Lezione 32

Valutazione delle prestazioni di calcolo

Proff. A. Borghese, F. Pedersini

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

Perché valutare le prestazioni?

- Perché ?
  - Misura quantitativa delle prestazioni
    - Misura di velocità / throughput
    - Fatturazione delle prestazioni
  - Fare scelte intelligenti
    - *Installare nuovo hardware o nuovo software?*
    - *Analisi costi/benefici relativi all'acquisto di nuovo hardware*
- La misura di prestazioni riguarda di norma il **tempo**
  - $\text{Prestazioni}_X > \text{Prestazioni}_Y \rightarrow T_X < T_Y \rightarrow 1/T_X > 1/T_Y$
- Misura relativa (percentuale):

$$\text{Prestazioni}_X = (1 + n/100) \cdot \text{Prestazioni}_Y$$

$$T_Y = (1 + n/100) \cdot T_X \rightarrow n = 100 \cdot (T_Y - T_X) / T_X$$

Criteri (metriche) di valutazione orientati all'utente

CRITERI di VALUTAZIONE:

- Velocità di esecuzione
- Quantità di informazione elaborata

- Il criterio di valutazione dipende dall'utilizzo del calcolatore:
  1) Utilizzo personale → **tempo di esecuzione**
  2) Utilizzo come server → **throughput**
     - THROUGHPUT: Ammontare di lavori svolti in un dato tempo (accessi a banche dati, programmi, transazioni commerciali…)

- In casi molto particolari, si utilizzano anche altri criteri:
  - Efficienza energetica: Joule / operazione
  - Ingombro (form factor): op./sec / m^3

Criteri di valutazione orientati alla macchina

- **Tempo di risposta**: tempo totale per completare un lavoro
  - includendo accessi a disco, accessi a memoria, attività di I/O…
- **Tempo di CPU**: rappresenta il tempo speso dalla CPU per eseguire il programma dato
  - Non include i tempi di attesa per I/O o esecuzione di altri programmi
  - Include:
    - ***tempo di CPU utente*** (tempo speso dalla CPU per eseguire le linee di codice che stanno nel nostro programma)
    - ***tempo di CPU di sistema*** (speso dal sistema operativo per eseguire i compiti richiesti dal programma)

- Comando `time` (UNIX):
  - Produce tutti i tempi relativi al lavoro indicato
  - Sintassi: `time <job>` `90.7u 12.9s 2:39 65%`

Unità di misura di prestazioni: **CPI, T_{CPU}**

- Tempo di CPU (**T_{CPU}**):

$$T_{CPU} = N\_cicli\_clock * T_{CLOCK} = N\_cicli\_clock / f_{CLOCK}$$

- **CPI**: *Clock Per Instruction*
  - È il numero di cicli di clock / istruzione:

$$CPI = N\_cicli\_clock\_programma / N\_istruzioni$$

  - T_{CPU} e CPI sono quindi legate dalla seguente espressione:

$$T_{CPU} = CPI * N\_Istruzioni * T_{clock}$$

- Esempio:
  - *Calcolare CPI di un programma di 400.000 istruzioni, che necessita per la sua esecuzione (T_{CPU}) 1,2 sec, su un elaboratore con f_{CLOCK} = 1 MHz*
  - Per l'esecuzione del programma, occorrono:
  - N_cicli_clock = 10^6 cicli/sec × 1,2 sec = $1{,}2 \cdot 10^6$ cicli
  - CPI = N_cicli / N_istruzioni = $1{,}2 \cdot 10^6$ cicli / $4 \cdot 10^5$ istr. = 3 cicli / istruzione
  - Sulle macchine di oggi il CPI è inferiore a 1

Misura delle prestazioni : **$T_{medio/istruzione}$**

- Tempo esecuzione singola istruzione, ma…
  - in genere, istruzioni di tipo diverso richiedono quantità diverse di tempo
    - la moltiplicazione richiede più tempo dell'addizione
    - l'accesso alla memoria richiede più tempo dell'accesso ai registri

