Valutazione delle prestazioni

Salvatore Orlando

Valutazione delle prestazioni

- L'approccio RISC ha semplificato l'ISA, ma ha anche portato a
  - sviluppo di tecniche di ottimizzazione molto spinte
  - nuova sensibilità per analisi più quantitative delle prestazioni dei sistemi

- Studieremo le modalità per
  - misurare, descrivere e sintetizzare le prestazioni di un sistema hardware/software
  - ci concentreremo prima sulle prestazioni della CPU

Valutazione delle prestazioni

- Nel misurare le prestazioni, dovremo anche considerare il software
  - istruzioni che compongono i programmi
  - tipi di riferimenti alla memoria
  - operazioni di I/O

- Misurare oggettivamente le prestazioni è MOLTO DIFFICILE

- A cosa serve studiare le metriche per valutare le prestazioni, e il modo di misurarle?
  - per fare buone scelte di progetto, anche software
    - capire perché le prestazioni di un programma sono cattive, anche se l'algoritmo usato dovrebbe essere teoricamente ottimo
  - fare buone scelte nell'acquisto di nuovo hardware, imparando a leggere e comparare le prestazioni

Valutazione delle prestazioni

- **Questo studio dovrebbe rendere più semplice poter rispondere a domande del tipo:**

  Perché può succedere che un sistema sia migliore di un altro per il programma A, e risulti invece peggiore per il programma B ?

  I fattori che influenzano le prestazioni sono relative all'hardware o al software?
  Es.: è meglio istallare un nuovo OS, o comprare una nuova macchina?

  Come può un certo set di istruzioni (RISC vs CISC) influenzare le prestazioni di una CPU?

  Come può l'uso di un nuovo compilatore influenzare le prestazioni di una CPU rispetto ad un certo insiemi di programmi?

Il problema delle metriche

Aereo	Posti	Autonomia (km)	Velocità (km/h)	Portata (Pass. x km/h)
Boeing 777	*375*	*7450*	*980*	*367500*
Boeing 747	*470*	*6680*	*980*	*460600*
BAC/Sud Concorde	*132*	*6440*	*2170*	*286440*
Douglas DC-8-50	*146*	*14030*	*875*	*127750*

- **Metriche diverse**
  - **BAC/Sud Concorde è il più veloce (di quanto rispetto al DC-8?)**
  - **Boeing 747 è più capiente (di quanto rispetto al 777?)**
  - **Douglas DC-8-50 ha maggiore autonomia (di quanto rispetto al 747?)**

- **Nota la colonna Portata:**
  - **è una misura che tiene conto sia della Capienza e sia della Velocità**
  - **metrica utile usata se non siamo interessati a valutate la velocità di spostamento di un singolo passeggero, ma a valutare quanti passeggeri siamo in grado di spostare contemporaneamente a quella data velocità**

Misure di prestazioni di un computer

- **Tempo di esecuzione (latenza)**
  - **— Quanto impiega il mio job ad essere eseguito a sistema scarico ?**
  - **— Quanto devo attendere per una query di un database?**
- **Throughput (Banda di elaborazione)**
  - **— Quanti job possono essere eseguiti assieme su una macchina?**
  - **— Qual è il tempo di esecuzione medio?**
  - **— Quanto lavoro viene completato in un certo tempo?**

- **Quali delle due misure viene influenzata**
  - **se compriamo una nuova CPU?**
    - **Diminuiamo *tempo di esecuzione* dei job e aumentiamo *throughput***
  - **se aggiungiamo un'altra workstation ad un laboratorio ?**
    - **Aumentiamo il *throughput,* ma non il *tempo di esecuzione***
    - **Diminuiamo comunque *tempo di risposta* se ci sono molti job, e questi sono costretti a rimanere in coda ⇒ riduciamo le attese in coda**
      - **tempo di risposta = tempo attesa in coda + tempo di esecuzione**

