Prova d'esame – 10 Gennaio 2017

Parte 1

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

1. Un programma con una frazione seriale del 5% ha efficienza 80% se eseguito su P processori. Quanto vale P secondo la legge di Amdahl?

2. Se un programma parallelo $\mathcal{P}_1$ ha work minore di un programma $\mathcal{P}_2$, è vero che $\mathcal{P}_1$ è sempre più veloce di $\mathcal{P}_2$ se eseguito su $P \geq 1$ processori? Se l'affermazione è vera dimostratela, altrimenti spiegate perché è falsa e fornite un controesempio.

3. Considerate un problema che può essere risolto in tempo n^3 (per esempio, la moltiplicazione di due matrici quadrate di lato n) e supponete che l'unità di tempo sia un nanosecondo (cioè 10^{-9} secondi). Quindi ad esempio, per $n = 1000$, il calcolo richiederebbe $1000^3 \times 1\,ns = 1\,s$. Assumete di avere un algoritmo parallelo perfettamente scalabile, ovvero con tempo di esecuzione n^3/P su P processori.

   3a) Quanto sono grandi gli input che possono essere gestiti in 1 secondo assumendo di avere rispettivamente 8, 1000, 1 milione di processori disponibili?

   3b) Quanti processori sarebbero necessari per risolvere in un anno il problema se $n = 10^7$ (10 milioni)?

   3c) Assumete ora che l'algoritmo sia inefficiente di un fattore $\log n$. Ad esempio, per $n = 1000$ e $P = 100$, inefficienza $\log n$ implica che il calcolo richiederebbe $1000^3 \times \log 1000 \times 1\,ns \,/\, 100 = 0.1\,s$ (si assuma per semplicità che $\log 1000 \approx 10$). Quanti processori sarebbero necessari per risolvere in un anno il problema se $n = 10^7$?

Domanda 2. Supponete di avere un array di parentesi aperte e chiuse e di voler sapere se c'è un matching tra le parentesi o meno. Ad esempio:

- per la sequenza "((()))()" c'è un matching;
- per le sequenze ")(())()(" o "((()))(()" non c'è un matching.

Usando le operazioni parallele note (ad esempio map, reduce, somme prefisse) mostrate come risolvere il problema in tempo $O(n/P + \log P)$ su un sistema a P processori. Analizzate le prestazioni dell'algoritmo proposto.

Domanda 3. Descrivete l'implementazione parallela dell'algoritmo mergesort: analizzate le prestazioni della parallelizzazione naive (con merge sequenziale), presentate poi l'algoritmo per la fusione in parallelo di due array e discutetene l'integrazione nel mergesort. Analizzate work e span degli algoritmi considerati.

Prova d'esame – 10 Gennaio 2017

Parte 2

Domanda 4. Discutete il concetto di attesa passiva e i meccanismi offerti da Java per implementarlo.

Domanda 5. Considerate la classe `Counter` e due possibili usi mostrati nelle classi `Example1` ed `Example2`:

```
public class Counter{
    long count = 0;
    public void add(long value){  this.count += value;  }
}

public class CounterThread extends Thread{
    protected Counter counter = null;
    public CounterThread(Counter counter){  this.counter = counter;  }

    public void run() {
        for(int i=0; i<10; i++) counter.add(i);
    }
}

public class Example1 {
    public static void main(String[] args){
        Counter counterA = new Counter();
        Counter counterB = new Counter();
        Thread  threadA = new CounterThread(counterA);
        Thread  threadB = new CounterThread(counterB);

        threadA.start();
        threadB.start();
    }
}

public class Example2 {
    public static void main(String[] args){
        Counter counter = new Counter();
        Thread  threadA = new CounterThread(counter);
        Thread  threadB = new CounterThread(counter);

        threadA.start();
        threadB.start();
    }
}
```

Rispondete alle seguenti domande, considerando sia il codice di `Example1` che quello di `Example2` e motivando sempre le risposte:

5a) Il codice soffre di data race? Spiegate perché e, in caso di risposta affermativa, descrivete come e dove usare `synchronized` per renderlo corretto.

5b) Qual è il valore del contatore al termine dell'esecuzione?

5c) Caratterizzate i possibili interleaving delle istruzioni, focalizzandovi sul metodo `add` e discutendo come varia il valore del contatore durante l'esecuzione.

5d) La risposta sarebbe stata diversa se invece di richiamare il metodo `start` su `threadA` e `threadB` avessimo invocato `run`?

5e) I 10 incrementi del contatore nel metodo `run` della classe `CounterThread` sono atomici? In caso di risposta negativa, discutete come modificare il codice per renderli atomici.

5f) La keyword `volatile` potrebbe essere usata nell'implementazione di questo contatore? Se si, discutete se una implementazione tramite `volatile` sia più o meno efficiente di quella basata su `synchronized`.

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

```
void __kernel PatternMatcher(__global char* s,
                             const unsigned long len,
                             __global unsigned int* results)
```

L'array `results` calcolato in output ha lunghezza `len` ed è tale che `results[i]=1` se le posizioni da `i` a `i+4` dell'array `s` contengono i caratteri della stringa "abcde", 0 altrimenti.

Bonus. Come procedereste se, invece di `results`, voleste restituire l'array di tutte le posizioni iniziali della stringa "abcde" all'interno di `s` ? Quali primitive potrebbero tornare utili? Potete immaginare di usare l'array `results` come risultato intermedio.