Lezione 5
Valutazione delle Prestazioni

http://www.dii.unisi.it/~giorgi/didattica/arcal1

Prestazioni - Introduzione

- Buone prestazioni hardware $\Rightarrow$ efficacia dell'intero sistema
- Difficolta' a misurare le prestazioni
  - complessita' dei moderni sistemi software
  - peculiarita' hardware non sempre ideali in qualsiasi situazione
- Non basta il manuale del Set di Istruzioni e un insieme significativo di applicazioni
  - le metriche possono variare a seconda del tipo di applicazione
- Le prestazioni sono un parametro decisionale per l'acquisto di un calcolatore
  - i produttori e i venditori hanno interesse a mettere in luce gli aspetti piu' convenienti per loro
- Quali elementi di un calcolatore influiscono sulla metrica di interesse
  - CPU, Memoria, I/O
- Nostro obiettivo: capire le implicazioni su costi e prestazioni di determinate scelte architetturali

Definizioni

- Le prestazioni sono in unita' di "oggetti-per-secondo"
  - "grande" e' meglio
- Se siamo soprattutto interessati al tempo di risposta
  - $P_X = \frac{1}{E_X}$
  - P_X = prestazioni della macchina X
  - E_X = tempo di esecuzione della macchina X
- Speed up rispetto a una macchina di riferimento
  - $S_{XY} = \frac{P_X}{P_Y} = \frac{E_Y}{E_X}$
  - Lo speed up di X rispetto a Y dice "X e' S_{XY} volte piu' veloce di Y"
  - In percentuali, $s_{xy}=(S_{XY}-1)*100$ dice "X e' s_{xy}% piu' veloce di Y"
- Esempio:
  - E_X = 1sec, E_Y = 1.2sec $\Rightarrow$ $S_{XY} = 1.2 \Rightarrow$ X e' 1.2 volte piu' veloce di Y
    $s_{xy} = 20\% \Rightarrow$ X e' il 20% piu' veloce di Y

Tempo Totale e Tempo di CPU

- T_J = Tempo Totale
  (o Tempo Assoluto o Tempo di Risposta
  o Tempo Trascorso (Elapsed Time))
  - tempo necessario per completare un lavoro; comprende una serie di tempi che sono "allungati" dalle interferenze di altri programmi che generano attivita' come:
    - accesso ai dischi
    - accesso alla memoria
    - attivita' di I/O
    - sovraccarico del Sistema Operativo
- T_{CPU} = Tempo di CPU (Tempo di Esecuzione del programma)
  - Tempo durante il quale un processore ha lavorato su un singolo programma
    - senza tempo speso in I/O (puo' includere il tempo "dell'uomo")
    - senza tempo per eseguire altri programmi
  - Puo' essere suddiviso in
    - T_U = tempo di CPU relativo all'utente
    - T_K = tempo di CPU relativo al Sistema Operativo (K=Kernel)

$$T_U + T_K = T_{CPU}$$

Tempo di CPU misurato in UNIX/LINUX

- Dato il programma myprogram

```
time <myprogram>

90.7u  12.9s  2:39  65%
```

T_U

T_K

T_J

$$\frac{T_{CPU}}{T_J}\ \% = \frac{(T_U+T_K)}{T_J} * 100$$

Per il 35 % la macchina e' usata
- per I/O
- per altri programmi
- per altra attivita' di sistema

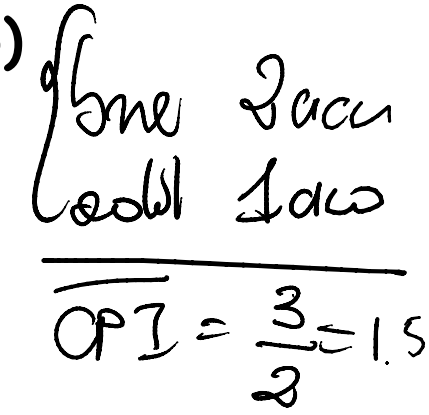
- TK spesso non viene incluso
  - perche' puo' nascondere inaccuratezze di misura
  - il calcolo puo' risultare sbilanciato se si usano sistemi operativi diversi
  - su alcune macchine il codice puo' essere considerato di kernel e su altre di utente
  - d'altra parte nessun programma puo' essere eseguito senza usare qualche parte di kernel

- Prestazioni di Sistema $\Rightarrow T_J$ Prestazioni CPU $\Rightarrow T_U$

Equazione delle prestazioni

$$T_{CPU} = C_{CPU} \cdot T_C = \frac{C_{CPU}}{fc} = N_{CPU} \cdot \overline{CPI} \cdot T_C$$