Misure di prestazioni del computer

- Il termine Performance (Prestazioni) è usato
  - sia come *misura generica della velocità di un computer*
    - " …. per migliorare le prestazioni.. "
    - " … provoca l'aumento delle prestazioni.."
  - ma soprattutto come sinonimo di Throughput
- Rispetto al Tempo di esecuzione, possiamo essere più precisi:
  - Elapsed Time o Response Time (wall clock time)
    - tiene conto di ogni cosa *(accessi al disco (I/O), multiprogrammazione e attese nelle code (OS), ecc.),*
  - Tempo di CPU
    - vogliamo distinguere rispetto ai tempi spesi per I/O e multiprogrammazione
    - es.: comando UNIX *time* (time <prog>)

- Il nostro primo obiettivo sarà quello di analizzare soltanto il tempo di CPU impiegato per l'esecuzione dei programmi utente (user CPU time)
  - ovvero, senza considerare i codici dell'O.S., l'I/O, o altro
  - Execution time = user CPU time

Performance, Execution time, Speedup

- **Misuriamo il tempo di esecuzione (user CPU time) di un programma su una macchina X:**

  Execution time_X

- **Definiamo la Performance (throughput) come:**

  $\text{Performance}_X = 1 / \text{Execution time}_X$

- **Se la macchina X è più veloce di Y a eseguire il programma**
  - **Speedup =** $\text{Execution time}_Y / \text{Execution time}_X =$ $\text{Performance}_X / \text{Performance}_Y = n$
  - **"X è n volte più veloce di Y"**

- **Problema:**
  - **la macchina A esegue un programma in 20 s**
  - **la macchina B lo esegue in 25 s**
  - **di quanto A è più veloce di B?**

Speedup = Exec Time maggiore / Exec time minore = 25/20 ≅ 1.2

Cicli di clock

- **Legame tra tempo di esecuzione e numero di cicli di clock per eseguire un programma**

  CPU time (in sec.) = no. cicli × T
  (durata/periodo del ciclo di clock in sec.)

- **Clock rate (Frequenza) = cicli al secondo (1 Hz. = 1 cycle/sec)**
  - **Freq = 1/T (dove T è il periodo del ciclo di clock)**
  - **T = 1 / Freq *(sec.)***
- **Un clock a 200 Mhz. (200×10^6 Hz) ha un periodo T di clock uguale a**

$$\frac{1}{200 \times 10^6} \times 10^9 = 5 \text{ nanoseconds}$$

- **Conoscendo no. di cicli per l'esecuz. di un programma, e frequenza del clock:**

  CPU time (in sec.) = no. cicli / Frequenza in Hz

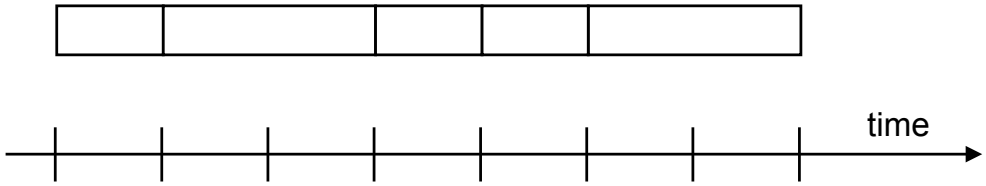
Miglioramento prestazioni

CPU time (in sec.) = no. cicli × Periodo clock in sec.

CPU time (in sec.) = no. cicli / Frequenza in Hz

- **Per migliorare le prestazioni possiamo**
  diminuire il # di cicli per eseguire un programma, oppure
  diminuire il ciclo di clock (cycle time) oppure, detto in modo diverso,
  aumentare la frequenza (clock rate).

