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

B

Esame del 14/02/2017 – Durata 1h 30' (non 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:

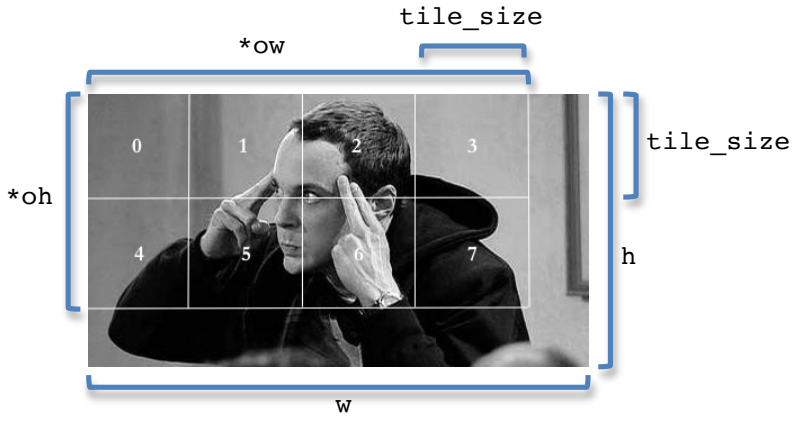
- `in`: puntatore a un buffer di dimensione `w*h` byte che contiene l'immagine di input in formato row-major[1];
- `w`: larghezza di `in` in pixel (numero di colonne della matrice di pixel);
- `h`: altezza di `in` in pixel (numero di righe della matrice di pixel);
- `tile_size`: lato in pixel della mattonella 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

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

prodotta l'immagine di output `sheldon-mosaic.pgm`.

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 (Estrazione sottosequenza)

Estendere la classe List con un metodo project che, data come parametri una lista di Booleani b, restituisce la sottolista ottenuta prendendo tutti e soli gli elementi della lista corrispondenti agli elementi true di b. Ad esempio, se `List(1,2,3,4) project List(true,false, true,true)` vale `List(1,3,4)`.

Se le liste non hanno la stessa lunghezza, l'output sarà ottenuto considerando la più lunga delle due liste troncata alla lunghezza dell'altra. Ad esempio, `List(1,2,3,4) project List(false,true)` è equivalente a `List(1,2) project List(false,true)`.

Scrivere la soluzione in un file `B2.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `B2Main.scala`:

```
import MyList._

object B2Main extends App {

    // test 1
    val l1 = List("uno", "due", "tre", "quattro", "cinque")
    val mask1 = List(true, true, false, true, false)
    val p1 = l1 project mask1
    println(p1 + " [corretto: List(uno, due, quattro)]")

    // test 2
    val l2 = (1 to 10).toList.map(_*3)
    val mask2 = (0 to 9).toList.map(_%2 == 0)
    val p2 = l2 project mask2
    println(p2 + " [corretto: List(3, 9, 15, 21, 27)]")

    // test 3
    val l3 = (1 to 10).toList
    val mask3 = (1 to 20).toList.map(_%3 == 0)
    val p3 = l3 project mask3
    println(p3 + " [corretto: List(3, 6, 9)]")
}
```

La soluzione non deve usare alcun costrutto della programmazione imperativa e in particolare alcuna variabile `var`.

Esercizio 3 (Funzioni di ordine superiore)

Scrivere una funzione `max` che, date due funzioni `f1` ed `f2` da `Int` a `String`, resituisce la funzione che dato `x` restituisce la stringa più lunga fra `f1(x)` ed `f2(x)`.

Scrivere la soluzione in un file `B3.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `B3Main.scala`:

```
object B3Main extends App {

    def f1(i:Int):String = i match {
        case 1 => "monday"
        case 2 => "tuesday"
        case 3 => "wednesday"
        case 4 => "thursday"
        case 5 => "friday"
        case 6 => "saturday"
        case 7 => "sunday"
        case _ => "error"
    }

    def f2(i:Int):String = i match {
        case 1 => "lunedì"
        case 2 => "martedì"
        case 3 => "mercoledì"
        case 4 => "giovedì"
        case 5 => "venerdì"
        case 6 => "sabato"
        case 7 => "domenica"
        case _ => "errore"
    }

    val m1:Int=>String = B3.max(f1,f2)
    val l1 = (1 to 8) map (x => m1(x))
    println(l1.toList +
        " [corretto: List(lunedì, martedì, mercoledì, " +
        "thursday, venerdì, saturday, domenica, errore)]")
}
```

La soluzione non deve usare alcun costrutto della programmazione imperativa e in particolare alcuna variabile `var`.