Valutazione delle prestazioni
Parte V

Alberto Borghese
Università degli Studi di Milano
Dipartimento di Scienze dell'Informazione
email: borghese@dsi.unimi.it

Perché valutare le prestazioni?

- Misura/Valutazione quantitativa delle prestazioni (velocità….).
- Fare scelte intelligenti (e.g. installare nuovo hardware o nuovo sw).
- Orientarsi nell'acquisto di nuovo hw.
- Fatturazione delle prestazioni.

La misura delle prestazioni è il tempo.

$Prestazioni_X > prestazioni_Y \Rightarrow tempo_X < tempo_Y \Rightarrow 1/tempo_X > 1/tempo_Y$

$Prestazioni_X = (1+n/100)xPrestazioni_Y.$
$Tempo_Y=(1+n/100)xTempo_X \Rightarrow n=100x(Tempo_Y-Tempo_X)/Tempo_X$
ΔT

Le prestazioni migliorano perché:
- Incrementano le prestazioni.
- Diminuisce il tempo di esecuzione.

Criteri (metrica) di valutazione orientati all'utente

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…).

Domande:
Un processore più veloce cosa influenza?
Più processori dedicati, cosa modificano?

Criteri (metrica) di valutazione orientati alla macchina

Tempo di risposta rappresenta la latenza per il completamento di 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 il tempo di attesa per I/O o per l'esecuzione di altri programmi. Comprende il ***tempo utente di CPU*** (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: 90.7u 12.9s 2:39 65%

Unità di misura delle prestazioni (CPI)

Tempo di CPU =
Numero_cicli_clock * Durata_clock =
Numero_cicli_clock / Frequenza_clock.

Determinazione del numero di cicli di clock:

Cicli di clock per istruzione (CPI) =
Cicli_clock_CPU_programma / Numero_istruzioni

Quindi:

T_{CPU} = CPI * Numero_Istruzioni * T_{clock}

Misura delle prestazioni

Tempo esecuzione singola istruzione, ma:

In genere, istruzioni di tipo diverso richiedono quantità diverse di tempo. Esempi:
- la moltiplicazione richiede più tempo dell'addizione
- l'accesso alla memoria richiede più tempo dell'accesso ai registri.

Tempo esecuzione medio (pesato) di un mix di istruzioni:

$$t_{medio} = \frac{\sum_{i=0}^{S} l_i t_i}{\sum_{i=0}^{S} l_i}$$

Misura delle prestazioni mediante CPI

T_{CPU} = CPI * Numero_Istruzioni * T_{clock}

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

- CPI_i numero di cicli di clock per istruzioni di tipi 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.
  ($\sum_{i=1}^{n} l_i$ rappresenta il numero di istruzioni)

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

Esempio

Si consideri un calcolatore in grado di eseguire le istruzioni riportate in tabella:

	Frequenza	cicli di clock
ALU	43%	1
Load	21%	4
Store	12%	4
Branch	12%	2
Jump	12%	2

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.

CPI = 0,43 * 1 + 0,21 * 4 + 0,12 * 4 + 0,12 * 2 + 0,12 * 2= 2,23
$T_{CPU} = 200 * 2{,}23 * 2_{ns} = 892_{ns}$

MIPS = milioni di istruzioni per secondo

MIPS = numero_istruzioni / (tempo_esecuzione_istruzione * 10^6)
MIPS = frequenza_clock / (CPI * 10^6)

Problemi:
· 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 rispetto a quelle che lavorano con interi, quindi i programmi che usano l'hardware opzionale per la virgola mobile in luogo delle routine software per tali operazioni impiegano meno tempo ma hanno un MIPS più **basso**. L'implementazione software delle istruzioni in virgola mobile esegue semplici istruzioni, con il risultato di avere un elevato MIPS, ma ne esegue talmente tante da avere un più elevato tempo di esecuzione!!

Misure & Problemi

MIPS relativi = $tempo_{CPU}$ /$tempo_{CPU\_ref}$ * $MIPS_{CPU\_ref}$. La CPU_{ref} è VAX-11/780. Problema: evoluzione dei sistemi.

MFLOPS per i super computer. Problema: misure di picco.

MIPS di picco e sostenuti. Problema: poco significative.

Benchmarks = Programmi per valutare le prestazioni.

Benchmarks: Whetstone, 1976; Drystone, 1984.

Kernel benchmark. Loop Livermore, Linpack, 1980. Problema: polarizzaione del risultato.

Benchmark con programmi piccoli (10-100 linee, 1980). Problema: mal si adattano alle strutture gerarchiche di memoria.

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

Insieme di programmi test.
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.

Orientamento: Benchmark specifici.

Esempio becnhmark SPEC95

Elaborazione intera:

1) Go	Intelligenza artificiale
2) m88ksim	Simulatore chip Motorola 88K; esecuzione di un programma.
3) gcc	Compilatore Gnu C che genera codice SPARC.
4) compres	Compressione e decompressione di un file in memoria.
5) li	Interprete lisp
6) ijpeg	Compressione e decompressione di immagini grafiche.
7) perl	Manipolazione di stringhe e numeri primi nel linguaggio di programmazione dedicato Perl.
8) vortex	Programma di gestione di una base di dati.

Esempio becnhmark SPEC95

Elaborazione 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: simuilazione MonteCarlo.
4) Hydro2D Astrofisica: equazione idrodinamiche di Naiver 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 Risoluzione di problemi di temperatura, velocità del vento e diffusione di agenti inquinanti.
9) Fpppp Chimica quantistica.
10) Wave5 Fisica dei plasmi: simulazione di particelle elettromagnetiche.

Miglioramento delle prestazioni

- Riduzione del numero di cicli di clock.
- Diminuzione del periodo di clock (aumentare la frequenza).

Tempo esecuzione = Numero_Cicli_clock * Durata del clock

CPI rappresenta il tempo di esecuzione medio delle istruzioni.
Miglioramenti dell'architettura per ridurre il CPI.
Miglioramento del compilatore per ridurre il CPI.
Ridurre la durata del clock (aumentarne la frequenza).

Espressione dei risultati

Il tempo totale di esecuzione dipende da diverse caratteristiche: dischi, sottosistema di I/O, sottosistema grafico ….

Per questo motivo occorre menzionare la **configurazione** del sistema.

Valutazione delle prestazioni, coerenza

	Calcolatore A	Calcolatore B
Istruzione 1 (s)	1	10
Istruzione 2 (s)	1000	100
Istruzione 3 (s)	10	100

Qual è più veloce? Dipende dal peso dei programmi.
Media pesata: $T = 1/n \sum_{i=0}^{n} n_i t_i$

Programma 1: 1000 istruzioni A, 1 istruzione B, 10 istruzioni C.
T_A = 1/1001 * (1000*1 + 1*1000+ 10*10) = 2100/1011
T_B = 1/1001 * (1000*10 + 1*100 + 10*100) = 11100/1011

Programma 2: 100 istruzione A, 10 istruzioni B, 10 istruzioni C.
T_A = 1/120 * (100*1 + 10*1000+ 10*10) = 10200/120
T_B = 1/120 * (100*10 + 10*100 + 10*100) = 3000/120

Come rendere più veloci i calcolatori

Rendere veloce il caso più comune.
Si deve favorire il caso più frequente a discapito del più raro.
Il caso più frequente è spesso il più semplice e può essere quindi reso più veloce del caso infrequente.

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 era eseguito.

Esempio: Pentium e PentiumPro.

Speed-up

Il miglioramento globale proporzionale al miglioramento di una parte del sistema?

Speed up (accelerazione):

prestazioni_intero_lavoro_con_miglioramento / prestazioni_senza

Oppure

tempo_intero_lavoro_senza_miglioramento / tempo_con_miglioramento.

Speed-up - esempio

Consideriamo un calcolatore (CALC1) con ALU ed una FP_ALU. Consideriamo un secondo calcolatore (CALC2) in cui la ALU è stata velocizzata (2x).

Consideriamo un'applicazione che prevede un 90% di istruzioni in aritmetica intera. Di quanto è lo speed-up?

	ISTRUZIONI INTERE		ISTRUZIONI TOTALI	
Calcolatore	**T_ESEC**	**Speedup_m**	**T_ESEC**	**Speedup**
CALC1	90	1.0	100	1.0
CALC2	45	2.0	55	1.82

Speed-up = 100/55 = 1.818….

Corollario della legge di Amdhal

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 uno meno tale frazione: $Speedup_{globale} <= [1- 1/(1-F_m)]$.

Definizioni:

1. **$Frazione_{migliorato}$** ($F_m \leq 1$), ovvero la frazione del tempo di calcolo della macchina originale che può essere modificato per avvantaggiarsi dei miglioramenti. Nell'esempio precedente la frazione è 0.90.
2. **$Speedup_{migliorato}$** ($S_m \geq 1$), ovvero il miglioramento ottenuto dal modo di esecuzione più veloce. Nel precedente esempio questo valore viene fornito nella colonna chiamata Speedup_migliorato (pari a 2).

Dimostrazione

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

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

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

$1/ [1-F_m + F_m / S_m)] < 1/ [1-F_m]$ c.v.d. $(S_m \rightarrow \infty)$

Esempio precedente:

$T_{new} = 100 * (1 – 0,9 + 0,9/2) = 55$

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?

$$\text{Speedup}_{\text{globale}} = 1/ [1-F_m+ F_m / S_m)]$$

$\text{Frazione}_{\text{migliorato}} = 0.4$ **$\text{Speedup}_{\text{migliorato}} = 10$**
$\text{Speedup}_{\text{globale}} = 1.56$

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 cinque la velocità della CPU?

$\text{Speedup}_{\text{globale}} = 1.67$ Incremento di costo = 2.33

L'incremento di costo è quindi più grande del miglioramento di prestazioni: la modifica *non* migliora il rapporto costo/prestazioni.

Speed-up dovuto a cache memory

Supponiamo che una cache sia **5** volte più veloce della memoria principale ed inoltre che la cache possa venire usata per il **90%** del tempo.

Qual'è il guadagno in velocità dovuto all'uso della cache?

$$\text{Speedup}_{\text{globale}} = 1/ [1-F_{\text{tempo\_cache}} + F_{\text{tempo\_cache}} / S_{\text{cache}})] =$$

$$1 / (1-0.9+0.9/5) \approx 3.6$$

Otteniamo quindi una velocità **3.6** volte superiore usando la cache.

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.

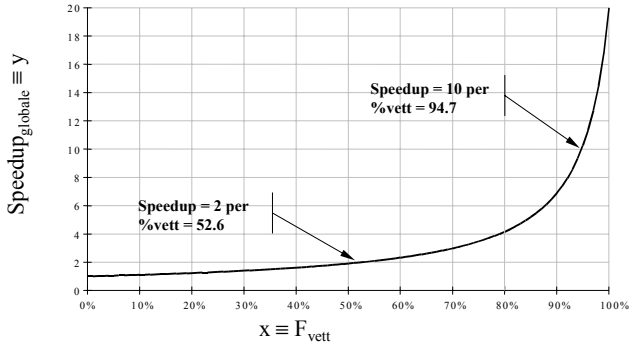
- Disegnare un grafico che riporti lo speedup come percentuale della computazione effettuata in modo vettoriale.
- Quale percentuale di vettorizzazione è necessaria per uno speedup di 2?
- Quale per raggiungere la metà dello speedup massimo?

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.

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

Curva di speed-up

$Speedup_{globale} \equiv y = 1/ [1- x + x / 20)] = 20/(20-19x)$ $\quad x \equiv F_{vett}$



Speed-up dovuto a HW

$Speedup_{original} = 1/ [1- 0.7 + 0.7 / 20)] = 1/(1-0.7*19/20) = 2,9851$

$Speedup_{HW} = 1/ [1- 0.7 + 0.7 / 40)] = 1/(1-0.7*39/40) = 3,1496$

$Speedup_{compiler} = 3,1496 = 1/ [1- x + x / 20)]$ ➔ $F_{vettoriale} = 71,84\%$

Principio di località

I programmi riutilizzano dati e istruzioni che hanno usato di recente.

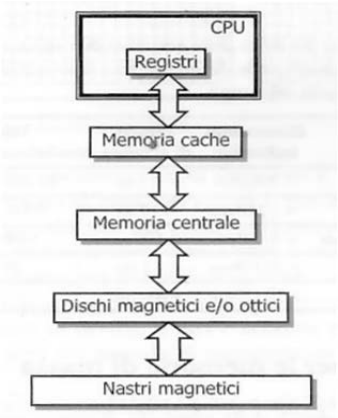
Regola pratica: un programma spende circa il **90%** del suo tempo di esecuzione per solo il **10%** del suo codice.

Basandosi sul passato recente del programma, è possibile predire con ragionevole accuratezza quali dati e istruzioni userà nel prossimo futuro.

località temporale elementi ai quali si è fatto riferimento di recente saranno utilizzati ancora nel prossimo futuro.

località spaziale elementi i cui indirizzi sono vicini, tendono ad essere referenziati in tempi molto ravvicinati.

Gerarchia di memorie



Gerarchia di memorie - caratteristiche

Livello	Dimensioni indicative	Tempo di Accesso	Velocità di Trasferimento (Mbyte/s)
Registri	< 1 Kbyte	< 0,5ns	20,000 – 100,000
Cache	< 1 Mbyte	< 10 ns	5,000 - 10,000
Memoria centrale	< 4 Gbyte	< 100ns	1,000 – 5,000
Dischi	> 50 Gbyte	< 10ms	20 - 40
Nastri	> 10 Gbyte	> 100ms	1

Gerarchie di memoria

HIT Successo nel tentativo di accesso ad un dato: è presente al livello superiore della gerarchia.

