: GPU Nvidia A2000
- Raggio delle regioni di equalizzazione: 10
- Clip limit: 4
- Ripetizioni per affidabilità: 3

Il dataset[5] è stato convertito in scala di grigi prima di poter essere utilizzato ed è stato disabilitato il salvataggio delle immagini in output in seguito all'histogram equalization.

3.1. CPU vs GPU

Il primo test è stato condotto per confrontare le performance tra la versione sequenziale (sulla CPU) e quella parallela di base (sulla GPU senza shared memory).

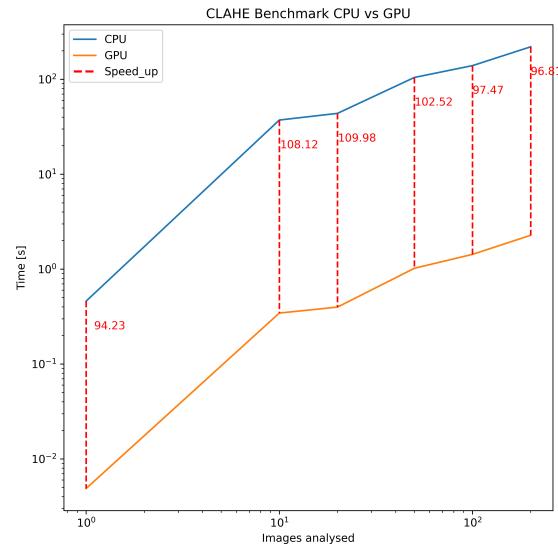


Figure 6

È possibile osservare in figura (6) che, come ci si aspettava, lo speed up è notevole, inoltre esso resta abbastanza costante con l'aumentare del numero delle immagini.

3.2. GPU vs GPU con shared memory

Il secondo test è stato condotto per confrontare le performance tra le due versioni parallele, rispettivamente con e senza utilizzo di memoria condivisa.

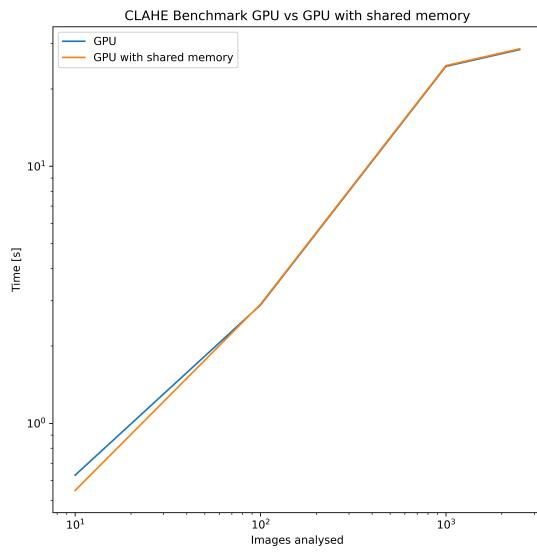


Figure 7

Dai risultati ottenuti (7) le differenze osservate sono pressoché insignificanti. Il motivo principale probabilmente risiede nell'utilizzo già ottimo delle cache L1 e L2 da parte dell'architettura che va a nullificare quindi l'utilizzo della shared memory.

References

- [1] S. Barrett, “STB Image and STB Image Writer Libraries.” GitHub, ongoing. <https://github.com/nothings/stb>.
- [2] W. contributors, “histogram equalization formulation,” 2024. https://en.wikipedia.org/wiki/Histogram_equalization#Implementation.
- [3] W. contributors, “Adaptive histogram equalization,” 2022. https://en.wikipedia.org/w/index.php?title=Adaptive_histogram_equalization&oldid=1115555840.
- [4] Nvidia Corporation, *Nvidia Nsight*, 2024. <https://developer.nvidia.com/nsight-systems>.
- [5] H. Xiao, “Weather phenomenon database (WEAPD),” 2021. <https://www.kaggle.com/datasets/jehanbhathena/weather-dataset>.