Numero di cicli differenti per istruzioni differenti



- **Moltiplicazioni impiegano più tempo delle addizioni**
- **Operazioni FP impiegano più tempo delle operazioni su interi**
- **L'accesso alla memoria costa di più che accedere i registri**

Punto importante*: se cambiamo il ciclo di clock (frequenza), spesso otteniamo, come effetto collaterale, la modifica del numero di cicli necessari per eseguire le varie istruzioni, dovuto agli accessi alla memoria ….. (approfondimenti nel seguito)*

Esempio

- **Per un dato programma da eseguire sul computer A conosciamo**
  il tempo di CPU sul computer A: $T_A = 10$ s
  la frequenza di clock del computer A: $Freq_A = 400$ MHz

- **Vogliamo costruire un computer B dove**
  il tempo di CPU è : $T_B = 6$ s

- **Il progettista può *aumentare* la *frequenza di B*, ma questo provoca l'*incremento* dei *numero di cicli* per l'esecuzione del programma (1.2 in più dei cicli necessari sulla macchina A). Di quanto dovremmo aumentare $Freq_B$ per garantire $T_B = 6$ s ?"**

- T_A = # cicli A / $Freq_A$ $\Rightarrow$ no. cicli A = $T_A * Freq_A = 10 * 400 * 10^6$

- $T_B = 1.2 *$ # cicli A / $Freq_B$

sostituendo: $6 = 1.2 * 4000 * 10^6 / Freq_B$

$\Rightarrow$ $Freq_B = 1.2 * 4000 * 10^6 / 6$ Hz = $0.2 * 4000$ MHz = **800 Mhz**

Cicli di clock e prestazioni

- **Per l'esecuzione di un programma (CPU time) necessari**

  un certo # di istr. Macchina: **IC (Instr. Count)**

  un certo # di cicli: **no. cicli**

  un certo numero di secondi: **T_{exe}**

$$T_{exe} = \text{no. cicli} / \text{Freq} = \text{no. cicli} * T$$

$$\text{no. cicli} = IC * CPI$$

- **CPI (cicli per istruzione) = no. cicli / IC**
  - ***CPI dipende dal mix di istruzioni***
  - ***Un'applicazione FP-intensive (istruzioni complesse) potrebbe avere un CPI più alto della media !!***
- **MIPS (milioni di istruzioni per secondo) = $IC / (T_{exe} * 10^6)$**
  - ***i MIPS potrebbero essere più alti per programmi che usano istruzioni più semplici !!***
  - ***È una misura di throughput***

Tempo di esecuzione e altre misure

- **Siamo interessati al tempo di esecuzione (user CPU time) !!**

- **Le altre misure, prese singolarmente, potrebbero portarci a conclusioni errate nel valutare le prestazioni di un programma**
  - ***no. cicli* per eseguire un programma**
  - ***IC* = no. di istruzioni in un programma**
  - ***FREQ* = no. di cicli per secondo**
  - ***CPI* = no. medio di cicli per istruzione**
  - ***no. medio di istruzioni eseguite per secondo* (es. MIPS)**

- **Esempio**
  - **un programma viene *compilato* per la macchina A e viene eseguito. Viene quindi ricavata la misura di prestazioni in MIPS**
  - **lo stesso programma viene *compilato* per la macchina B, e i MIPS calcolati sono maggiori di quelli della macchina A**
  - **possiamo concludere che la macchina B è più veloce**

Esempio relativo al CPI

- Le macchine A e B implementano la stessa instruction set architecture (ISA).
  - Macchina A ha un ciclo di clock di 10 ns.
  - Macchina B ha un ciclo di clock di 20 ns.
- Per alcuni programmi:
  - Macchina A ha CPI = 2.0
  - Macchina B ha CPI = 1.2
- Quale macchina risulta più veloce per questo programma, e di quanto?
- Prima di risolvere il problema, rispondere al seguente quesito:
  - se due macchine hanno lo stesso ISA, quale delle quantità seguenti sarà sempre identico ?

    frequenza, CPI, tempo di esecuzione, IC, MIPS
- Soluzione problema:

  T_A = IC * CPI_A * periodo di $clock_A$ = IC * 2 * 10 ns.

  T_B = IC * CPI_b * periodo di $clock_B$ = IC * 1.2 * 20 ns.

