Prova d'esame – 2 Febbraio 2016

Parte 1

Domanda 1. Due programmatori implementano due diversi algoritmi per risolvere lo stesso problema su input di dimensione n. L'algoritmo $\mathcal{A}_1$ esegue n^3 operazioni e il 40% del suo tempo di esecuzione è seriale. L'algoritmo $\mathcal{A}_2$ esegue n^4 operazioni, ma solo lo 0.1% del suo tempo di esecuzione è seriale.

1. Se $\mathcal{A}_1$ viene eseguito su una piattaforma con 2 processori e $n = 10^2$, di quanti processori P abbiamo bisogno per fare in modo che l'esecuzione di $\mathcal{A}_2$ sullo stesso input risulti più veloce usando P processori?
2. Qual è lo speedup di $\mathcal{A}_2$ su 100 processori in base alla legge di Amdahl?
3. Nel calcolo dello speedup, è importante che il tempo di esecuzione di $\mathcal{A}_2$ sia n^4? Spiegare perché.

Domanda 2. Il seguente codice Java calcola lo speculare di un array:

```
static int[] reverse(int[] array) {
    int [] answer = new int[array.length];
    fjPool.invoke(new Reverse(answer,array,0,array.length));
    return answer;
}

class Reverse extends RecursiveAction {
    int[] out;
    int[] in;
    int lo;
    int hi;
    Reverse(int[] o, int[] i, int l, int h) {
        out = o; in = i; lo = l; hi = h;
    }
    public void compute() {
        if(hi==lo+1) out[in.length-lo-1] = in[lo];
        else {
            Reverse left  = new Reverse(out,in,lo,(hi+lo)/2);
            Reverse right = new Reverse(out,in,(hi+lo)/2,hi);
            left.fork();
            right.fork();
            right.join();
        }
    }
}
```

a) Il codice contiene un errore: quale?

b) Disegnate il DAG di esecuzione per $n = 4$, etichettando i nodi del DAG in modo da individuare il thread associato a ciascun nodo e la porzione di array su cui il thread opera.

c) Discutete work e span dell'algoritmo in funzione del numero n di elementi dell'array.

d) Spiegate in modo sintetico come ottimizzare il codice.

e) Se $n = 2^{30}$, quanti thread fork-join sono creati dal codice (opportunamente corretto come descritto al punto a)? Applicando un cut-off sequential di 1024, quanti thread fork-join sarebbero creati?

f) Supponete ora di voler memorizzare l'array speculare *in loco*, ovvero di non avere un array di output ma di voler modificare direttamente l'array di input. Come modifichereste il codice in tale scenario?

Domanda 3. Descrivete l'algoritmo parallelo per la fusione di due array ordinati (`merge`). Illustrate sinteticamente un esempio di esecuzione e analizzate work e span, assumendo che i due array in input contengano n elementi ciascuno (è sufficiente spiegare come impostare le equazioni di ricorrenza, senza svolgerle esplicitamente).

Prova d'esame – 2 Febbraio 2016

Parte 2

Domanda 4. Considerate il seguente programma Java, il cui scopo è di mantenere il numero di posti liberi in un garage, regolando le entrate di nuove auto:

```
class ParkingGarage {
    private int places;

    public ParkingGarage(int places) {
        if(places < 0) places = 0;
        this.places = places;
    }

    // enter parking garage
    public synchronized void enter() {
        while (places == 0);
        places--;
    }

    // leave parking garage
    public synchronized void leave() { places++;}
}
```

Durante l'esecuzione, il numero `places` di posti liberi deve essere sempre ≥ 0. Rispondete alla seguenti domande, motivando le risposte:

1. Il programma soffre di race condition? Se si, di che tipo?
2. Mostrate un interleaving delle istruzioni che porta il programma a non terminare la sua esecuzione.
3. Definireste la non-terminazione descritta al punto 2 come un deadlock? Perché?
4. Per ovviare al problema descritto al punto 2, un programmatore modifica il codice come segue (il costruttore e il metodo `leave` sono inalterati):

```
[...]
private synchronized boolean isFull() { return (places == 0); }
private synchronized void reducePlaces() { places--; }

public void enter() {
    while ( isFull() );
    reducePlaces();
}
```

   Il nuovo programma soffre di race condition? Il nuovo programma termina sempre correttamente la sua esecuzione?

5. Date un'implementazione corretta della classe `ParkingGarage`, preferibilmente senza attesa passiva (spin waiting).

Domanda 5. Dite se le seguenti affermazioni sono vere o false, motivando le risposte.

a) La keyword `synchronized` garantisce sia la mutua esclusione degli accessi ad un oggetto che l'atomicità della sequenza di istruzioni di accesso.

b) Se tutti i metodi `set` che scrivono gli attributi di un oggetto sono `synchronized`, non è necessario rendere `synchronized` anche i metodi `get` che leggono gli attributi dell'oggetto.

c) Per avere un deadlock un programma deve accedere in mutua esclusione ad almeno due oggetti diversi.

d) Un programma che usa un meccanismo di locking consistente può avere data race.

e) Un programma che usa un meccanismo di locking consistente può avere bad interleaving.

Domanda 6. Scrivete un kernel OpenCL che riceve un array A di interi di dimensione n e calcola un array B di `float` di dimensione $\lfloor n/2 \rfloor$. B è tale che, per ogni posizione dispari p in A, $B[p/2]$ contiene la media degli elementi $A[p-1]$, $A[p]$ e $A[p+1]$, se esistono. Discutete qual è lo spazio di indicizzazione e il numero di work item eseguiti.