- **$T_{medio/istruzione}$**: tempo medio (pesato) di esecuzione di un set di istruzioni:

$$T_{medio/istruzione} = \frac{T_{tot}}{\#\,Istruzioni} = \frac{\sum_i t_i}{\#\,Istruzioni} = \frac{\sum_{i=0}^{S} l_i t_i}{\sum_{i=0}^{S} l_i}$$

l_i – Numero di volte che l'istruzione ***i*** viene eseguita nel programma

Misura delle prestazioni: CPI_{medio}, t_{medio}

CPI_i numero di cicli di clock per istruzioni di tipo *i*

l_i Numero di volte che l'istruzione *i* viene eseguita nel programma.

f_i Frequenza con cui l'istruzione *i* viene eseguita nel programma.

$$t_{medio} = \frac{\sum_{i=0}^{S} l_i t_i}{\sum_{i=0}^{S} l_i} \qquad CPI_{medio} = \frac{\sum_{i=1}^{n} (CPI_i * l_i)}{\sum_{i=1}^{n} l_i} = \sum_{i=1}^{n} (CPI_i * f_i)$$

($\sum_{i=1}^{n} l_i$ numero totale di istruzioni del programma)

$T_{CPU} = t_{medio} * N\_istruzioni = CPI * N\_Istruzioni * T_{clock}$

$$T_{CPU} = \sum_{i=1}^{n} (CPI_i * l_i) * T_{clock}$$

Esempio: calcolo T_{MI}

- Si consideri un calcolatore in grado di eseguire le istruzioni riportate in tabella:
- Calcolare CPI e il tempo di CPU per eseguire un programma composto da 200 istruzioni supponendo di usare una frequenza di clock pari a 500 MHz.

	Frequenza	CPI
ALU	43%	1
Load	21%	4
Store	12%	4
Branch	12%	2
Jump	12%	2

- **CPI** = 0,43 * 1 + 0,21 * 4 + 0,12 * 4 + 0,12 * 2 + 0,12 * 2 = **2,23**
- T_{CPU} = 200 * 2,23 * 2 nsec = **892 nsec**

$t_{clock} * CPI = t_{medio}$

MIPS: Million Instructions Per Second

- **MIPS = (N_istruzioni / 10^6) / T_{CPU}**
- **MIPS = f_{CLOCK} [MHz] / CPI**

- **Problemi di misura con MIPS:**
  - dipende dall'insieme di istruzioni, quindi è difficile confrontare computer con diversi insiemi di istruzioni;
  - varia a seconda del programma considerato;
  - può variare in modo inversamente proporzionale alle prestazioni!

- **Esempio:** macchina con hardware opzionale per virgola mobile:
  - Le istruzioni in virgola mobile richiedono più cicli di clock che quelle su interi
  - Hardware dedicato per la virgola mobile (in luogo delle routine software) impiegano meno tempo pur avendo un MIPS più basso

Misure & Problemi

- **MIPS relativo = T_{CPU} / T_{CPU_REF} * $MIPS_{CPU\_REF}$**
  - La CPUref è VAX–11/780
  - Problema: evoluzione dei sistemi.

- **MFLOPS:** per i supercomputer
  - Problema: misure di picco.
  - MIPS di picco e sostenuti. Problema: poco significative.

- **BENCHMARKS = Programmi per valutare le prestazioni**
  - Es: Whetstone, 1976; Drystone, 1984.
  - Kernel benchmark. Loop Livermore, Linpack, 1980. Problema: polarizzazione del risultato.
  - Benchmark con programmi piccoli (10-100 linee, 1980).
    - Mal si adattano alle strutture gerarchiche di memoria.

Indici **SPEC ('89, '92, '95)**

- METODO di MISURA: Insieme di programmi test
  - Misura in più condizioni diverse
    (singolo / multiplo processore / time sharing...)
- Benchmark specifici per valutare S.O. e I/O.
  - **SPEC '95 → SPECint, SPECfp**
  - Base: Sun SPARCstation 10/40
- Benchmark particolari:
  - **SDM** (Systems Development Multitasking).
  - **SFS** (System-level File Server).
  - **SPEChpc96** Elaborazioni scientifiche ad alto livello

Esempio **becnhmark SPEC**

SPEC '95 – Elaborazione intera:

1)	`Go`	Intelligenza artificiale
2)	`m88ksim`	Simulatore chip Motorola 88K
3)	`gcc`	Compilatore Gnu C che genera codice SPARC.
4)	`compress`	Compressione e decompressione di un file in memoria.
5)	`li`	Interprete LISP
6)	`ijpeg`	Compressione e decompressione immagini
7)	`perl`	Manipolazione di stringhe e numeri primi (in PERL)
8)	`vortex`	Gestione di una base di dati.

