Concorrenza

Concorrenza: Aspetti teorici

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Concorrenza e parallelismo

- Terminologia
  - Concorrenza
    - Atto di eseguire più calcoli nello stesso intervallo di tempo
    - Task multipli (due o più) vengono eseguiti negli stessi intervalli di tempo, senza ordine specifico particolare
    - L'esecuzione dei task sembra contemporanea, ma non lo è
    - L'effetto è dovuto al time-slicing della CPU, ovvero allo scheduler (context switching) che dedica l'unità di calcolo ai vari task per unità di tempo infinitesimali

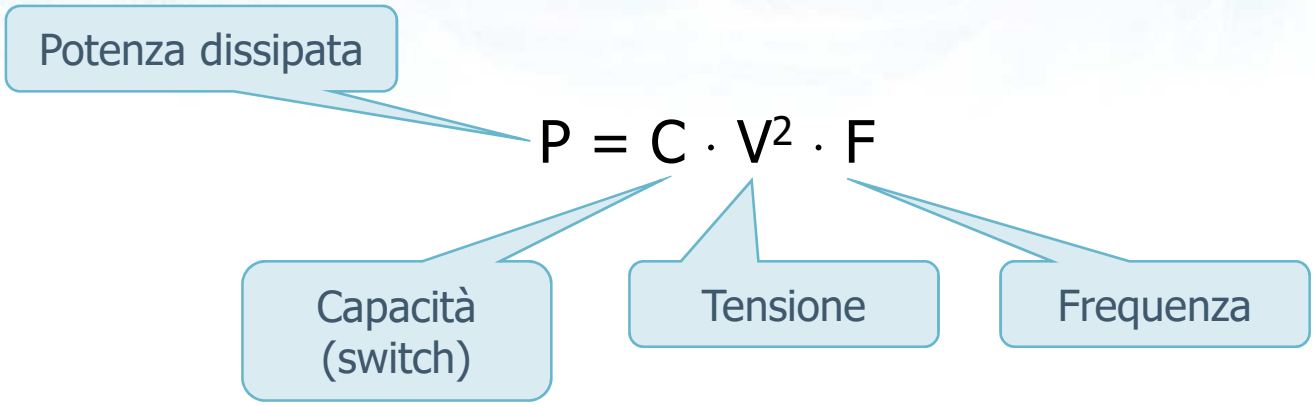
Concorrenza e parallelismo

- Parallelismo
  - Atto di eseguire più calcoli simultaneamente
  - Task multipli (o parti diverse dello stesso task) vengono eseguiti (eseguite) contemporaneamente in sistemi multi-processori o multi-core
  - Viene permesso da strutture hardware apposite (multi-CPU o multi-core)
  - Si ottiene trasformando un flusso di esecuzione sequenziale in uno parallelo

Concorrenza e parallelismo

- Concorrenza e parallelismo non identificano la stessa cosa
  - Concorrenza = espletare azioni nello stesso intervallo di tempo dando l'illusione della simultaneità
  - Parallelismo = espletare azioni nello stesso istante
  - I termini spesso si usano in maniera intercambiabile
- I due concetti esistono da molti anni
  - Lo sviluppo è stato sorprendente dopo il 2004
  - Nel 2004 Intel cancella i processori Tejas e Jayhaw
    - L'aumento della frequenze (clock) dei processori raggiunse i suoi limiti a causa dell'eccessivo consumo di potenza

Concorrenza e parallelismo



- A causa della fine del "frequency scaling"
  - Il parallelistmo è diventato uno dei principali paradigmi della programmazione
  - La programmazione parallela ha fatto nascere nuove sfide e nuove insidie (bug)

Concorrenza e parallelismo

- Questi cambiamenti hanno portato a cambiamenti analoghi nei paradigmi utilizzati
  - Prima legge di Moore (1965)
    - Il numero di transistori nei processori raddoppierà ogni 12 mesi
      - 24 mesi negli anni '80
      - 18 mesi negli anni '90
    - Esempio
      - Maggio 1997, Pentium II, 7.5 milioni transistor, f_{clock} = 300 MHz
      - Novembre 2000, Pentium 4, 42 milioni di transistor, f_{clock} = 1.5 GHz