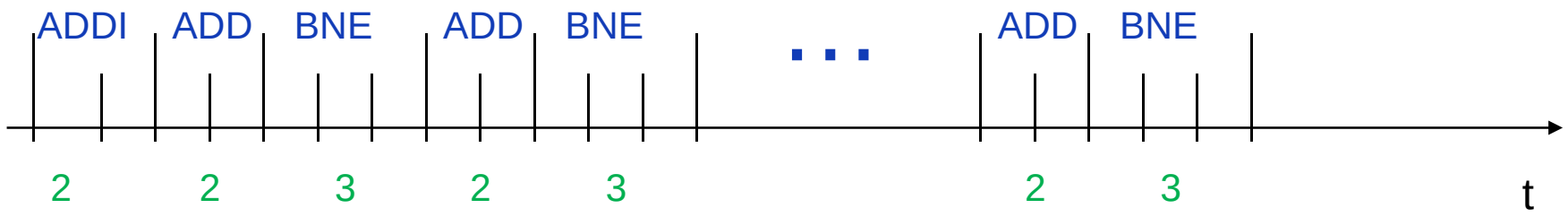
- T_C e' il periodo di clock (f_C e' la frequenza di clock)
- C_{CPU} sono i cicli di clock
- N_{CPU} e' il Numero di istruzioni eseguite DINAMICAMENTE
- /CPI e' il numero di Cicli Per Istruzione (medio)



Esempio

Assumiamo $18=0, $19=0 :

```
        addi $19, $19, 100
 loop:  add  $18, $18, 1
        bne  $18, $19, loop
```



$C_{CPU}=502$, $N_{CPU}=201$

/ CPI= 502/201 ≈ 2.5

- Per ottenere maggiori prestazioni si puo':
  - ridurre i cicli di clock per un programma
  - aumentare la frequenza di clock
- Spesso pero' l'abbassamento del numero di cicli comporta invece un abbassamento della frequenza di clock

Esempio 1

- $T_{CPU}(A) = 10s \quad f_C(A) = 400\ MHz$
- $T_{CPU}(B) = 6s \quad f_C(B) = ?$

Supponiamo che in B si riesca a ridurre il tempo ma con il risultato di avere

$$C_{CPU}(B) = 1.2 * C_{CPU}(A)$$

- Quanto deve essere la frequenza di clock della macchina B affinche' il tempo di esecuzione del programma sia 6s?

- Soluzione:

$$C_{CPU}(A) = (f_C * T_{CPU})|_A = 4 * 10^9$$

$$f_C = (C_{CPU} / T_{CPU})|_B = (1.2 * 4 * 10^9 / 6) = 800\ MHz$$

- Per ottenere una diminuzione del 40% del tempo di esecuzione e' necessario raddoppiare la frequenza di clock

Esempio 2

- $f_c(A)$ = 1 GHz $f_c(B)$ = 500 MHz
- /CPI(A) = 2.5 /CPI(B) = 1.2
- Quale calcolatore e' piu' veloce?
  (il set di istruzioni rimane lo stesso)

- $S_{AB} = \frac{T_{CPU}(B)}{T_{CPU}(A)} = \frac{N_{CPU}(B)*/CPI(B)*f_c(A)}{N_{CPU}(A)*/CPI(A)*f_c(B)} = \frac{1.2*1000}{2.5*500}$
- set istruzioni uguale → $N_{CPU}(A) = N_{CPU}(B)$
- S_{AB}= 0.9 → B e' 1.1 volte piu' veloce di A

Come misurare i parametri fondamentali dell'equazione

- T_{CPU} → orologio
- fc → dato di targa
- N_{CPU}, /CPI → difficili da misurare

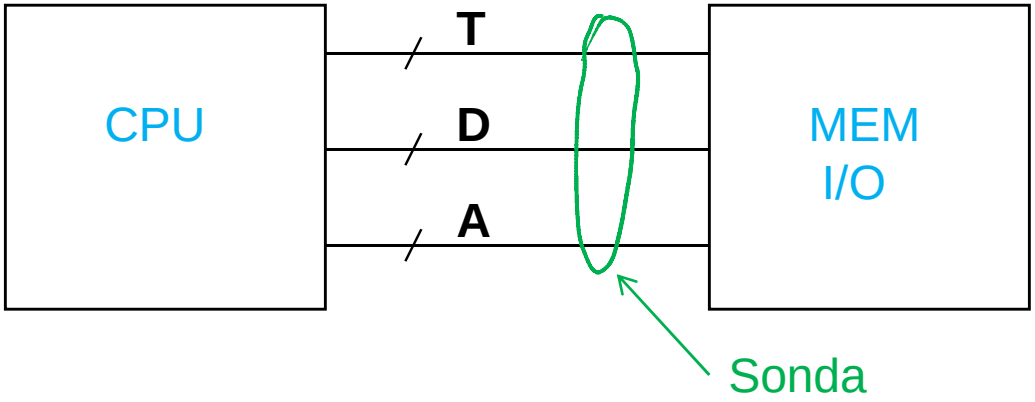
	N_{CPU}	/CPI	fc
Programma	X		
Compilatore	X	(X)	
Sistema Operativo	X		
ISA	X	X	
Struttura CPU		X	X
Sottosistema di Memoria		X	
Tecnologia			X