SPEC '95 – Elaborazione in virgola mobile

1)	`Tomcatv`	Programma per generazione di griglie
2)	`Swim`	Modello per acqua poco profonda con griglia 513 x 513
3)	`Su2cor`	Fisica quantistica: simulazione MonteCarlo
4)	`Hydro2D`	Simulazione sistemi fluidodinamici con equazioni di Navier-Stokes
5)	`Mgrid`	Risolutore multi-griglia in campo di potenziale 3D
6)	`Applu`	Equazioni alle differenze parziali paraboliche/ellittiche
7)	`Turb3D`	Simulazione di turbolenza isotropica ed omogenea in un cubo
8)	`Apsi`	Simulatore meteorologico, per il calcolo della diffusione di agenti inquinanti
9)	`Fpppp`	Chimica quantistica
10)	`Wave5`	Fisica dei plasmi: simulazione di particelle in campi elettromagnetici

SPEC 2000: programmi simili, di utilizzo comune (**`gzip`**, compilaz. C, ...)

Sommario

- Valutazione quantitativa delle prestazioni:
  Legge di Amdahl

Valutazione delle prestazioni, coerenza

- *Considero 2 macchine con prestazioni differenti:*

	Calcolatore I	*Calcolatore II*
Durata istruzione A	*1*	*10*
Durata istruzione B	*1000*	*100*
Durata istruzione C	*10*	*100*

- *Qual è più veloce?*

- Dipende dalla frequenza delle diverse istruzioni.
- Tempo **medio** di esecuzione di un'istruzione: $T = 1/n \sum_{i=0}^{n} n_i t_i$

Programma 1: 1000 istruzioni A, 1 istruzione B, 10 istruzioni C.

$T_I = 1/1011 * (1000*1 + 1*1000 + 10*10) = 2100/1011 \approx \mathbf{2}$

$T_{II} = 1/1011 * (1000*10 + 1*100 + 10*100) = 11100/1011 \approx \mathbf{100}$

Programma 2: 100 istruzioni A, 10 istruzioni B, 10 istruzioni C.

$T_I = 1/120 * (100*1 + 10*1000 + 10*10) = 10200/120 \approx \mathbf{100}$

$T_{II} = 1/120 * (100*10 + 10*100 + 10*100) = 3000/120 \approx \mathbf{30}$

Come rendere più veloci i calcolatori

- PRINCIPIO: rendere veloce il caso più frequente!
  - Si deve favorire il caso più frequente a discapito del più raro.

Definizioni:

- Frazione$_{migliorato}$: f_m ($0 \le f_m \le 1$):
  - la frazione del tempo di calcolo della macchina originale che può essere modificato per avvantaggiarsi dei miglioramenti.
- Speedup$_{migliorato}$: s_m ($s_m \ge 1$):
  - il fattore di aumento di velocità dovuto al miglioramento, rispetto alla velocità originale.

Speed-up - esempio

Consideriamo un calcolatore (Comp1) ...
... ed un secondo calcolatore (Comp2) in cui la **ALU è stata velocizzata (2x)**
Consideriamo un'applicazione che prevede **90%** di istruzioni in aritmetica intera.
Di quanto è lo **speed-up globale di Comp2 rispetto a Comp1**?

Speed-up attività modificata: $s_M = 2$
Frazione di tempo attività modificata: $f_M = 0.9$ (90%)

T Computer 1: $T_1 = T_{ALU} + T_R$; $T_{ALU} = 0.9\ T_1$, $T_R = 0.1\ T_1$
T Computer 2: $T_2 = T_{ALU}/2 + T_R = 0.45\ T_1 + 0.1\ T_1 = 0.55\ T_1$
Speed-up globale: $S = T_1/T_2 = 1 / 0.55 = 1.82$

In generale:
T Computer 1: $T_1 = T_{ALU} + T_R$; $T_{ALU} = f_M \cdot T_1$, $T_R = (1-f_M) \cdot T_1$
T Computer 2: $T_2 = T_{ALU}/s_M + T_R = f_M \cdot T_1/s_M + (1-f_M) \cdot T_1 = T_1 [f_M/s_M + (1-f_M)]$

