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

A

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

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

Esercizio 1 (Scala)

Si vuole scrivere un metodo Scala `select` che, data una lista di automobili, trova per ogni anno di immatricolazione di una qualche auto della lista il nome del proprietario più giovane che ha un'auto immatricolata in quell'anno.

Scrivere la soluzione in un file `A1.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `A1Main.scala`:

```
case class Owner(name:String, age:Int)
case class Car(model:String, year:Int, owner:Owner)

object A1Main extends App {
    val cars = List(Car("Tesla Model X", 2017, Owner("Elon", 50)),
                    Car("Open Zafira", 2008, Owner("Anna", 25)),
                    Car("Audi Quattro", 2008, Owner("Ron", 34)),
                    Car("Rover 220 SDI", 2017, Owner("Lucy", 19)))
    val res:List[(Int,String)] = A1.select(cars)
    println(res + " - corretto: List((2017,Lucy),(2008,Anna))")
}
```

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

Esercizio 2 (Scala)

Si vuole definire un nuovo costrutto `mywhile` della forma:

```
mywhile(test, expr) { instr }
```

che ripete le istruzioni `instr` fintantoché `test` è vero. All'inizio di ciascuna iterazione deve essere valutata l'espressione `expr` e stampato il risultato.

Scrivere la soluzione in un file `A2.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `A2Main.scala`:

```
import A2._

object A2Main extends App {
    var i = 0
    mywhile(i<5, "i="+i) {
        i+=1
    }
}
```

Il programma deve stampare:

```
i=0
i=1
i=2
i=3
i=4
```

Esercizio 3 (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.