Motivi delle dipendenze dei parametri fondamentali

	N_{CPU}	/CPI	f_C
Programma	X		
Compilatore	X	(X)	
Sistema Operativo	X		
ISA	X	X	
Struttura CPU		X	X
Sottosistema di Memoria		X	
Tecnologia			X

- **Programma**
  - **Fibonacci-ricorsivo vs. Fibonacci-iterativo → N_{CPU}**
- **Compilatore**
  - **Opzioni –O1, -O2, -O3 → N_{CPU}, /CPI**
- **Sistema Operativo**
  - **Una syscall puo' essere implementata diversamente in Linux rispetto a Windows → N_{CPU}**
- **Instruction Set (ISA)**
  - **MIPS vs. x86 vs. ARM, #cicli necessari per es. add → N_{CPU}, /CPI**
- **Struttura CPU**
  - **Una pipeline elabora piu' istruzioni per ciclo → /CPI**
  - **Stadi (logica combinatoria) con meno porte logiche → maggiore fc**
- **Sottosistema di memoria**
  - **Es. add → 1 ciclo, lw → 100 cicli**
- **Tecnologia (elettronica)**
  - **Es. 14 nm invece che 90nm**

Come valutare NCPU (1)



T = TYPE={IF,DR,DW}

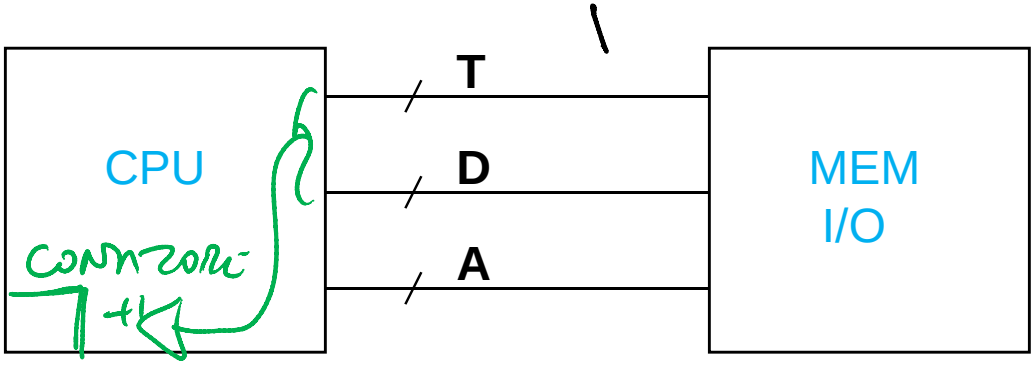
FETCH DATA-READ DATA-WRITE

D = DATA

A = ADDRESS

- Strumentazione Hardware (sonda o probe) che legge le istruzioni che il processore preleva dalla memoria
- Richiede di poter aprire la macchina
- Richiede la disponibilita' di una opportuna sonda
- Richiede la disponibilita' di un analizzatore di stati logici

Come valutare NCPU (2)



T = TYPE={IF,DR,DW}

FETCH DATA-READ DATA-WRITE

D = DATA

A = ADDRESS

- Contatori Hardware che contino il numero di istruzioni
  - Disponibili sui processori Intel (ed Alpha)
- Supporto da parte del processore e del sistema
  - Alpha: ATOM tools, Intel VTUNE
- Inserimento di "sonde software" (program instrumentation)
  - MIPS R4400: Pixie
- Simulatori dell'architettura (tecnica Software)
  - Es. SPIM

Come valutare NCPU (3)

Codice Originale

```
add $1 $2 $3    #1ciclo
lw $1, 0($2)    #100cicli
bne $5,$6,Loop  #3cicli
```

