Calcolo Distribuito e sistemi ad alte prestazioni

Relazione GPU Computing: Image Convolution

De Angelis Fabio

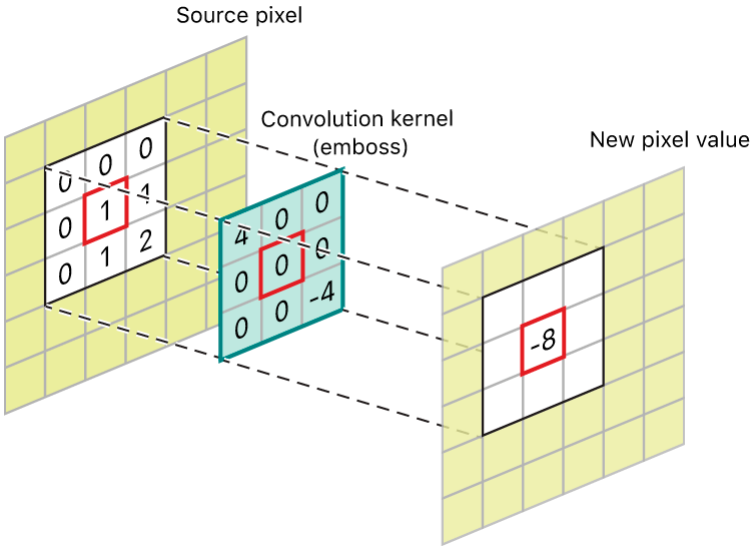
matricola: 318037

Introduzione

L'elaborazione dei dati e delle istruzioni dei programmi è compito specifico del processore o CPU (Central Processing Unit). I vari core che compongono il processore sono ottimizzati per l'esecuzione seriale sequenziale delle istruzioni: i pacchetti dati in arrivo dalla memoria di lavoro e le istruzioni dei programmi sono eseguite una alla volta, in successione, in maniera seriale. Tutti i core, dunque, si occupano allo stesso tempo della medesima istruzione sino a che questa non è completata, generando dunque un flusso di lavoro sequenziale.
Le GPU, invece anziché eseguire una sola operazione alla volta (serialmente), ne portano avanti diverse migliaia contemporaneamente (in parallelo). Ciò ha avuto un innegabile influsso sulla struttura delle componenti interne: anziché avere pochi core ottimizzati per un lavoro seriale, la scheda grafica ha diverse migliaia di unità di elaborazione, di grandezza e potenza minore rispetto a quella di una CPU, ottimizzate però per lavorare parallelamente.
Le varie unità di calcolo possono essere programmate per eseguire porzioni di codice o programma anche di natura differente grazie all'utilizzo degli shader.

Convoluzione

Una tecnica di elaborazione delle immagini di frequente utilizzo è la tecnica della convoluzione, ossia la generazione di un'immagine di output ottenuta modificando tramite un apposito filtro di convoluzione l'immagine di partenza.
Un filtro di convoluzione è una matrice quadrata che deve avere un numero dispari di righe e colonne.
In base al filtro che si vuole applicare all'immagine, quando questa viene elaborata, viene calcolato un pixel di output per ogni pixel di input mescolando l'intorno di quest'ultimo secondo il tipo di filtro utilizzato.



$$V = \left| \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} f_{ij}}{F} \right|$$

f_{ij} = the coefficient of a convolution kernel at the position i,j in the kernel
d_{ij} = the data value of the pixel which corresponds to f_{ij}
n = the dimension of the kernel, assuming a square kernel
F = la somma dei coefficienti del kernel oppure 1, se la somma è 0
V = il valore finale del pixel

Dato che la convoluzione richiede al suo livello più basso un numero elevato di operazioni, nello specifico moltiplicazioni e sommatorie in virgola mobile, una tecnica di ottimizzazione ragionevole sta nell'utilizzo di della GPU per parallelizzare questa mole di calcolo, poiché le moderne schede grafiche sono progettate per essere in grado di eseguire in parallelo un numero enorme di operazioni in virgola mobile.
Questa relazione mostra come implementare questa tecnica utilizzando la libreria OpenCL per la parallelizzazione su GPU andando poi a confrontare i risultati ottenuti tramite questa tecnica con quelli ottenuti utilizzando un algoritmo sequenziale "classico".

Parallelizzazione

Il codice per l'esecuzione in parallelo deve essere strutturato in due sottoprogrammi, ossia l'host program e il kernel program.

Il kernel program è la procedura/funzione da eseguire tramite OpenCL, ovvero sulla GPU , mentre l'host program contiene tutte le istruzioni per l'inizializzazione ed il controllo dell'esecuzione del kernel program.

