Prova d'esame – 13 Gennaio 2016

Parte 1

Domanda 1. Rispondete alle seguenti domande, motivando le risposte:

a) Un programma ha efficienza 50% usando 16 processori. Qual è la percentuale di codice seriale del programma in base alla legge di Amdahl?

b) Se un programma $\mathcal{P}_1$ ha uno speedup maggiore di un programma $\mathcal{P}_2$ e i due programmi eseguono lo stesso work, è vero che $\mathcal{P}_1$ è sempre veloce almeno quanto $\mathcal{P}_2$? Se l'affermazione è vera dimostratela, altrimenti fornite un controesempio.

c) Illustrate la legge di Gustafson, dimostrandola e spiegandone le implicazioni.

Domanda 2. Immaginate di avere una classe `Worker1` contenente un metodo pubblico `doTask1()`, una classe `Worker2` contenente un metodo pubblico `doTask2()`, ecc. Supponete inoltre che la vostra applicazione crei quattro thread τ_1 ... τ_4 i cui metodi `run()` contengono il seguente codice:

τ_1: `worker1.doTask1(); worker2.doTask2();`
τ_2: `worker5.doTask5();`
τ_3: `worker3.doTask3();`
τ_4: `worker4.doTask4();`

dove le esecuzioni di `doTask1()`, `doTask3()` e `doTask4()` richiedono 20 unità di tempo ciascuna, `doTask2()` ne richiede 30 e `doTask5()` 40. Infine, assumete che il thread `main` esegua il seguente frammento di codice:

1. τ_1`.start()`
2. τ_2`.start()`
3. τ_2`.join()`
4. τ_3`.start()`
5. τ_1`.join()`
6. τ_3`.join()`
7. τ_4`.start()`

Rispondete alle seguenti domande, motivando le risposte:

A) Qual è il minimo tempo per l'esecuzione di questo programma? Mostrate un interleaving delle esecuzioni dei metodi `doTask1()` ... `doTask5()` che permetta di ottenere il minimo tempo di esecuzione.

B) Il minimo tempo per l'esecuzione sarebbe diverso se le righe 5 e 6 nel `main` fossero invertite? Perché?

C) Avendo un numero sufficiente di processori, potrebbe accadere che `doTask4()` sia eseguito in parallelo a `doTask1()`? Perché?

D) Avendo un numero sufficiente di processori, potrebbe accadere che `doTask3()` sia eseguito in parallelo a `doTask1()`? Perché?

Domanda 3. Descrivete il problema del packing, presentate un algoritmo di packing parallelo ed analizzatene le prestazioni. Discutete inoltre l'applicazione di tale algoritmo al quicksort, illustrando l'implementazione di partition ed analizzando work, span e parallelismo ottenuto nel caso migliore.

Prova d'esame – 13 Gennaio 2016

Parte 2

Domanda 4. Considerate il seguente programma Java:

```
public class Application
extends Thread {
  public static X x;
  public static Y y;
  public static Z z;

  public void run() {
    z = new Z(); x = new X(z);
    y = new Y(z);
    System.out.print("C");
    execute1();
    z.h();
    execute2();
  }

  public void execute1() {
    System.out.print("A");
    x.start();
  }

  public void execute2() {
    y.start();
    System.out.print("L");
  }
}
```

```
class X
extends Thread {
  public Z z;

  public X(Z zz) {
    z = zz;
    z.n = 0;
  }

  public void run() {
    synchronized (z) {
      z.n = 1;
      z.notify();
      System.out.print("K");
    }
  }
}
```

```
class Y
extends Thread {
  public Z z;

  public Y(Z zz) {
    z = zz;
  }

  public void run() {
    System.out.print("J");
    synchronized (z) {
      while (z.n == 0) {
        try {
          z.wait();
        } catch (
            InterruptedException
              e) {};
      }
      System.out.print("Q");
    }
  }
}
```

```
public class Root {
  public static void main(String[]
      args) {
    Application app = new
        Application();
    app.start();
  }
}
```

```
class Z {
  public int n;
  public void h() {
    System.out.print("P");
  }
}
```

Rispondete alle seguenti domande, motivando le risposte:

A) Illustrate almeno tre possibili output per il programma e i relativi interleaving delle istruzioni Java che li producono.

B) Dite se le seguenti affermazioni sono vere o false:

B_1) "A" è sempre stampata per seconda.

B_2) "K" è sempre stampata dopo "P".

B_3) "J" è sempre stampata dopo "P".

B_4) "Q" può essere stampata prima di "K".

C) Supponete che il metodo `h()` nella classe `Z` sia modificato come segue:

```
public void h() {
    n = 100;
    System.out.print("P");
}
```

Il programma così modificato soffre di race condition? Il programma termina sempre la sua esecuzione producendo una stampa?

D) Supponete ora che il metodo `h()` nella classe `Z` sia modificato come segue:

```
synchronized void h() {
    n = 0;
    System.out.print("P");
}
```

Il programma così modificato soffre di race condition? Il programma termina sempre la sua esecuzione producendo una stampa?

Domanda 5. Spiegate la differenza tra strategie di locking coarse-grained e fine-grained usando come esempio l'implementazione di liste concorrenti.

Domanda 6. Si vogliono trovare tutte le occorrenze di una stringa pattern all'interno di una stringa di ricerca. Si scriva il codice di un kernel OpenCL per risolvere tale problema, assumendo che il kernel abbia la seguente intestazione:

```
void __kernel PatternMatcher(
                    __global char* pattern_string, const unsigned long pattern_length,
                    __global char* buffer_string,  const unsigned long buffer_length,
                    __global unsigned int* results)
```

L'array `results` calcolato in output ha lunghezza `buffer_length` ed è tale che `results[i]=1` se le posizioni da `i` a `i+pattern_length-1` dell'array `buffer_string` contengono i caratteri della stringa `pattern_string`, 0 altrimenti.