UNIVERSITÀ DEGLI STUDI DI PERUGIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

Quarta esercitazione:

CALCOLO DISTRIBUITO E PARALLELO

APPLICAZIONE DI FILTRI AD IMMAGINI SIA IN PARALLELO CHE SEQUENZIALE GRAZIE ALLA MATRIX CONVOLUTION

Studente: Manuel Severi

INDICE

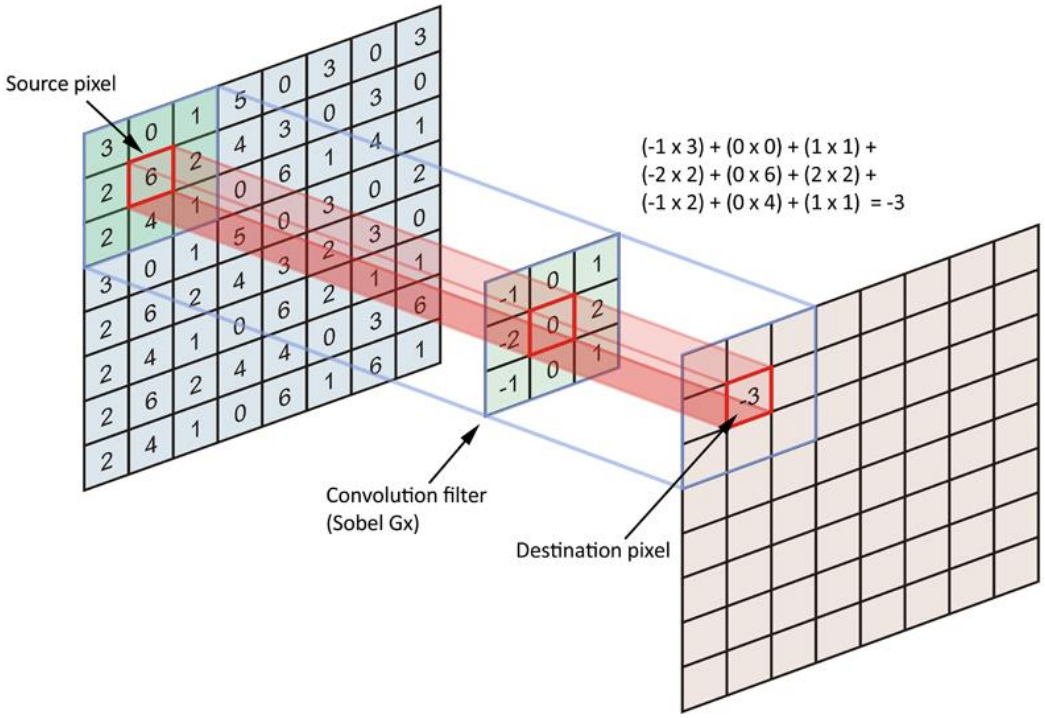
1 INTRODUZIONE

All’interno di questa relazione, eseguirò la modifica di un’immagine. Nello specifico tramite il programma si andrà ad aprire una determinata immagine, verrà applicato un filtro per ottenere il negativo dell’immagine e successivamente viene salvato il tutto. Per eseguire questa modifica ho creato un file scritto in C++ utilizzando Visual Studio come piattaforma d’appoggio.
L’intento di questo studio `e dimostrare che applicare il filtro negativo ad un’immagine in modo sequenziale e/o parallelo porta allo stesso risultato (ovvero il negativo dell’immagine) ma impiegando tempi diversi a favore di quella parallela.

1.1 MATRICE DI CONVOLUZIONE

Nell'elaborazione digitale delle immagini, una matrice di convoluzione, è una piccola matrice usata per applicare filtri ad altre immagini. Risulta dunque utile per la sfocatura, affilatura, goffratura, riconoscimento dei contorni e altro ancora. Attraverso l'applicazione della convoluzione di due matrici bidimensionali di cui la prima rappresenta l'immagine originale e la seconda, detta kernel, rappresenta il filtro da applicare.
Consideriamo la matrice A che rappresenta la matrice contenente i valori di grigio di tutti i pixel dell'immagine originale e la matrice B che rappresenta la matrice kernel. Sovrapponiamo la matrice B alla matrice A in modo che il centro della matrice B sia in corrispondenza del pixel della matrice A da elaborare. Il valore di ciascun pixel della matrice A oggetto di elaborazione viene ricalcolato come la somma pesata dei prodotti di ciascun

elemento della matrice kernel con il corrispondente pixel della matrice A sottostante.



Con la precedente immagine si va a comprendere, tramite un esempio, come avviene l'applicazione del filtro tramite la matrice di convoluzione.

2 | STRUMENTI UTILIZZATI

Il seguente progetto è stato realizzato sfruttando un a scheda grafica AMD e un processore intel core i7 di 8th generazione.

2.1 MICROSOFT VISUAL STUDIO

Si è scelto di utilizzare Visual Studio 2019 per la realizzazione del progetto perché questo IDE sviluppato da Microsoft è gratuito nella sua versione comunity. Visual Studio integra la tecnologia IntelliSense che permette di correggere eventuali errori sintattici, e anche alcuni logici, senza compilare l'applicazione, possiede un debugger interno per il rilevamento e la correzione degli errori logici nel codice in runtime e fornisce diversi strumenti per l'analisi delle prestazioni. Visual Studio consente di reperire e installare template e componenti aggiuntivi di terze parti dal Web per ottenere ulteriori funzionalità rispetto a quelle già presenti all'interno del suo codice.

2.2 C++

Nato nel 1983 dall'ingegno di Bjarne Stroustrup, allora ricercatore presso AT&T, C++ è tra i primi 5 linguaggi più utilizzati al mondo. I campi di applicazione sono i più svariati: dal gaming alle applicazioni real-time, dai componenti per sistemi operativi ai software di grafica e musica,

dalle app per cellulari ai sistemi per supercomputer. Praticamente, C++ è ovunque. Negli anni si sono avvicendate diverse versioni di questo linguaggio di programmazione, introducendo via via diverse modifiche ed alcune caratteristiche che ne fanno ancora un linguaggio assolutamente importante e moderno. Il C++ è un linguaggio pensato per la programmazione orientata agli oggetti, che rende questo linguaggio una piattaforma ideale per realizzare progetti di grosse dimensioni, favorendo l'astrazione dei problemi. Ciò ci consente di sviluppare software seguendo i più moderni pattern di progettazione: esistono moltissime librerie già pronte e riutilizzabili, integrabili con i progetti grazie ad opportune strategie e design pattern come Adapter o Facade. C++ mette insieme l'espressività dei linguaggi orientati agli oggetti con l'efficienza e la compattezza del linguaggio C dal quale discende e di cui eredita potenza, flessibilità e la possibilità di programmare a basso livello. Questo ci permette di scrivere programmi di tutte le dimensioni, utilizzando ricchi stack di framework per dialogare ad alto livello, oppure lavorare a basso livello fino ad arrivare all'intergrazione di istruzioni assembly.

2.3 OPENCL

OpenCL (Open Computing Language, tradotto in italiano "linguaggio di calcolo aperto") è un framework basato sul linguaggio ANSI C e C++ con una struttura host-devices che può esser eseguito su una molteplicità di piattaforme, CPU, GPU, e altri tipi di processori. In

particolare, le potenzialità di OpenCL sono bene espresse con architetture altamente parallelizzabili e potenti come le GPU, e in questo caso si parla dell'ambito GPGPU. Sui vari dispositivi viene eseguito il codice del kernel basato sul Linguaggio di programmazione OpenCL C o la versione OpenCL C++ (basati sugli standard C99 e C++14).
Lo standard è stato originariamente proposto da Apple, successivamente ratificato dalla stessa assieme alle principali aziende del settore (Intel, NVIDIA, AMD), e infine portato a compimento dal consorzio no-profit Khronos Group.
Il Khronos Group ha annunciato l'intenzione di unire le librerie OpenCl con le API Vulkan, trasformandole in un’unica piattaforma di sviluppo sia per il settore videoludico che per quello gpgpu.
Molti programmi offrono supporto nativo alle librerie OpenCl, tra cui:

- Adobe Photoshop
- Gimp
- FFmpeg
- Libreoffice Calc
- Microsoft Excel

3 INIZIALIZZAZIONE DEL PROGETTO

Per iniziare a svolgere il seguente progetto inizialmente è stato scaricato un software per permettere alla scheda grafica installata nel mio computer portatile, ovvero una Radeon 500 series, di essere programmata con OpenCL. Il software che è un SDK per AMD, non è più reperibile direttamente reperibile dal sito del produttore della GPU ma su siti terzi.
AMD Accelerated Parallel Processing SDK è il nome del software che installa tutte le librerie del caso e diversi esempi di progetti che utilizzano il calcolo parallelo.

Una volta installato il tutto si procede a controllare se nelle variabili di sistema è presente una nuova variabile dal nome AMDAPPSDKROOT e che il suo valore corrisponda alla cartella dove sono presenti tutti i progetti di esempio.

Andiamo a descrivere Windows bitmap che è un formato dati utilizzato per la rappresentazione di immagini raster sui sistemi operativi Microsoft Windows. Noto soprattutto come formato di file, fu introdotto con Windows 3.0 nel 1990. I bitmap, come sono comunemente chiamati i file d'immagine di questo tipo, hanno generalmente l'estensione .bmp ed ogni pixel viene rappresentato come un array contenente i valori per generare il colore desiderato. Attualmente è poco utilizzato perché uno stesso file salvato in bmp è molto più pesante che se salvato in un qualsiasi altro formato d’immagine.

4 | FASE IMPLEMENTATIVA

Nel progetto si è utilizzata la libreria bitmap **image.hpp**, una libreria che permette la modifica delle immagini in formato BMP. Ricordiamo che una qualsiasi immagine è possibile vederla come una matrice e ogni i-esimo elemento della matrice è un vero e proprio pixel dell'immagine. A sua volta ogni singolo pixel è suddiviso in un array formato da 3 elementi, ossia i rispettivi canali RGB (red green blue). In base ai valori che assumono i 3 record i ogni pixel si andrà a formare un colore che a sua volta andrà a formare l'immagine.

4.1 LIBRERIE INCLUSE

Analizziamo inizialmente tutte le librerie e file che siamo andati ad includere nel nostro progetto.

```
#define _CRT_SECURE_NO_DEPRECATE
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <CL/cl.h>
#include "utils.h"
#include "bmp-utils.h"
#include "gold.h"
```

Le librerie stdio e stdlib sono le classiche librerie di C++, che permettono di stampare a video e utilità generale; math.h permette di utilizzare delle funzionalità matematiche. Più interessante è sottolineare il funzionamento di cl.h che permette di programmare in OpenCL. Le librerie che ci permettono di lavorare con file bmp, che come sappiamo sono delle estensioni di file di tipo immagine, sono utils.h e bmp-utils.h.

4.2 DESCRIZIONE GENERALE DEL CODICE

Analizziamo ora le parti principali. Iniziamo descrivendo la variabile che contiene la funzione “filtro” da applicare alla nostra immagine e come vediamo nell’esempio sottostante

```
static float gaussianBlurFilterFactor = 273.0f;
static float gaussianBlurFilter[25] = {
   1.0f,  4.0f,  7.0f,  4.0f, 1.0f,
   4.0f, 16.0f, 26.0f, 16.0f, 4.0f,
   7.0f, 26.0f, 41.0f, 26.0f, 7.0f,
   4.0f, 16.0f, 26.0f, 16.0f, 4.0f,
   1.0f,  4.0f,  7.0f,  4.0f, 1.0f };
static const int gaussianBlurFilterWidth = 5;
 static float gaussianBlurFilterFactor = 273.0f;
static float gaussianBlurFilter[25] = {
   1.0f,  4.0f,  7.0f,  4.0f, 1.0f,
   4.0f, 16.0f, 26.0f, 16.0f, 4.0f,
   7.0f, 26.0f, 41.0f, 26.0f, 7.0f,
   4.0f, 16.0f, 26.0f, 16.0f, 4.0f,
   1.0f,  4.0f,  7.0f,  4.0f, 1.0f };
static const int gaussianBlurFilterWidth = 5;
```

I cui valori non sono completamente casuali, ma sono stati scelti presi da diversi forum online, perché esistono alcuni effetti convenzionali, come quello sopra riportato, che hanno dei valori appunto “predefiniti”. Il valore che in questo caso prende il nome di gaussianBlurFilterFactor è quanto viene applicato il presente filtro sulla nostra immagine. Sono stati implementati altri filtri, consultabili nella sezione codice, bisogna ricordare che qualsiasi filtro sia creato, anche nuovo di nostra invenzione va inserito nella lista opportuna per essere utilizzato.