Host program:

L'host program è caratterizzato dai seguenti passaggi:
• Richiesta in input del filtro che si vuole applicare per la convoluzione dell'immagine
• Inizializzazione delle platform
• Inizializzazione dei device
• Creazione del context
• Creazione della command queue
• Creazione dei buffer
• Creazione del programma associato al codice del kernel program
• Creazione e compilazione del programma
• Creazione del kernel
• Impostazione dei parametri del kernel
• Esecuzione del kernel
• Lettura del risultato della convoluzione
• Rilascio delle risorse utilizzate

In questo caso nell'host program è stata utilizzata anche la libreria libpng per la creazione, gestione e modifica di immagini in formato png, fondamentale per poter visualizzare l'output della convoluzione sotto forma di immagine.

Kernel Program:

Il kernel program, come già detto in precedenza è il codice OpenCl da eseguire sulla GPU
Viene definito all'interno di un file con estensione cl che viene passato all'host program sotto forma di stringa dalla quale viene generato il kernel program eseguibile tramite l'apposita funzione OpenCl.

Risultati e tempi di esecuzione

Di seguito vengono mostrati i risultati ottenuti utilizzando i vari filtri di convoluzione forniti dall'algoritmo, insieme ai tempi di esecuzione delle due versioni dell'algoritmo (sequenziale (CPU) e parallelo (GPU)).
Le immagini di output dei due algoritmi vengono salvate all'interno della directory del progetto, rispettivamente come cpu.png e gpu.png.

Immagine Originale

File: image.png

Risoluzione: 512x512

Filtri Applicati:

Gaussian Blur (Kernel 5x5, sigma: 0.8)

CPU: 0.827 s

GPU: 0.217 s

Gaussian Blur (kernel 5x5, sigma: 3.0)

CPU: 0.847 s

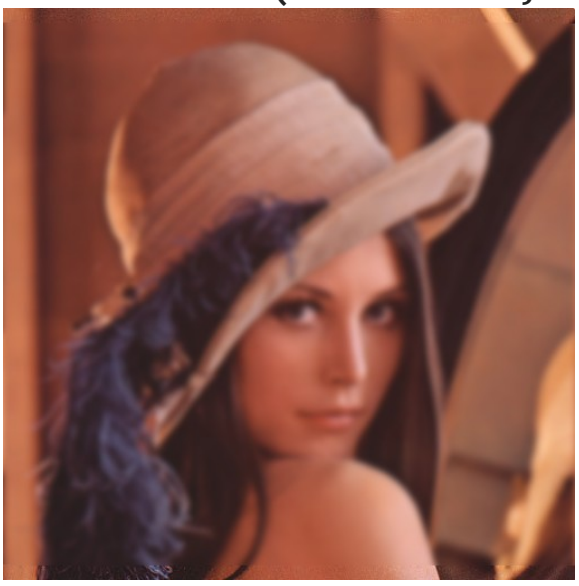
GPU: 0.250 s

Gaussian Blur (kernel 15x15, sigma 3)

CPU: 1,268 S

GPU: 0,238 S

Gaussian Blur (kernel 25x25, sigma 3)



CPU: 4,454 s

GPU: 0,249 s

Gaussian Blur (kernel 35x35, sigma: 0.8)

CPU: 5.743 s

GPU: 2.620 s

Gaussian Blur (kernel 35x35, sigma : 3.0)