  Devo confrontare T_A con T_B:

  IC * 2 * 10 con IC * 1.2 * 20, da cui $T_B > T_A$

  Speedup = T_B / T_A = IC * 1.2 * 20 ns. / IC * 2 * 10 ns. = 2.4 / 2 = 1.2

Esempio relativo all'IC

- Per una certa implementazione di un'ISA, abbiamo 3 classi di istruzioni:
  - Classe A: 1 ciclo di clock (CPI della classe A)
  - Classe B: 2 cicli di clock (CPI della classe B)
  - Classe C: 3 cicli di clock (CPI della classe C)
- Progettando un compilatore, nel tradurre una porzione di codice dobbiamo decidere tra due sequenze di codice macchina:
  - 1ª sequenza: 2 di A, 1 di B, e 2 di C (IC = 5)
  - 2ª sequenza: 4 di A, 1 di B, e 1 di C (IC = 6)
- Quale sequenza è più veloce, e di quanto? Qual è il CPI medio per ciascuna sequenza?
  - no. cicli$_1$ = 2*1+1*2+2*3=10 no. cicli$_2$ = 4*1+1*2+1*3=9
    poiché Freq è la stessa, la 2ª sequenza è più veloce:
    Speedup = 10/9 ≅ 1,1
  - per calcolare il CPI medio, dobbiamo guardare al mix di istruzioni…
    - CPI$_1$ = no. cicli$_1$ / IC$_1$ = 10 / 5 = 2
    - CPI$_2$ = no. cicli$_2$ / IC$_2$ = 9 / 6 = 1.5

Esempio relativo ai MIPS

- Due compilatori differenti sono testati per una macchina a 100 MHz con tre classi di istruzioni differenti
  - Classe A, B, e C, che richiedono 1, 2, e 3 cicli (rispettivamente).
- Il 1º compilatore genera un codice che usa
  - 5 M di istruzioni di Classe A, 1 M di Classe B, e 1 M di Classe C
- Il 2º compilatore genera un codice che usa
  - 10 M di istruzioni di Classe A, 1 M di Classe B, e 1 M di Classe C

- Quale sequenza sarà più veloce rispetto al tempo di esecuzione?
  - # $cicli_1$ = 5 M + 2 * 1 M + 3 * 1 M = 10 M
    T_1 = # $cicli_1$ / Freq = 10 M / 100 M = 0.1 s
  - # $cicli_2$ = 10 M + 2 * 1 M + 3 * 1 M = 15 M
    T_2 = # $cicli_2$ / Freq = 15 M / 100 M = 0.15 s
  - Speedup = T_2 / T_1 = 1.5 ⇒ Il 1º è l' 1.5 più veloce del 2º
- Quale sequenza sarà più veloce rispetto ai MIPS?
  - $MIPS_1 = IC_1 / T_1$ = 7M / 0.1 = 70 MIPS
  - $MIPS_2 = IC_2 / T_2$ = 12 M / 0.15 = 80 MIPS ⇒ Il 2º sembra più veloce

Benchmarks

- Per misurare le prestazioni, è meglio eseguire applicazioni reali
  - programmi che rappresentano un ***workload tipico***
  - spesso si usano programmi che rappresentano classi di applicazioni
    - compilatori/editori, applicazioni scientifiche, grafica, ecc.
  - i programmi considerati sono in formato sorgente, da compilare per le varie piattaforme
    - si considerano implicitamente anche la bontà dei compilatori
- Piccoli benchmark (software kernel)
  - sono comodi per i progettisti
  - possono portare ad abusi
- SPEC (System Performance Evaluation Cooperative)
  - insieme standard di programmi reali con relativi input
  - stabiliti sulla base di accordo tra aziende
  - ci possono essere ancora abusi
  - gli SPEC sono considerati un indicatore significativo delle prestazioni di un *hw* (e della tecnologia di compilazione)
  - Oltre alla CPU SPEC esistono altri benchmark SPEC …