Concorrenza e parallelismo

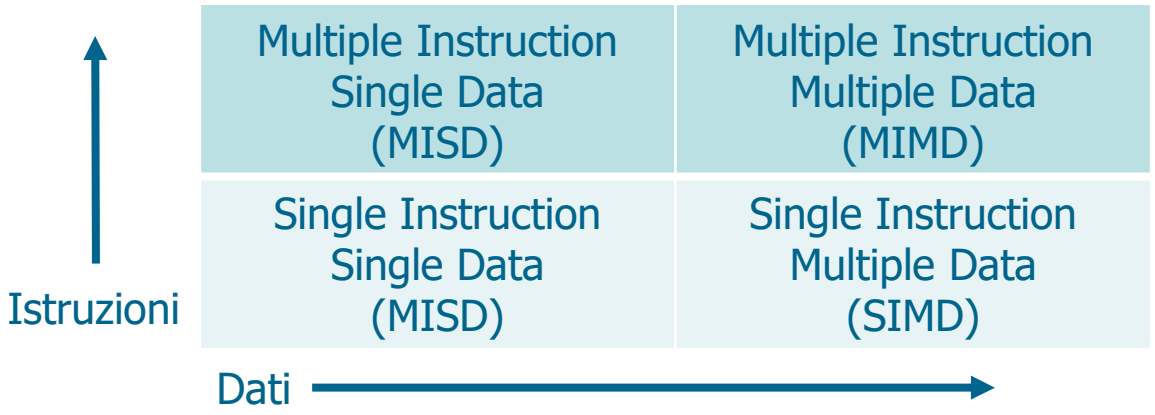
- Legge di Bill Dally (NVIDIA, 2010, Forbes)
  - Seguire la legge di Moore non ha più senso
  - Possiamo aumentare il numero di transistor e dei core di 4 volte ogni 3 anni. Facendo lavorare ogni core leggermente più lentamente, perciò in maniera più efficiente, possiamo più che triplicare le prestazioni mantenendo lo stesso consumo totale

Architetture parallele

- Esistono diverse tipologie di parallelismo
  - Bit-level
    - La lunghezza di una "word" determina l' "efficacia" di una istruzione (e.g., adder a 8 vs 16 bit)
  - Instruction-level
    - Utilizzo di "multi-stage" pipelines per l'esecuzione del flusso di istruzioni (e.g., fetch, decode, execute)
  - Task-level
    - Computazioni distinte sono eseguite in parallelo (e.g., un ordinamento e un prodotto matriciale sono eseguiti contestualmente)

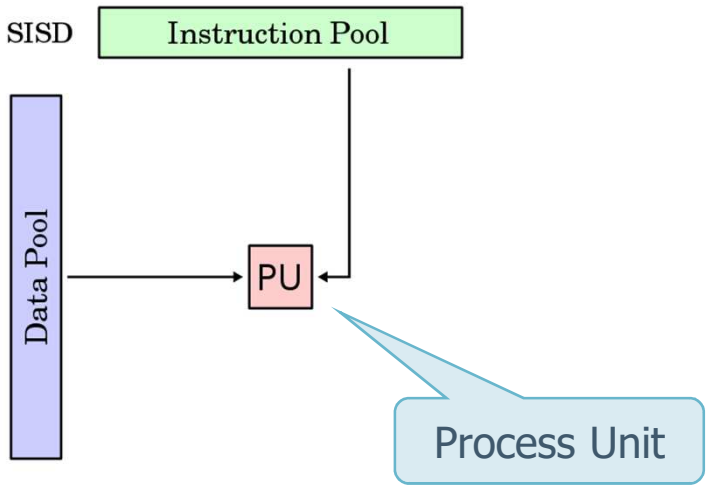
Architetture parallele

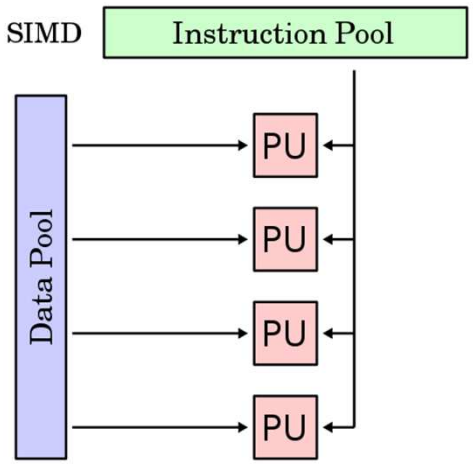
- La prima classificazione per architetture parallele è stata introdotta da Flynn (1966)
  - Parzialmente superata
    - Molte architetture sono miste o non classificabili direttamente
  - Ancora ampiamente utilizzata
    - Semplice e comprensibile



Architetture parallele

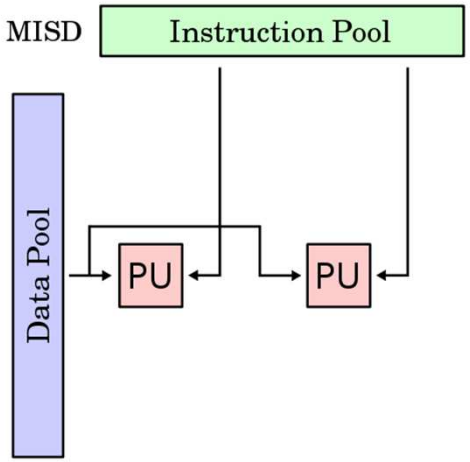
- Single Instruction Single Data (SISD)
  - Schema classico
  - No parallelismo
- Single Instruction Multiple Data (SIMD)
  - Una istruzione singola opera su diversi flussi di dati