CPU: 3.633 s

GPU: 1.215 s

Blur

CPU: 0.792 s

GPU: 0.209 s

Sharpen

CPU: 0.767 s

GPU: 0.244 s

Emboss

CPU: 0.758 s

GPU: 0.248 s

Edge Detection

CPU: 0.753 s

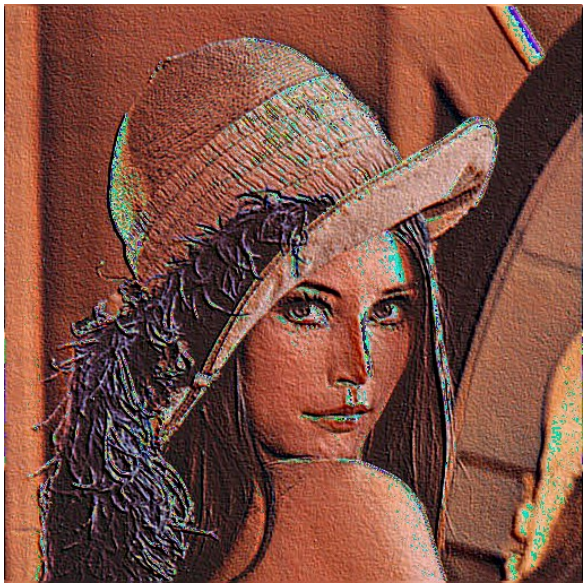
GPU: 0.299 s

Sharpen 5x5

CPU: 0.937 s

GPU: 0.289 s

Relief



CPU: 0.918 s

GPU: 0.497 s

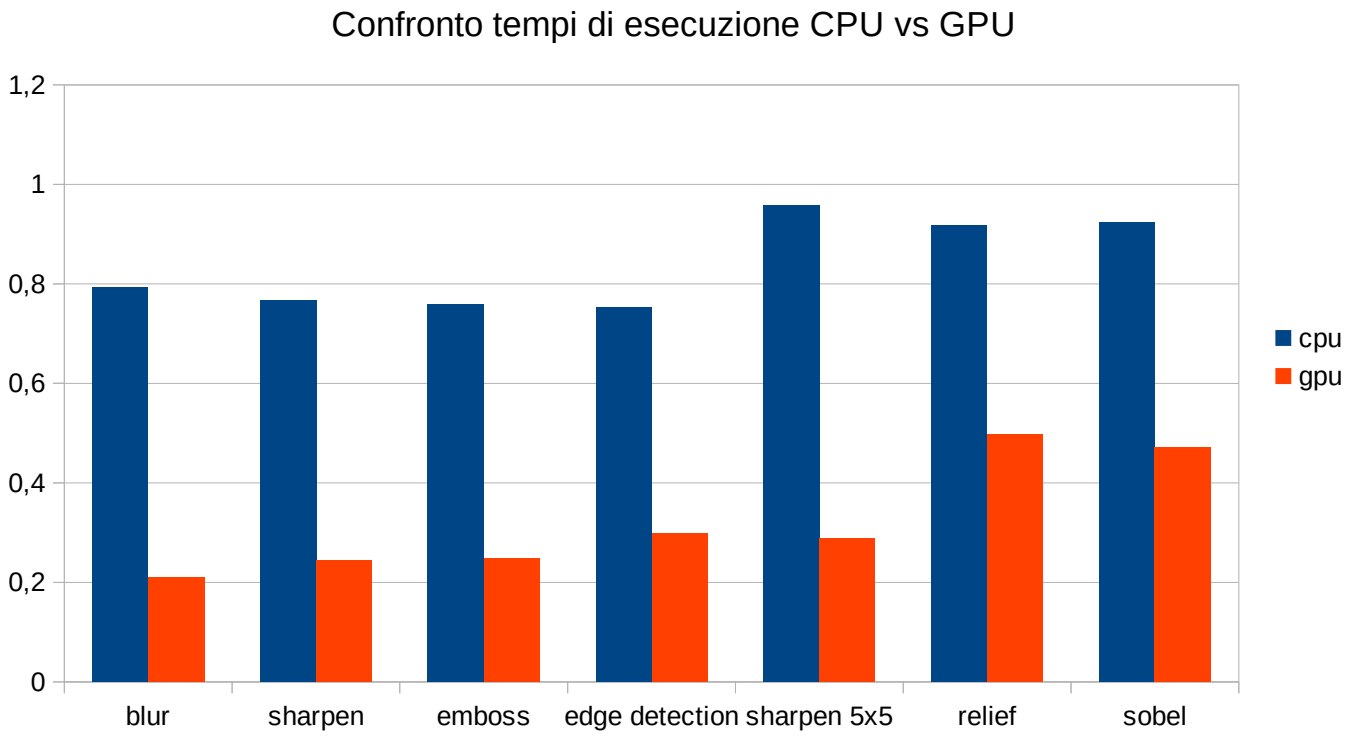
Sobel

CPU: 0.924 s

GPU: 0.471 s

Conclusioni

Dai grafici seguenti, ottenuti confrontando i tempi di esecuzione tra algoritmo seriale e tempo di esecuzione che si ottiene parallelizzando la computazione sulla GPU, si evince come sia assolutamente più conveniente in termini di tempo di esecuzione, l'utilizzo della parallelizzazione su GPU.
È emblematico il secondo grafico, in cui si confrontano i due algoritmi al crescere della grandezza del kernel utilizzato nel Gaussian Blur, nel quale, appunto, si nota come all'aumentare della grandezza della kernel matrix i tempi di esecuzione dell'algoritmo sequenziale crescono notevolmente mentre quelli dell'algoritmo parallelo rimangono sostanzialmente stabili facendo registrare giusto qualche impercettibile variazione.



Gaussian Blur: