# Ingegneria degli Algoritmi (A.A. 2012-2013)

Corsi di Laurea in Ingegneria Informatica e Automatica, Ingegneria dei Sistemi Informatici, e Laurea Magistrale in Ingegneria Informatica

*Sapienza Università di Roma*

**Esercitazione di laboratorio (28/05/2013):**
**fotoritocco mediante filtri di convoluzione su GPU – Durata 2h 30'**

---

**Preliminari (aula 17)**

1. Fare login in Linux (Scientific Linux 5.1) con le credenziali: `studente` (login) e `*****` (password)
2. Creare sulla scrivania (`/local/studente/Desktop`) una directory chiamata: `ida2013`
3. Entrare nella directory `/local/studente/Desktop/ida2013`
4. Copiare il testo e il codice fornito per l'esercitazione nella directory di lavoro: `cp -rf /home/distrib/demetres-sw/eserc130528 .`
5. La directory `/local/studente/Desktop/ida2013/eserc130528` contiene:
    a. `convolution`: directory di lavoro per l'esercitazione
    b. `esempi`: esempi di programmi OpenCL da cui trarre ispirazione
    c. `docs`: documentazione utile su OpenCL

---

**Esercizio 1**

La *convoluzione* è una tecnica usata in programmi di fotoritocco come Adobe Photoshop o Gimp per ottenere filtri di immagine, ad esempio per aumentare la nitidezza, sfocare, estrarre i contorni, ecc. In questo esercizio si chiede di scrivere codice OpenCL per applicare a immagini a toni di grigio dei filtri grafici basati su convoluzione, effettuando il calcolo su GPU.

Un'immagine di altezza `h` pixel e larghezza `w` pixel è rappresentata mediante una matrice `I` con `h` righe e `w` colonne di `unsigned char`, dove il valore 0 denota il nero e 255 il bianco. Gli altri valori rappresentano toni di grigio intermedi. La cella I[y][x] contiene il tono di grigio del pixel di coordinate (x,y).

Un filtro di convoluzione {F, *factor*, *bias*} è costituito da:

- una matrice F di `float` di dimensioni $s \times s$ con $s$ dispari (es. $3\times3$, $5\times5$, ecc.);
- due parametri `float` *factor* e *bias*;

L'immagine di output `O` ottenuta applicando il filtro all'immagine di input `I` è definita per ogni pixel $x \in [0, w-1]$ e $y \in [0, h-1]$ come segue:

$$O[y][x] = \max\{\min\{\lfloor val_{y,x} \rfloor, 255\}, 0\}$$

dove:

$$val_{y,x} = bias + factor \cdot \sum_{u,v \in [0,s-1]} I\left[y - \left\lfloor \frac{s}{2} \right\rfloor + u\right]\left[x - \left\lfloor \frac{s}{2} \right\rfloor + v\right] \cdot F[u][v]$$

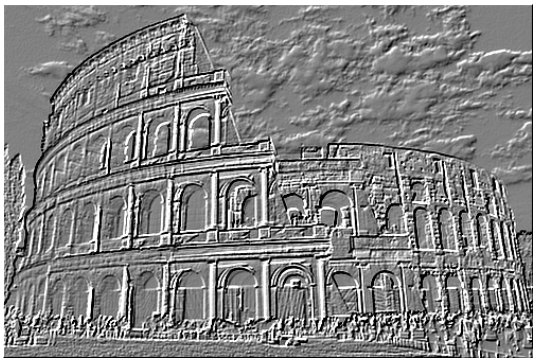
in cui la sommatoria è limitata ai soli addendi per cui gli indici di `I` sono validi, cioè:

$$0 \leq y - \left\lfloor \frac{s}{2} \right\rfloor + u < h \quad \text{e} \quad 0 \leq x - \left\lfloor \frac{s}{2} \right\rfloor + v < w$$

Ad esempio, l'immagine (b) è ottenuta dall'immagine (a) applicando il filtro *emboss*:

$$F = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \qquad bias = 128.0 \qquad factor = 1.0$$

(a) Immagine originale



(b) Immagine con filtro *emboss*

**Obiettivo.**

Si vada nella directory di lavoro `convolution` e si definisca nel file `convolution.cl` un kernel OpenCL C con il seguente prototipo:

```
__kernel void convolution(__global unsigned char* I,
                          __global unsigned char* O,
                          int h, int w,
                          __global float* F,
                          float factor, float bias, int s)
```

dove:

- `I`: puntatore a buffer di dimensione `h*w*sizeof(unsigned char)` byte nella *memoria globale* del device che contiene l'immagine di input in formato row-major;
- `O`: puntatore a buffer di dimensione `h*w*sizeof(unsigned char)` byte nella *memoria globale* del device in cui scrivere l'immagine di output in formato row-major;
- `h`: altezza di `I` e `O` in pixel (numero di righe della matrice);
- `w`: larghezza di `I` e `O` in pixel (numero di colonne della matrice);
- `F`: puntatore a buffer di dimensione `s*s*sizeof(float)` byte nella *memoria globale* del device contente la matrice di convoluzione del filtro in formato row-major;
- `factor`, `bias`: parametri del filtro;

- `s`: altezza e larghezza della matrice `F` in pixel (numero di righe e di colonne).

Il kernel deve definire i valori delle celle della matrice `O` applicando il filtro {`F`, factor, bias} alla matrice I.

**Nota bene**: il kernel viene eseguito su un NDRange a due dimensioni dove la dimensione 0 corrisponde all'asse orizzontale (x) e la dimensione 1 corrisponde all'asse verticale (y).

**Compilazione e test.**

Directory di lavoro: `~/Desktop/ida2013/eserc130528/convolution/`

1. Compilazione programma di test (*una tantum*): dare il comando `make convolution1`, che genera il file eseguibile `convolution1`
2. Compilazione ed esecuzione kernel:
   a. `./convolution1` (o `./convolution1 Colosseo`): test su immagine 500×334 a 256 toni di grigio del Colosseo;
   b. `./convolution1 Lena-512x512`: test su una versione 512×512 a 256 toni di grigio della classica immagine di test "Lena" del 1973;
   c. `./convolution1 Lena-300x300`: test su una versione 300×300 di "Lena".

Nella directory `results` verranno generate varie immagini ottenute applicando diversi filtri di convoluzione alle immagini date in input. Le immagini, salvate in formato PGM (Portable Graymap Format), possono essere visualizzate con il programma Gimp.

---

**Esercizio 2**

L'Esercizio 1 si concentra sulla scrittura del kernel OpenCL C per l'applicazione di un filtro di convoluzione, mentre il resto del programma è fornito in forma mista sorgente C / codice oggetto. Questo esercizio richiede invece di scrivere anche la parte di codice host che esegue il kernel sul device.

**Obiettivo.**

Si apra il file `convolution.c` e si scriva il codice host della funzione:

```
void convolution(unsigned char* I, unsigned char* O, int h, int w,
                 float* F, float factor, float bias, int s,
                 clut_device* dev, double* td)
```

dove:

- `I`: puntatore a buffer di dimensione `h*w*sizeof(unsigned char)` byte che contiene l'immagine di input in formato row-major;
- `O`: puntatore a buffer di dimensione `h*w*sizeof(unsigned char)` byte in cui scrivere l'immagine di output in formato row-major;

- `h`: altezza di `I` e `O` in pixel (numero di righe della matrice);
- `w`: larghezza di `I` e `O` in pixel (numero di colonne della matrice);
- `F`: puntatore a buffer di dimensione s*s*sizeof(float) byte contente la matrice di convoluzione del filtro;
- `factor`, `bias`: parametri del filtro;
- `s`: altezza e larghezza della matrice F in pixel (numero di righe e di colonne);
- `dev`: struttura che raggruppa il contesto di esecuzione, la coda dei comandi e il programma da eseguire sul device;
- `td`: puntatore a buffer in cui passare il tempo richiesto dall'esecuzione del kernel sul device.

La funzione deve lanciare sul device `dev` il kernel sviluppato nell'Esercizio 1, applicando il filtro {F, factor, bias} alla matrice I e scrivendo il risultato nella matrice O. La funzione deve restituire in `*td` il tempo in secondi impiegato dall'esecuzione del kernel.

La definizione del tipo `clut_device` è nel file header `clut.h`. Per le operazioni di:

1. creazione kernel
2. creazione buffer
3. passaggio dei parametri al kernel
4. esecuzione data-parallel del kernel
5. copia di dati da host a device e viceversa
6. rilascio delle risorse

si veda la documentazione OpenCL nella directory `docs` e gli esempi contenuti nella directory `esempi`.

**Compilazione e test.**

Si compili il programma con `make convolution2` e lo si esegua con `./convolution2`, con le stesse modalità dell'Esercizio 1.