Programmazione Funzionale e Parallela (A.A. 2016-2017)
Corso di Laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

A

Esame del 14/02/2017 – Durata 1h 30' (esonerati)

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

Esercizio 1 (Filtri grafici mediante OpenCL)

Lo scopo dell'esercizio è quello di scrivere un modulo C basato su OpenCL che, dati in input un'immagine a 256 toni di grigio di dimensione $w \times h$ e il lato di una mattonella quadrata, crei una nuova immagine in cui le mattonelle quadrate in cui viene decomposta l'immagine di input vengano ricomposte in ordine casuale a formare un puzzle a mosaico. L'esempio seguente è ottenuto con mattonelle 78×78:

(a) Immagine di input (445×243)

(b) Immagine di output (390×234)

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

```
void mosaic(unsigned char* in, int w, int h,
            unsigned char** out, int* ow, int* oh,
            unsigned tile_size, clut_device* dev, double* td);
```

dove:

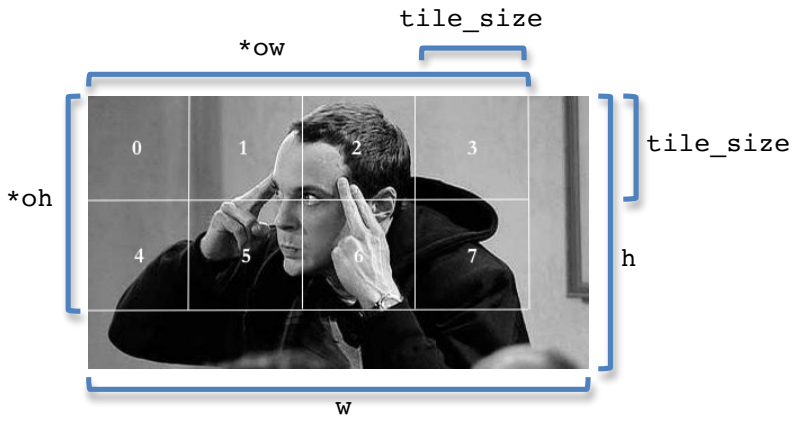
- `in`: puntatore a un buffer di dimensione `w*h` byte che contiene l'immagine di input in formato row-major[1];
- `w`: larghezza di `in` in pixel (numero di colonne della matrice di pixel);
- `h`: altezza di `in` in pixel (numero di righe della matrice di pixel);
- `tile_size`: lato in pixel della mattonella quadrata in cui viene suddivisa l'immagine;
- `ow`: puntatore a oggetto in cui va scritta la larghezza di `*out` in pixel, pari al più grande multiplo di `tile_size` non superiore a `w`;
- `oh`: puntatore a oggetto in cui va scritta l'altezza di `*out` in pixel, pari al più grande multiplo di `tile_size` non superiore ad `h`;
- `out`: puntatore a un oggetto in cui va scritto l'indirizzo di un buffer di dimensione `(*ow)*(*oh)` byte che deve contenere l'immagine di output in formato row-major; **il buffer deve essere allocato nella funzione mosaic.**

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

---

[1] Cioè con le righe disposte consecutivamente in memoria. Si assume inoltre che `sizeof(char)==1`.

Per ricombinare casualmente le mattonelle, assumere che siano numerate per riga come nel seguente esempio:



e permutarne la posizione mediante la funzione `rand_perm` come mostrato nella funzione `mosaic_host` nel file `main.c`, che risolve questo stesso esercizio in modo sequenziale. Si noti che `rand_perm(n)` genera un array con i numeri [0, n-1] permutati casualmente.

Esercizio 2 (Elaborazione di immagini mediante vettorizzazione SSE)

Lo scopo dell'esercizio è quello di scrivere un modulo C basato su vettorizzazione SSE che, dato in input un array di n short e due valori min e max, verifica se tutti gli elementi dell'array sono compresi tra min e max (inclusi).

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

```
int inrange(const short* data, unsigned n,
            short min, short max)
```

che restituisce 1 se tutti gli n elementi x di data sono tali che min <= x <= max, e 0 altrimenti.

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

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 SSE4.1:

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

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