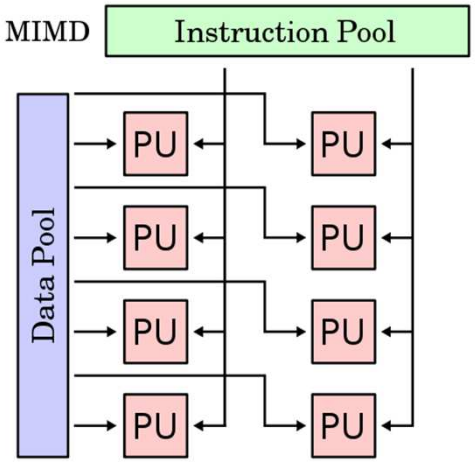




Architetture parallele

- Multiple Instruction Single Data (MISD)
  - Istruzioni multiple operano su un singolo flusso di dati
- Multiple Instruction Multiple Data (MIMD)
  - Istruzioni multiple operano su diversi flussi di dati





Speed-up

- Lo scopo principale del parallelismo è l'aumento dell'efficienza
  - Come valutiamo l'efficienza?
  - Valutazioni possibili
    - Tempo
    - Memoria
- Esistono diverse metriche per il calcolo dei tempi di esecuzione

Speed-up

- User time
  - Tempo totale che la CPU dedica a eseguire un determinato task a livello utente
    - Non tiene conto del tempo "perduto" nella gestione, e.g., operazioni di I/O, esecuzione di routine a livello kernel, etc.

Speed-up

- CPU time
  - Tempo totale dedicato dalla CPU all'esecuzione di un task
    - Questo tempo include i tempi di I/O, la gestione del task da parte del kernel, etc.
  - Su una architettura parallela occorrerà valutare il tempo dedicato al task da parte di tutte le unità di calcolo
    - Nella maggior parte dei casi tale tempo sarà maggiore del tempo del processo sequenziale in esecuzione su un processore unico
    - Nei confronto in base allo user o al CPU time le architetture parallele sono tendenzialmente peggiori

Speed-up

- Wall clock time
  - Anche detto elapsed time (tempo trascorso)
  - Tempo effettivamente richiesto per terminare un determinato compito
  - Ovvero
    - Wall clock time = tempo di fine – tempo di inizio task
  - Questo è equivalente a "cronometrare" il tempo di esecuzione indifferentemente dal fatto tale esecuzione sia portata avanti su sistemi mono o multi processore
  - Prendendo tale metrica come riferimento
    - $speedup = \frac{Wall\text{-}clock\ time\ sequential\ algorithm}{Wall\text{-}clock\ time\ parallel\ algorithm}$

Speed-up

- In teoria, raddoppiando il numero di elementi di processamento dovrebbe dimezzarsi il wall-clock time
  - I vantaggi dovrebbero essere lineari
- Tale comportamento si ottiene
  - Raramente
    - Se un processo non è parallelizzabile aumentare il numero di processori/core non modifica il run-time
  - Solo per un numero ridotto di processori e core
    - Dopo una dipendenza lineare iniziale, la curva dello speed-up si appiattisce (asintoto orizzontale)

Legge di Amdhal

- I vantaggi teorici ottenibili mediante concorrenza sono stati analizzati per la prima volta da Amdhal
- La legge di Amdhal [1967] specifica il miglioramento teorico ottenibile tramite parallelismo
  - $speedup = \frac{1}{S+\frac{1-S}{n}}$

S = percentuale dell'elapsed time trascorso ad eseguire la parte sequenziale di un programma (non parallelizzabile)

n = numero di processori o core

Legge di Amdhal

- Alla legge di Ahmdal occorrerebbe aggiungere l'overhead relativo alla gestione degli n thread (o processi)
  - $speedup = \frac{1}{S+\frac{1-S}{n}+H(n)}$