Speed-up: $S = T_1/T_2 = 1 / [f_M/s_M + (1-f_M)]$

Corollario della legge di Amdhal

- **Legge di Amdahl:**
  Il miglioramento delle prestazioni globali ottenuto con un miglioramento particolare (e.g. un'istruzione), dipende dalla frazione di tempo in cui il miglioramento viene eseguito.

$$Speedup_{globale} = 1 / [\, 1 - F_m + F_m/S_m \,) \,]$$

- Corollario:
  - *Se un miglioramento è utilizzabile solo per una frazione del tempo di esecuzione complessivo (F_m), allora non è possibile accelerare l'esecuzione più del **reciproco di** ($1 - F_m$):*

$$Speedup_{globale} < 1/(1 - F_m)$$

In generale, per N fattori di speed – up :

$$S_{globale} = \frac{1}{1 - \sum_{i=1}^{N} f_{M,i} + \sum_{i=1}^{N} \frac{f_{M,i}}{s_{M,i}}}$$

Dimostrazione

$$T_{new} = T_{nm} + T_m = T_{old} * (1 - f_m) + (T_{old} / s_m) * f_m$$

0.1 — 0.9 / 2

$$T_{new} = T_{old} * (1 - f_m + f_m / s_m) = T_{old} * [1 - f_m * (1 - 1/s_m)]$$

Istruzioni non accelerate — Istruzioni accelerate

$$Speedup_{globale} = T_{old}/T_{new} = T_{old} / T_{old} * [1 - f_m * (1 - 1 / s_m)] =$$

$$1/ [1 - f_m + f_m / s_m)] < 1 / [1 - f_m] \quad \text{c.v.d.} \quad (s_m \to \infty)$$

Istruzioni non accelerate

Corollario:
Se il tempo di esecuzione delle istruzioni accelerate aumentasse all'infinito, il tempo di esecuzione T_{NEW} coinciderebbe con il tempo di esecuzione delle istruzioni non accelerate: $T_{OLD}(1-f_m)$

Quindi: $S_{max} = T_{OLD} / T_{NEW} = 1 / (1-f_m)$

Esempio 2

- **Esempio:**
  - Si consideri un miglioramento che consente un funzionamento 10 volte più veloce rispetto alla macchina originaria, ma che sia utilizzabile solo per il 40% del tempo.
  - Qual è il guadagno complessivo che si ottiene incorporando detto miglioramento?

- $Speedup_{globale} = 1 / [1 - F_m + F_m/S_m)]$
  - $\mathbf{Frazione_{migliorato} = F_m = 0.4}$
  - $\mathbf{Speedup_{migliorato} = S_m = 10}$
- $\mathbf{Speedup_{globale} = 1.56}$

Speed-up dovuto a cache memory

- Supponiamo che una cache sia **5** volte più veloce della memoria principale e che possa venire usata per il **90%** del tempo
- Qual'è il guadagno in velocità dovuto all'uso della cache?

- $Speedup_{globale} = 1/ [1 - F_{tempo\_cache} + F_{tempo\_cache}/ S_{cache})] =$
  $= 1 / (1 - 0.9 + 0.9/5) \approx 3.6$

- Velocità **3.6** volte superiore usando la cache

Esempio 3

- Supponiamo di potere aumentare la velocità della CPU della nostra macchina di un fattore 5 (senza influenzare le prestazioni di I/O) con un costo 5 volte superiore
- Assumiamo inoltre che la CPU sia utilizzata per il 50% del tempo ed il rimanente sia destinato ad attesa per operazioni di I/O.
- Se la CPU è un terzo del costo totale del computer è un buon investimento da un punto di vista costo/prestazioni, aumentare di un fattore 5 la velocità della CPU?

- $Speedup_{globale} = 1 / ((1–0.5) + 0.5/5) = 1.67$
- Incremento di costo = 2/3 + 1/3*5 = 7/3 = 2.33
- L'incremento di costo è quindi "più grande" del miglioramento di prestazioni
  - la modifica ***non*** migliora il rapporto costo/prestazioni

Esempio – speedup dovuto a vettorializzazione

- Si deve valutare un miglioramento di una macchina per l'aggiunta di una modalità vettoriale.
  La computazione vettoriale è 20 volte più veloce di quella normale.
  - La *percentuale di vettorizzazione* è la porzione del tempo che può essere spesa usando la modalità vettoriale.

