Secondo esonero – 22 Dicembre 2015

Domanda 1. Discutete in modo sintetico i concetti di race condition, data race, bad interleaving, mutua esclusione e deadlock.

Domanda 2. Considerate la seguente classe `Counter`, che ha un singolo attributo `value` e tre metodi per ottenere, impostare e incrementare il valore del contatore:

```
class Counter {
    private int value = 0;
    public int getValue() { return value; }
    public void setValue(int someValue) { value = someValue; }
    public void increment() { value++; }
}
```

(a) Supponete di avere un oggetto `x` di tipo `Counter` e due thread. Il thread 1 esegue le seguenti istruzioni:

```
x.setValue(0);
x.increment();
int i = x.getValue();
```

e il thread 2 esegue `x.setValue(2)`. Quali sono i possibili valori di `i`? Per ciascuno dei valori possibili, mostrate un interleaving delle istruzioni che lo genera.

(b) Immaginate ora di avere una classe `SynchCounter` uguale in tutto e per tutto a `Counter`, ma i cui metodi siano tutti `synchronized`. Rispondete alle seguenti domande:

- Nello scenario di esecuzione descritto al punto (a), usando la classe `SynchCounter` i possibili valori di `i` cambiano? Motivare la risposta.
- Supponete ora un diverso scenario di esecuzione, in cui sia il thread 1 che il thread 2 eseguono `x.increment()`, con il valore del contatore originariamente a 0.
  - Se `x` è un oggetto di tipo `SynchCounter`, l'esecuzione è deterministica? Ovvero, il valore del contatore sarà sempre pari a 2 dopo l'esecuzione dei due incrementi?
  - E se `x` fosse un oggetto di tipo `Counter`? Pensate a come l'istruzione `value++` si traduce in termini di bytecode: a quali operazioni di basso livello corrisponde l'operatore `++` e perché ragionare in termini di bytecode è importante?

(c) Assumete ora che un programmatore (non particolarmente furbo) decida di implementare, esternamente alla classe `SynchCounter`, il proprio metodo `setToOne()` nel seguente modo:

```
void setToOne(SynchCounter x) {
   x.setValue(0);
   x.increment();
}
```

Rispondete alle seguenti domande, motivando le risposte:

- Il codice è soggetto a data race?
- Il codice è soggetto a bad interleaving?
- Le risposte sarebbero diverse se `setToOne()` fosse `synchronized`?

Domanda 3. In questo esercizio si vuole calcolare lo speculare di un'immagine in OpenCL.

(a) Scrivete un *kernel* che, date una matrice di input IN di dimensioni $w \times h$ che rappresenta un'immagine a toni di grigio (0-255) e una matrice di output OUT delle stesse dimensioni, calcoli OUT come speculare di IN:

IN =  OUT = 

Assumete un NDRange di dimensioni $[0, w-1] \times [0, h-1]$.

(b) Sebbene l'esercizio non richieda di scrivere il codice host, discutete gli aspetti connessi al layout di memoria soggiacente: dove sono memorizzate le matrici IN e OUT? In che modo sono state allocate? Quali strutture dati devono essere gestite sul lato host? Come e in quali momenti avviene il trasferimento di dati (e quindi la comunicazione) tra host e device?