Parlando ora del main, le prime cose che si fanno, sono la dichiarazione di tutte le variabili e del percorso per aprire il file contente il codice eseguibile dal kernel con la seguente codice

```
FILE* fp;
char Namefile[] = "./AdvancedConvolution_Kernels.cl";
char* str_source;
size_t size_of_source;
```

Ora tramite uno switch si va “caricare” il filtro scelto dall’utente per poi andare a leggere l’immagine, che è sempre la medesima ma con risoluzioni diverse che vanno dalla 8k fino ad arrivare alla classica immagine HD, sempre comunque in formato bmp, controllando che non vi siano errori.

```
hinputimage = readBmpFloat(inputImagePath, &imageRows, &imageCols);
        printf("imageRows=%d, imageCols=%d\n", imageRows, imageCols);
        printf("filterWidth=%d, \n", filterWidth);

        if (!hinputimage) {
                printf("errore nel caricamento dell'immagine");
                return 1;
        }
```

Ora mi preoccupo di allocare lo spazio per l’immagine di output attesa tramite la funzione malloc, che come sappiamo è una funzione della libreria standard dei linguaggi di programmazione C e C++ per l'allocazione dinamica della memoria. Prima di avviare il kernel mi occorrono dei dati quali:

- Le informazioni del dispositivo
- Le informazioni della piattaforma utilizzata
- OpenCL context
- Creare il comando di coda

Tramite ile funzioni che vengono richiamate, come nell’esempio qua sotto:

```
rescl = clGetPlatformIDs(1, &platforms, &platformCount);
rescl = clGetDeviceIDs(platforms, CL_DEVICE_TYPE_GPU, 1, &device_id,
&platformCount);
context = clCreateContext(NULL, 1, &device_id, NULL, NULL, &rescl);
command_queue = clCreateCommandQueue(context, device_id, 0, &rescl);
```

per poi controllare se vi sono stati errori nel cercare di caricare il file contenente il codice del kernel:

```
fkernel = fopen(Namefile, "r");
        if (!fkernel) {
                fprintf(stderr, "ERROR load kernel.\n");
```

```
                exit(1);
        }
```

Ora vado a creare il programma dal codice sorgente e del kernel sia della cpu che del openCL controllando ogni volta che non vi siano errori. Se tutto va come dovrebbe si passa alla creazione dell'immagine di output vera e propria. Si esegue il kernel e si controlla che non presenti problemi.

```
size_t globalws[2] = { imageCols, imageRows };
        size_t localws[2] = { 8, 8 };
        rescl = clEnqueueNDRangeKernel(command_queue, kernel, 2, NULL, globalws,
localws, 0, NULL, NULL);
        if (rescl != CL_SUCCESS) {
                printf("Failed to enqueue kernel.\n");
                exit(1);
        }
```

Si copia il risultato ottenuto sull'Host in una variabile per poi andare a salvare l'immagine bmp tramite la funzione writeBmpFloat.

```
writeBmpFloat(houtput_image, "cat-filtered.bmp", imageRows, imageCols,
inputImagePath);
```

verifico il risultato ottenuto con la seguente parte di codice

```
float* refOutput = convolutionGoldFloat(hinputimage, imageRows, imageCols,
                filter, filterWidth);

writeBmpFloat(refOutput, "cat-filtered-ref.bmp", imageRows, imageCols,
                inputImagePath);

bool passed = true;
for (i = 0; i < imageRows * imageCols; i++) {
        if (fabsf(refOutput[i] - houtput_image[i]) > 0.001f) {
                passed = false;
        }
}
if (passed) {
        printf("Passed!\n");
}
else {
        printf("Failed!\n");
}
```

Infine libero tutte le risorse che ho utilizzato per applicare il filtro all'immagine.

```
free(refOutput);
free(hinputimage);
free(houtput_image);

clReleaseMemObject(filterBuffer);
clReleaseMemObject(inputImage);
clReleaseMemObject(outputImage);
clReleaseCommandQueue(command_queue);
clReleaseKernel(kernel);
clReleaseProgram(program);
clReleaseContext(context);
```

5 | CONCLUSIONE DEL PROGETTO

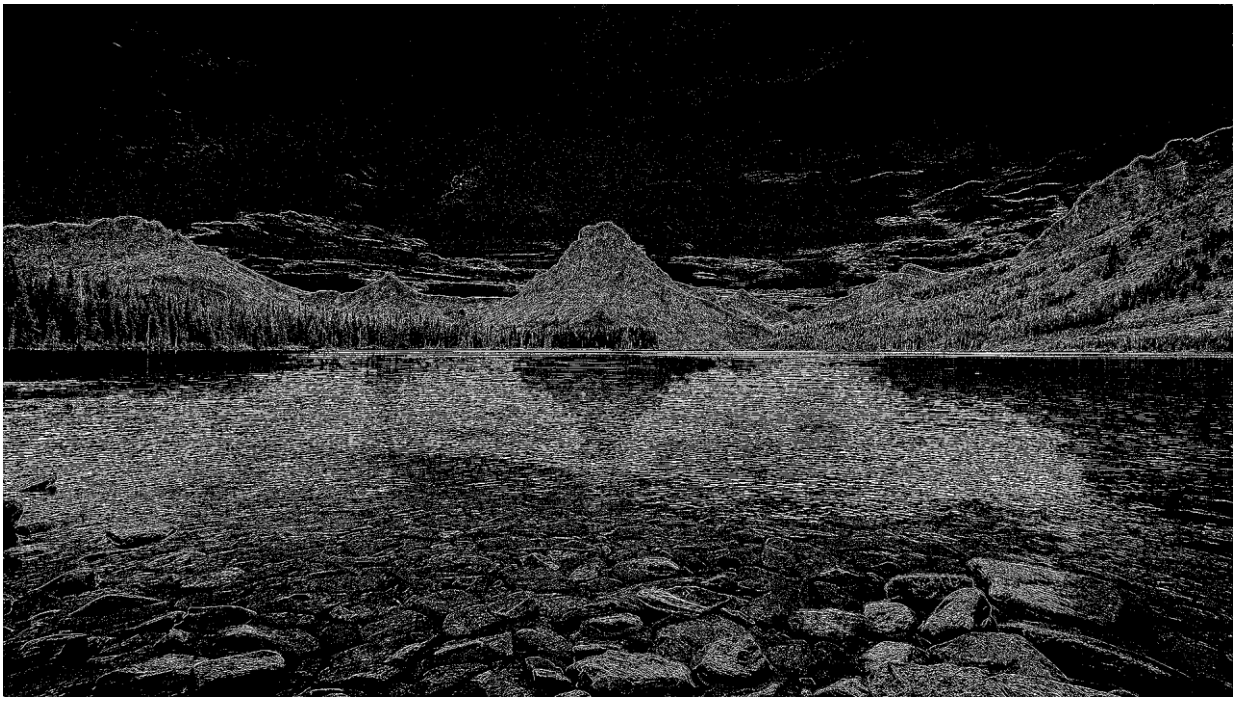
Come precedentemente detto, si vuole andare ad applicare un filtro all'immagine sottostante che non è altro che un file con estensione bpm.

Ora per dimostrare, che il codice prima esposto funziona andiamo ad applicare il filtro Edge filter, che serve ad evidenziare i bordi.
Quindi ad ogni singolo pixel va "applicata", come visto in precedenza, la seguente matrice:

0	1	0
1	-4	1
0	1	0

per ottenere l'immagine sottostante. Ricordiamo come basta modificare un qualsiasi valore della matrice o il valore del focus

per ottenere nuove immagini, anzi nuovi filtri da applicare alle immagini.

