Programmazione Funzionale e Parallela (A.A. 2017-18)
Corso di Laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

B

Esame del 31/01/2018 – Durata 1h 30' (esonerati)

Inserire nome, cognome e matricola nel file studente.txt.

Esercizio 1 (OpenCL)

Lo scopo dell'esercizio è quello di scrivere un modulo C basato su OpenCL che, date in input due immagini a 256 toni di grigio di dimensione $w1 \times h1$ e $w2 \times h2$, crei una nuova immagine di dimensioni $wo \times ho$, ottenuta mixando le due immagini nella loro intersezione. Ciascun pixel della matrice di output sarà ottenuto come media aritmetica dei pixel corrispondenti nelle due immagini di input. Esempio:


(a) Immagine input 1


(b) Immagine input 2


(b) Immagine output

Si completino i file `mix.c`/`mix.cl` realizzando la funzione `mix` con il seguente prototipo:

```
void mix(unsigned char* in1, int w1, int h1,
         unsigned char* in2, int w2, int h2,
         unsigned char** out, int* ow, int* oh,
         clut_device* dev, double* td)
```

dove:

- `in1`: puntatore a un buffer di dimensione `w1*h1*sizeof(unsigned char)` byte che contiene la prima immagine di input in formato row-major[1];
- `w1`, `h1`: larghezza e altezza di `in1` in pixel (numero colonne della matrice di pixel);
- `in2`: puntatore a un buffer di dimensione `w2*h2*sizeof(unsigned char)` byte che contiene la seconda immagine di input in formato row-major;
- `w2`, `h2`: larghezza e altezza di `in2` in pixel;
- `out`: puntatore a puntatore a buffer di dimensione `min{w1,w2}*min{h1,h2}*sizeof(unsigned char)` byte che deve contenere l'immagine di output in formato row-major; **il buffer deve essere allocato nella funzione mix**;
- `ow`: puntatore a `int` in cui scrivere la larghezza di `out` in pixel;
- `oh`: puntatore a `int` in cui scrivere l'altezza di `out` in pixel.

Per compilare usare il comando `make`. Per effettuare un test usare `make test`. Verrà prodotta l'immagine di output `mix.pgm`.

Nota bene: non usare come nome del kernel `mix`, che è già usato da OpenCL!

---

[1] Cioè con le righe disposte consecutivamente in memoria.

Esercizio 2 (Vettorizzazione SSE)

Lo scopo dell'esercizio è quello di scrivere un modulo C basato su vettorizzazione SSE che, dati in input tre array di short min, v e max, verifica se per ogni indice i comune ai tre array si ha che min[i]<=v[i]<=max[i].

Si completi nel file `inrange/inrange.c` la funzione `inrange` con il seguente prototipo:

```
int inrange(const short* min, unsigned minn,
            const short* v,   unsigned n,
            const short* max, unsigned maxn)
```

che restituisce 1 se per ogni i si ha che che min[i] <= v[i] <= max[i], e 0 altrimenti.

Per compilare usare il comando `make`. Per effettuare un test lanciare l'eseguibile `inrange`.

Suggerimento 1: Ai fini della soluzione si usi:

1) la funzione SSE2:

```
__m128i _mm_cmpgt_epi16 (__m128i a, __m128i b)
```

che restituisce un packed 16-bit integer dst tale che per ogni i in [0,7], dst[i] vale 0xFFFF se **a[i] > b[i]**, e zero altrimenti (dove a[0] denota i primi 16 bit di a, a[1] i secondi 16 bit, ecc.).

2) la funzione SSE2:

```
__m128i _mm_cmpeq_epi16 (__m128i a, __m128i b)
```

che restituisce un packed 16-bit integer dst tale che per ogni i in [0,7], dst[i] vale 0xFFFF se **a[i] = b[i]**, e zero altrimenti (dove a[0] denota i primi 16 bit di a, a[1] i secondi 16 bit, ecc.).

3) La funzione SSE4.1:

```
int _mm_test_all_ones (__m128i a)
```

che restituisce 1 se tutti i bit in a sono 1, e zero altrimenti.

Suggerimento 2: sotto certe condizioni, la somma calcola l'or bit a bit…