SPEC ‘95

	Benchmark	Description
interi	go	Artificial intelligence; plays the game of Go
	m88ksim	Motorola 88k chip simulator; runs test program
	gcc	The Gnu C compiler generating SPARC code
	compress	Compresses and decompresses file in memory
	li	Lisp interpreter
	ijpeg	Graphic compression and decompression
	perl	Manipulates strings and prime numbers in the special-purpose programming language Perl
	vortex	A database program
virgola mobile	tomcatv	A mesh generation program
	swim	Shallow water model with 513 x 513 grid
	su2cor	quantum physics; Monte Carlo simulation
	hydro2d	Astrophysics; Hydrodynamic Naiver Stokes equations
	mgrid	Multigrid solver in 3-D potential field
	applu	Parabolic/elliptic partial differential equations
	trub3d	Simulates isotropic, homogeneous turbulence in a cube
	apsi	Solves problems regarding temperature, wind velocity, and distribution of pollutant
	fpppp	Quantum chemistry
	wave5	Plasma physics; electromagnetic particle simulation

SPEC 2000 - CINT

gzip	C	Compression
vpr	C	FPGA Circuit Placement and Routing
gcc	C	C Programming Language Compiler
mcf	C	Combinatorial Optimization
crafty	C	Game Playing: Chess
parser	C	Word Processing
eon	C++	Computer Visualization
perlbmk	C	PERL Programming Language
gap	C	Group Theory, Interpreter
vortex	C	Object-oriented Database
bzip2	C	Compression
twolf	C	Place and Route Simulator

SPEC 2000 - CFP

wupwise	Fortran 77	Physics / Quantum Chromodynamics
swim	Fortran 77	Shallow Water Modeling
mgrid	Fortran 77	Multi-grid Solver: 3D Potential Field
applu	Fortran 77	Parabolic / Elliptic Partial Diff. Equat.
mesa	C	3-D Graphics Library
galgel	Fortran 90	Computational Fluid Dynamics
art	C	Image Recognition / Neural Networks
equake	C	Seismic Wave Propagation Simulation
facerec	Fortran 90	Image Processing: Face Recognition
ammp	C	Computational Chemistry
lucas	Fortran 90	Number Theory / Primality Testing
fma3d	Fortran 90	Finite-element Crash Simulation
sixtrack	Fortran 77	High Energy Nuclear Physics Accelerator Design
apsi	Fortran 77	Meteorology: Pollutant Distribution

CINT2006 (Integer Component of SPEC CPU2006)

• perlbench	C	Programming Language Derived from Perl V5.8.7.
• bzip2	C	Compression
• gcc	C	C Compiler
• mcf	C	Combinatorial Optimization
• gobmk	C	Artificial Intelligence
• hmmer	C	Search Gene Sequence
• sjeng	C	Artificial Intelligence: chess
• libquantum	C	Physics / Quantum Computing
• h264ref	C	Video Compression
• omnetpp	C++	Discrete Event Simulation
• astar	C++	Path-finding Algorithms (2D maps)
• xalancbmk	C++	XML Processing

CFP2006 (Floating Point Component of SPEC CPU2006)

• bwaves	Fortran	Fluid Dynamics
• gamess	Fortran	Quantum Chemistry
• milc	C	Physics / Quantum Chromodynamics
• zeusmp	Fortran	Physics / Computationalfluid dynamics
• gromacs	C,Fortran	Biochemistry / Molecular Dynamics
• cactusADM	C,Fortran	Physics / General Relativity
• leslie3d	Fortran	Computational Fluid Dynamics (CFD)
• namd	C++	Biology / Molecular Dynamics
• dealII	C++	Finite Element Analysis
• soplex	C++	Linear Programming, Optimization
• povray	C++	Image Ray-tracing
• calculix	C,Fortran	Structural Mechanics
• GemsFDTD	Fortran	Computational Electromagnetics
• tonto	Fortran	Quantum Chemistry
• lbm	C	Fluid Dynamics
• wrf	C,Fortran	Weather modeling
• sphinx3	C	Speech recognition

Misure sintetiche per insiemi di benchmark

- **Nel confrontare piattaforme che eseguono un mix di programmi, può succedere che**
  - **una piattaforma risulti migliore di un'altra rispetto all'esecuzione di un certo programma, e peggiore rispetto ad un altro programma.**
- **Abbiamo bisogno di un indice complessivo rispetto alle prestazioni misurate per un certo mix di programmi**
  - **Media aritmetica (pesata) dei tempi di esecuzione dei vari programmi su una certa macchina**

$$\frac{1}{n} \cdot \sum_{i=1}^{n} T_i$$

  - **Media geometrica (pesata) ancora dei tempi di esecuzione**

$$\sqrt[n]{\prod_{i=1}^{n} T_i}$$

SPECint e SPECfp

- SPECint e SPECfp fanno rispettivamente riferimento all'esecuzione del mix di programmi (interi o a virgola mobile) per l'elaborazione intera o FP
- Gli indici SPEC danno un'indicazione complessiva del comportamento del mix i programmi su una data piattaforma
  - indici ottenuti come media geometrica
  - invece di usare direttamente i tempi di esecuzione, i valori su cui si fa la media sono normalizzati rispetto ai tempi di esecuzione su una macchina di riferimento (vecchia). I valori di T_{refi} sono relativi a una
    - Sun Ultra5_10 - 300MHz (SPEC 2000)
    - Sun SPARCstation 10/40 (SPEC 95)
    - precedentemente si usava il Dec VAX-11/780

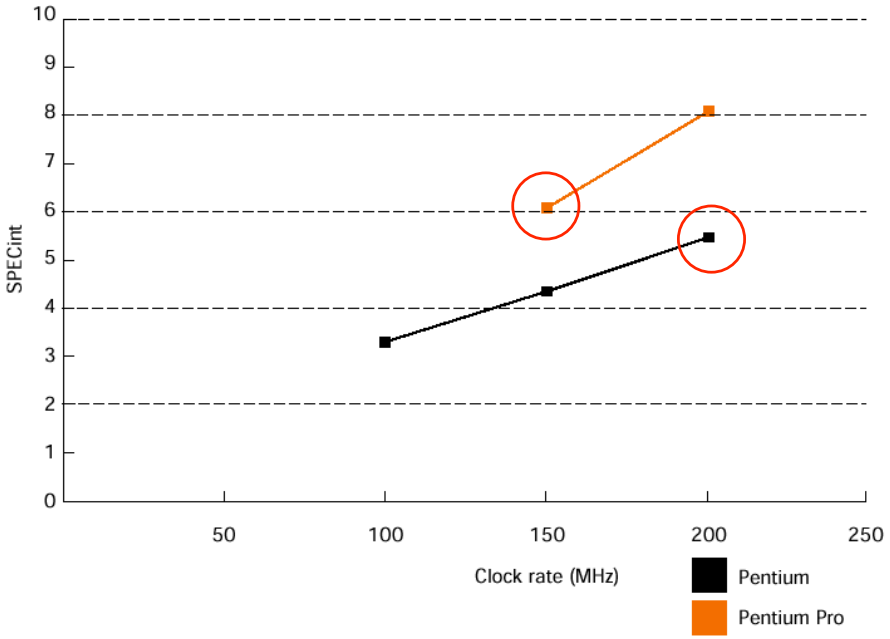
$$\sqrt[n]{\prod_{i=1}^{n} T_{rif_i} / T_i}$$

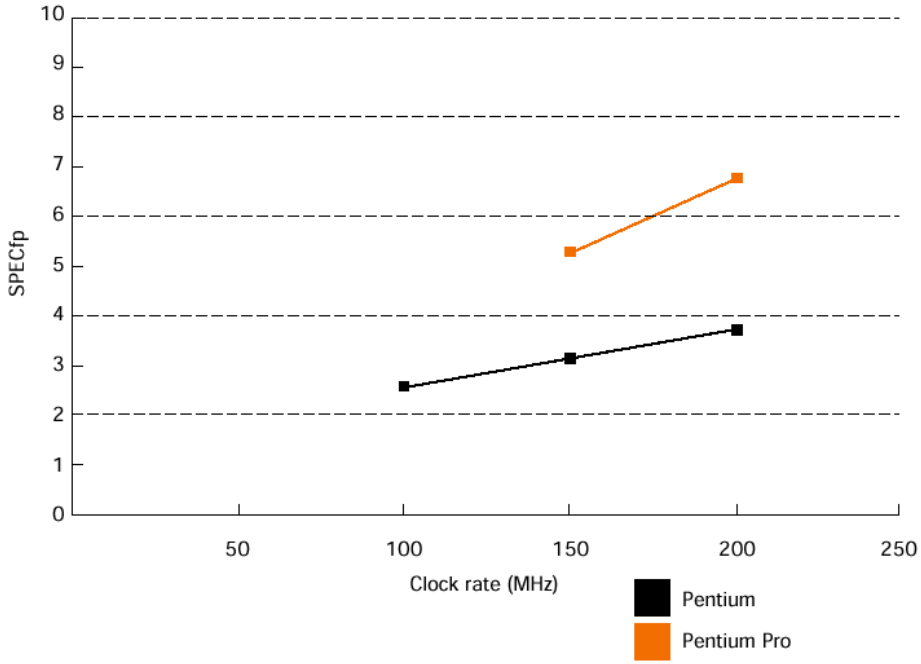
Spec ratio

  - in pratica si tratta di una media geometrica sui vari valori di *Speedup*
    - l'uso degli Speedup nella media evita che *programmi di benchmark computazionalmente più costosi* abbiano un'influenza maggiore sull'indice complessivo calcolato

SPEC ‘95

- Cosa succede raddoppiando solo la frequenza del clock?
  - I valori degli SPEC non raddoppiano ! Questo è soprattutto dovuto all’aumentato CPI, causato anche dai ritardi negli accessi alla memoria
  - Memoria costituisce il von Neumann bottleneck
- Nota che macchine con frequenza di clock minore possono risultare migliori
  - succede se il CPI è più basso (riduzione CPI dovuta a migliorie nelle tecniche di compilazione o nell’architettura interna della CPU), oppure se il compilatore riesce a diminuire il numero di istruzioni eseguite





Legge di Amdahl

- Questa semplice legge fissa un limite agli incrementi di prestazioni ottenibili (Speedup) quando introduciamo delle ottimizzazioni
- Supponiamo che le ottimizzazioni siano in grado di *ridurre* solo una parte del tempo totale di esecuzione originale T_{exe}:
  - $1/s\ T_{exe}$: frazione di T_{exe} non modificata dalle ottimizzazioni
  - $(1- 1/s)\ T_{exe}$: frazione di T_{exe} ridotta tramite le ottimizzazioni
  - n : fattore del miglioramento ottenuto tramite le ottimizzazioni
- Legge di Amdahl: $$T_{ott} = 1/s\ T_{exe} + ((1- 1/s)\ T_{exe}) / n$$
- La legge di Amdahl fissa un limite allo speedup massimo ottenibile
  - se l'ottimizzazione è molto consistente, n diventa molto grande, per cui possiamo approssimare: $T_{ott} \cong 1/s\ T_{exe}$
  - il massimo Speedup ottenibile (per n molto grande):

$$Speedup_{max} = T_{exe} / (1/s\ T_{exe}) = s$$

- Corollario alla legge di Amdahl (motivazione CPU RISC):

  è meglio rendere più veloce i casi più comuni (es. le istruzioni più usate, per la cui esecuzione si impiega la maggior parte di T_{exe})

Esempio di applicazione di Amdahl

- *Un programma viene eseguito in 100 s su una macchina. Di questo tempo, 80 s sono usati per le moltiplicazioni. Di quanto dobbiamo incrementare la velocità delle moltiplicazioni se vogliamo che il programma sia 4 volte più veloce?*
  - Se T_{ott} deve risultare uguale a 1/4 T_{exe}
  - Applicando la legge di Amdahl:

$$T_{ott} = 1/s\ T_{exe} + ((1- 1/s)\ T_{exe}) / n$$

$$0.25\ T_{exe} = 0.2\ T_{exe} + (0.8\ T_{exe}) / n$$

$$0.25\ 100 = 0.2\ 100 + (0.8\ 100) / n$$

$$25 = 20 + 80/n$$

$$n = 80/5 = 16$$

- *Incrementando solo la velocità delle moltiplicazioni, qual è lo speedup massimo teorico?*
  - ottimizzando al massimo, potremmo pensare che il tempo per le moltiplicazioni tenda a 0, ovvero che $0.8\ T_{exe} / n \rightarrow 0$
  - quindi, $T_{ott} = 0.2\ T_{exe} = 20$ da cui $Speedup_{max} = T_{exe}/T_{ott} = 100/20=5$

Considerazioni finali

- La misura delle prestazioni su una data piattaforma è significativa
  - se è relativa allo specifico programma
  - se è basata sul tempo totale di esecuzione effettivo

- Rispetto ad una data ISA gli incrementi delle prestazioni derivano da
  - incremento della frequenza del clock (se questo non aumenta considerevolmente il CPI)
  - miglioramento dell'organizzazione del processore per abbassare il CPI
  - miglioramenti del compilatore per ridurre CPI medio o IC

- Bisogna imparare a leggere le pubblicità che riguardano le prestazioni dei computer
  - diffidare ad esempio di pubblicità che riguardano i MIPS o i MFLOPS/ GFLOP di picco
  - prestazioni di picco ottenibili solo con programmi non significativi, che dovrebbero eseguire una sequenza lunga di istruzioni corte (con CPI piccolo), tutte indipendenti, senza accessi alla memoria

Confronto tra architetture a singolo e a multi ciclo

- Con le nozioni precedenti sulla valutazione delle prestazioni, possiamo confrontare le due architetture, a singolo e a multi-ciclo, descritte nel cap. 5
- Singolo ciclo
  - CPI =1
  - Ciclo di clock (periodo) = 8 ns
    - calcolato sulla base dell'istruzione più "costosa": `lw`
  - T_{exe}= IC * CPI * Periodo_clock = IC * 8 ns
- Multi ciclo: necessario conoscere la frequenza delle varie istruzioni
  - lw CPI_0=5 presenti nel 22% IC
  - sw CPI_1=4 presenti nel 11% IC
  - R-type, CPI_2=4 presenti nel 49% IC
  - branch CPI_3=3 presenti nel 16% IC
  - jump CPI_4=3 presenti nel 2% IC

  - CPI_{avg} = $0.22\ CPI_0 + 0.11\ CPI_1 + 0.49\ CPI_2 + 0.16\ CPI_3 + 0.02\ CPI_4$ = $0.22 * 5 + 0.11 * 4 + 0.49 * 4 + 0.16 * 3 + 0.02 * 3 = 4.04$
  - T_{exe}= IC * CPI_{avg} * Periodo_clock = IC * 4.04 * 2 ns = IC * 8.08 ns

- CPI_{avg} = $0.22\ CPI_0 + 0.11\ CPI_1 + 0.49\ CPI_2 + 0.16\ CPI_3 + 0.02\ CPI_4$ =
  $0.22 * 5 + 0.11 * 4 + 0.49 * 4 + 0.16 * 3 + 0.02 * 3 = 4.04$
- T_{exe} = IC * CPI_{avg} * Periodo_clock = IC * 4.04 * 2 ns = IC * 8.08 ns

- equivalente a:

- T_{exe} = (22% * IC * CPI_0 + 11% * IC * CPI_1 + 49% * IC * CPI_2 +
  16% * IC * CPI_3 + 2% * IC * CPI_4) * Periodo_clock =

  IC * ($0.22\ CPI_0 + 0.11\ CPI_1 + 0.49\ CPI_2 + 0.16\ CPI_3 + 0.02\ CPI_4$) *
  Periodo_clock = IC * 4.04 * 2 ns = IC * 8.08 ns