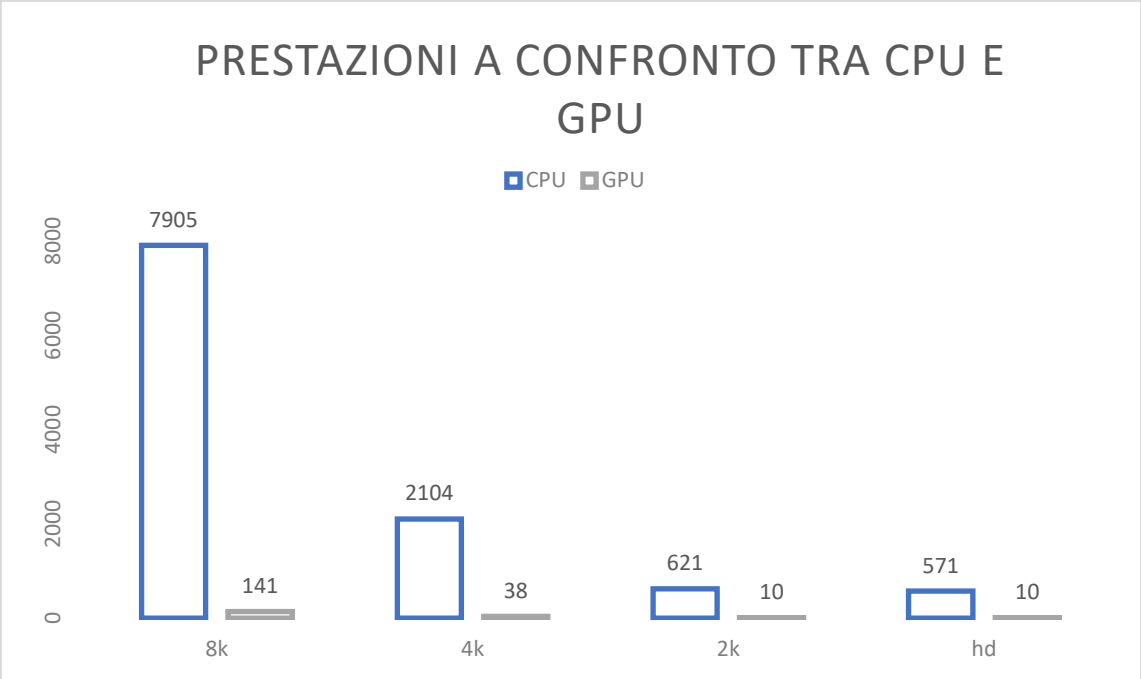


Una volta ottenuta l'immagine con l'effetto desiderato si va a considerare come lo stesso codice sia molto più veloce se svolto dalla GPU rispetto alla CPU. I tempi ottenuti in base alla risoluzione dell'immagine sono riportati nella tabella sottostante.

Risoluzione	Tempo CPU	Tempo GPU
8K	7905 ms	141 ms
4K	2104 ms	38 ms
2K	621 ms	10 ms
HD	571 ms	10 ms

Si può facilmente notare dai dati riportati come il tempo della CPU sia elevatissimo rispetto alla GPU, nel caso delle immagini 8k la GPU è 56 volte più veloce. Per renderci maggiormente conto di quanto è vasta la differenza dei tempi di esecuzione basta dare un rapido sguardo all'istogramma qua sotto dove le colonne blu

rappresentano il tempo della CPU mentre le, quasi invisibili, colonne grigie i tempi della GPU.



Come accennato nella relazione precedente la GPU è un componente hardware veramente potente se sfruttato a dovere come nel calcolo matriciale dove, come in questo caso, i tempi sono drasticamente a favore della scheda grafica. L'avvento di questa componentistica a migliorato tantissimo tutti i programmi che comprendono una qualsiasi funzionalità grafica, basta pensare a come sono migliorati i videogame che ora possono vantare dei frame rate veramente impressionanti.

CODICE DEL PROGRAMMA

Viste le dimensioni del codice non sembra opportuno riempire di codice le prossime pagine ma si allega un link della piattaforma mega.nz dove è possibile consultare e scaricare il codice di tutto il progetto da me sviluppato

https://mega.nz/#F!UYdjXKgA