MISS Fallimento del tentativo di accesso al livello superiore della gerarchia => l'indirizzo deve essere cercato al livello inferiore.

HIT_RATE Percentuale dei tentativi di accesso ai livelli superiori della gerarchia che hanno avuto successo.
HIT_RATE=Numero_successi/Numero_accessi_memoria

MISS_RATE Percentuale dei tentativi di accesso ai livelli superiori della gerarchia che sono falliti
MISS_RATE=Numero_fall./Numero_accessi_memoria

HIT_RATE + MISS_RATE = 1

Valutazione prestazioni memoria

Obbiettivo principale della gerarchia di memoria è incrementare le prestazioni => diminuire la velocità di accesso sia in caso di HIT che di MISS.

Cosa succede in caso di MISS?

HIT_TIME Tempo di accesso al livello superiore (che comprende anche il tempo necessario per determinare se l'accesso ha avuto successo oppure fallisce.

MISS_PENALTY è composto da:
TEMPO DI ACCESSO per accedere alla prima parola del blocco dopo che è stato rilevato il fallimento.
TEMPO DI TRASFERIMENTO per trasferire le altre parole del blocco al livello superiore.

MISS_TIME = > HIT_TIME + MISS_PENALTY

Tempo medio di accesso alla memoria

TEMPO DI ACCESSO. E' legato al tempo di accesso del livello inferiore di memoria.
TEMPO DI TRASFERIMENTO. E' legato alla larghezza di banda del canale di comunicazione tra i due livelli di memoria (e.g. bus).

Il tempo medio di accesso alla memoria sarà:

$\mathbf{T_{medio}}$ = HIT_RATE * HIT_TIME + MISS_RATE * MISS_TIME =

HIT_TIME*HIT_RATE+MISS_RATE*(HIT_TIME+MISS_PENALTY) =

HIT_TIME*(HIT_RATE+MISS_RATE)+MISS_RATE*MISS_PENALTY =

HIT_TIME + MISS_RATE * MISS_PENALTY

Impatto di una memoria cache

Il tempo di CPU è composto dal tempo richiesto dalla CPU per eseguire il programma e dal tempo che la CPU trascorre in attesa di risposta dal sottosistema di memoria.

T_{CPU} = (#Cicli della CPU in esecuzione + #Cicli di stallo) * T_{Clock}

Ipotesi:
•Tutti gli stalli di memoria sono dovuti al fallimento di accesso alla cache.
•I cicli di clock utilizzati per un accesso alla cache riuscito (HIT) sono inclusi nei cicli di clock della CPU in esecuzione.

Impatto di una memoria cache

Cicli_clock_stallo = #Accessi_Memoria*MISS_RATE*MISS_PENALTY

$Tempo_{CPU}$_Programma = (#Cicli_clock + #Cicli_clock_stallo)*T_{clock} = #Istruzioni*CPI_{exec}*T_{clock} + #Cicli_clock_stallo*T_{clock}

CPI_{con_cache} = CPI_{exec} + #Cicli_clock_stallo / #Istruzioni = CPI_{exec}+(#Accessi_memoria/#Istruzioni)*MISS_RATE*MISS_PENALTY

Caso ideale: (100% HIT, 0%MISS): CPI_{con_cache} = CPI_{exec}

Caso senza cache: (100% MISS): CPI_{senza_cache}= CPI_{exec} + (#Accessi_memoria/#Istruzioni)* MISS_PENALTY

Esercizio su cache

Si consideri il VAX-11/780. La MISS_PENALTY è di 6 cicli di clock, mentre tutte le istruzioni impiegano 8.5 cicli di clock se si ignorano i MISS (stalli della memoria). Ipotizzando un MISS_RATE dell'11% e che vi siano in media 2 riferimenti alla memoria per ogni istruzione,

⇒ Qual è l'impatto sulle prestazioni quando viene inserita la cache reale rispetto ad una cache ideale?

⇒ Qual è l'impatto sulle prestazioni tra il caso di cache reale e senza inserimento della cache?

Soluzione esercizio su cache

Dati di ingresso: MISS_PENALTY=6 CPI_{exec}=8.5 MISS_RATE=0,11
#Accessi_memoria/#Istruzioni = 2

CPI_{con_cache} = 8,5 + 2 * 0,11 * 6 = 9,82
$CPI_{con\_cache\_ideale}$ = 8,5
CPI_{senza_cache}= 8,5 + 2*6 = 20,5

Perdita in prestazioni (speed-up): CPI_{con_cache} / $CPI_{con\_cache\_ideale}$ = 8,5 / 9,82 = 0,865

Guadagno in prestazioni (speed-up): CPI_{con_cache} / CPI_{senza_cache} = 20,5 / 9,82 = 2,087