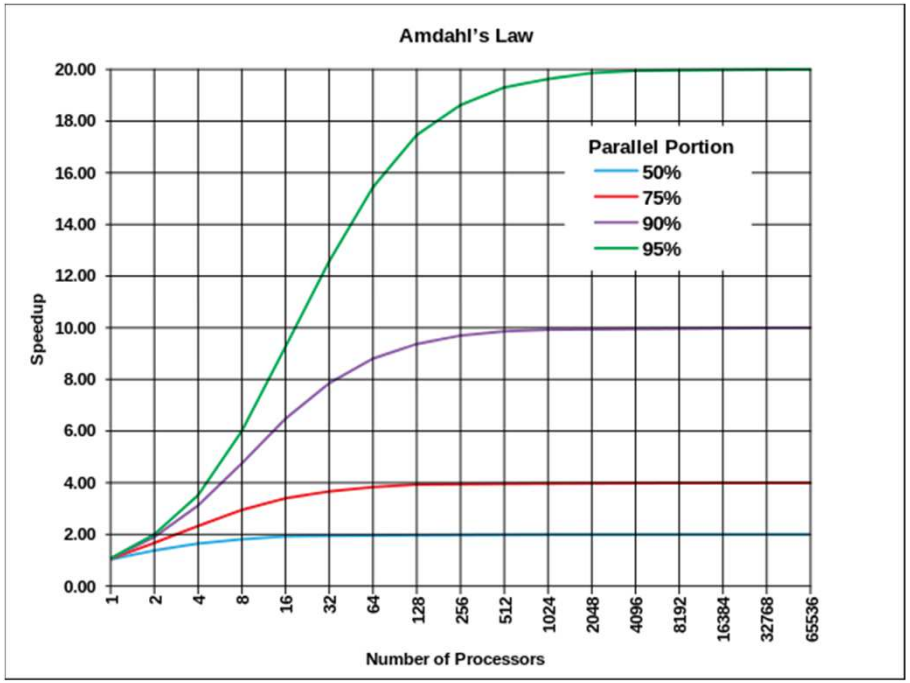
H(n) overhead di gestione: del sistema operativo e di sincronizzazione inter-thread

- Nell'ipotesi più ottimistica H(n)=0 e all'aumentare del numero di processori si ha
  - $speedup = \lim_{n\to\infty} \frac{1}{S+\frac{1-S}{n}} = \frac{1}{S}$

S = 10% → $speedup_{massimo}$ = 10
S = 20% → $speedup_{massimo}$ = 5
...
S = 50% → $speedup_{massimo}$ = 2

Legge di Amdhal

- Legge di Amdahl [1967]
  - Piccole porzioni di programma non parallelizzabili limitano lo speed-up complessivo ottenibile



Legge di Amdhal

- Corollario alla legge di Amdhal
  - Diminuire la parte serializzabile e aumentare la parte parallelizzabile è più importante che aumentare il numero di processori
- A seguito della legge di Amhdal l'utilizzo del parallelismo è stato oggetto di critiche per molti anni

Legge di Amdhal

- Limiti della legge di Amhdal
  - I limiti non dipendono solo dalla disponibilità di cicli di CPU, ma anche da altri fattori
    - I sistemi multi-core possono avere molteplici cache abbassando la latenza della memoria e aumentando l'efficienza del sistema
    - Alcuni algoritmi hanno formulazioni parallele migliori ovvero con un numero minore di passi computazionali
    - Amhdal assume la dimensione dei problemi rimanga costante, mentre in genere aumenta con l'aumentare delle risorse disponibili e ciò che rimane costante è il tempo di esecuzione

Legge di Gustafson

- Negli anni '80 presso i Sandia National Lab sono stati ottenuti speed-up lineari con 1024 processori su applicazioni sulle quali Amhdal avrebbe previsto un comportamento non lineare
- La legge di Gustafson (o equazione di Barsis) prevede uno speed-up lineare
  - $speedup = n + (1 - n) \cdot S$

n = numero di processori
S = frazione tempo spesa nella parte sequenziale

Parallelizzazione

- La parallelizzazione di un algoritmo può essere effettuata solo mediante decomposizione
  - Task decomposition
    - Decomposizione del programma in funzioni e analisi di quali funzioni possono agire in parallelo
  - Data decomposition
    - Decomposione del problema in base al parallelismo applicabile sui dati piuttosto che delle sua natura funzionale e logica
  - Data flow decomposition
    - Decomposizione del problema in base al flusso di dati tra le varie funzioni o dei task da completare

Grafo di precedenza

- La decomposizione di un problema può essere effettuata solo conoscendone le dipendenze
- I vincoli di precedenza possono essere rappresentati mediante **grafo di precedenza**

Relazione con il Control Flow Graph (CFG) e con gli Alberi di generazione dei processi

Grafo di precedenza

- Un grafo di precedenza è un grafo aciclico diretto in cui
  - I vertici corrispondono a istruzioni singole, blocchi di istruzioni, processi
  - Gli archi corrispondono a condizioni di precedenza
    - Un arco dal vertice A al vertice B significa che B può essere eseguito solo una volta terminato A
  - La precedenza può essere imposta mediante tecniche di sincronizzazione
    - Sincronizzazione = meccanismo utilizzato per imporre dei vincoli all'ordine di esecuzione delle unità di processamento (processi o thread)

Grafo di precedenza