1. Disegnare un grafico che riporti lo speedup in funzione della percentuale della computazione effettuata in modo vettoriale
   - Quale percentuale di vettorizzazione è necessaria per uno speedup di 2?
   - Quale per raggiungere la metà dello speedup massimo?

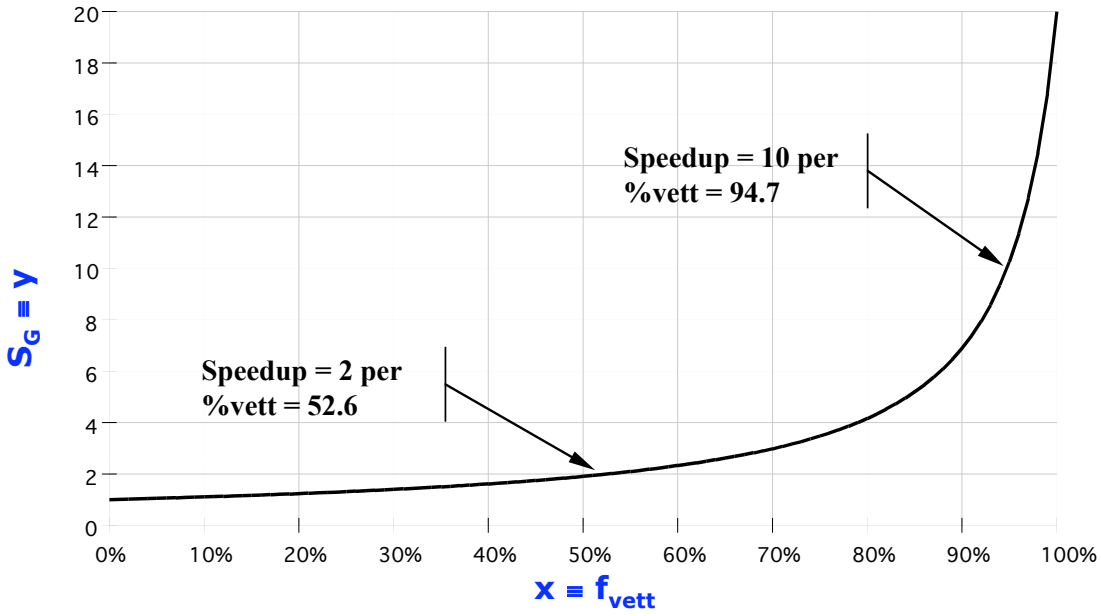
- La percentuale di vettorizzazione misurata è del 70 %
- I progettisti hardware affermano di potere raddoppiare la velocità della parte vettoriale se vengono effettuati significativi investimenti.
- Il gruppo che si occupa dei compilatori può incrementare la percentuale d'uso della modalità vettoriale

2. Quale incremento della percentuale di vettorizzazione sarebbe necessario per ottenere lo stesso guadagno di prestazioni
3. Quale investimento raccomandereste?

Esempio: curva di speed-up

- Speed-up come funzione della frazione di vettorizzazione:

$$S_g \equiv S_G(f_V) = y(x) = 1/ [1- x + x/20] = 20/(20-19x)$$



Esempio: speed-up dovuto a HW

- L'incremento di velocità con f_{VETT} = ***70 %*** ...
  - Speedup_G = 1/ [1 – 0.7 + 0.7 / 20)] =
    = 1/(1-0.7*19/20) = **2,9851**
- HW: raddoppiando la velocità dell'unità vettoriale:
  - Speedup_{HW} = 1/ [1 – 0.7 + 0.7 / 40)] =
    = 1/(1 – 0.7*39/40) = **3,1496**
- SW: aumentando la percentuale di vettorizzazione, f_{VETT} ...
  - Speedup_{SW} = 3,1496 = 1/ [1 – x + x / 20)]
    ➔ x = f_{VETT} = **71,84%**
- Soluzione SW ragionevolmente più vantaggiosa!