Codice Instrumentato

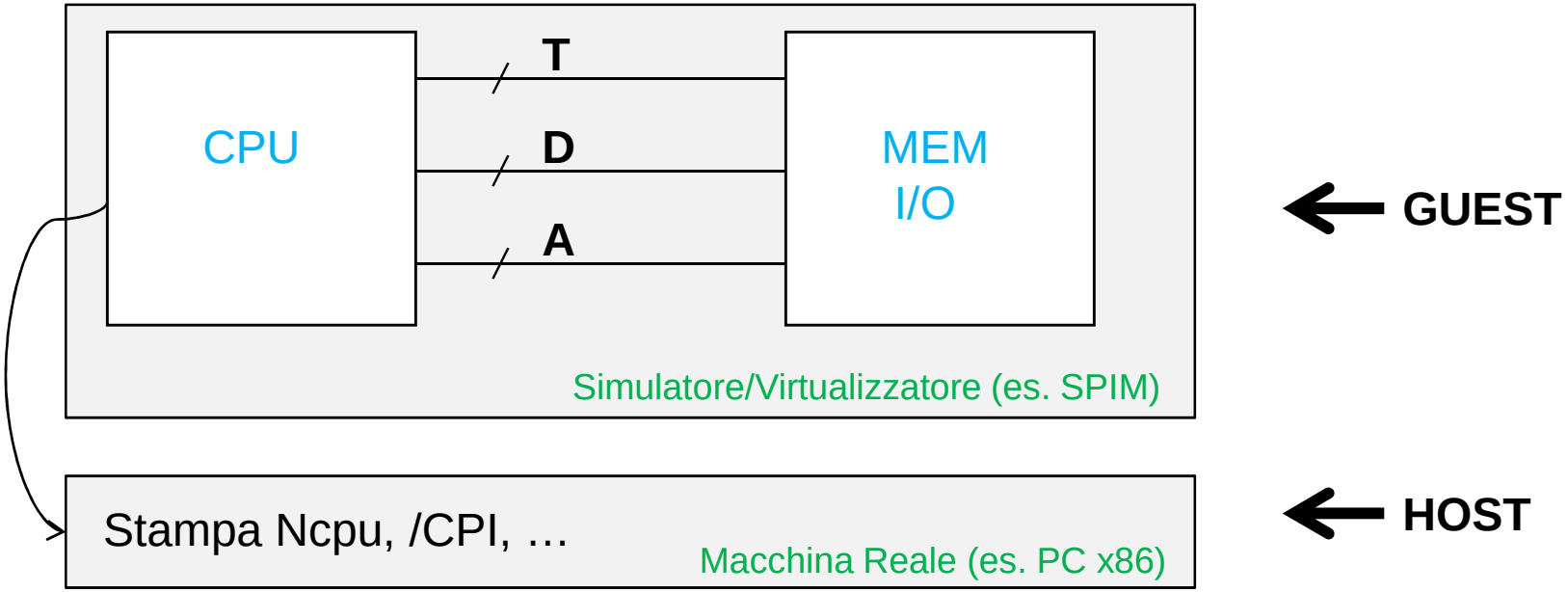
```
add  $25,3
add  $26,104
add $1 $2 $3
lw $1, 0($2)
bne $5,$6,Loop
```

1) Riservo un registro (es. $25) per contare le istruzioni
2) Riservo un registro (es. $26) per contare i cicli di ogni basic block*
3) Inserisco due istruzioni per tali conteggi per ogni basic-block* del programma (instrumentazioni)
4) Al termine del programma, $25 contiene Ncpu mentre $26/$25 e' pari a /CPI

- Metodo usato in vari processori
  - MIPS R4400: Pixie Alpha: Atom Intel: VTUNE

*Basic-block=gruppo di istruzioni con un solo punto di ingresso e un solo punto di uscita

Come valutare NCPU (4)



- Usare dei simulatori/virtualizzatori/emulatori
  - SPIM → simulatore di processore MIPS
  - QEMU → emulatore molto diffuso sotto Linux → virtual machine
  - Virtualbox, Vmware, VirtualPC → ulteriori virtual machines

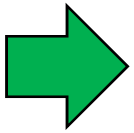
Come calcolare /CPI

Si puo' calcolare da quante istruzioni di ogni tipo vengono eseguite ($N_{CPU,K}$)

$$\overline{CPI} = \sum_{K=1}^{T} \overline{CPI}_K = \sum_{K=1}^{T} p_{CPU,K} \cdot CPI_K = \frac{1}{N_{CPU}} \cdot \sum_{K=1}^{T} N_{CPU,K} \cdot CPI_K$$

T = numero tipi diversi di istruzioni
$N_{CPU,K}$ = numero di istruzioni di tipo K
$p_{CPU,k}$ = frequenza relativa di quel tipo di istruzione $\equiv N_{CPU,K}/N_{CPU}$
CPI_K = CPI delle istruzioni di tipo K
$/CPI_K$ = CPI MEDIO delle istruzioni di tipo K $\equiv p_{CPU,k} * CPI_K$

Op	$p_{CPU,K}$ $\equiv N_{CPU,K}/N_{CPU}$	CPI_K	$/CPI_K$ $\equiv p_{CPU,k} * CPI_K$	$\%T_{CPU}$ CPI_K : /CPI
Aritmethic	0.50	1	0.5	33%
Load	0.20	2	0.4	27%
Store	0.10	2	0.2	13%
Branch	0.20	2	0.4	27%



/CPI = 0.5+0.4+0.2+0.4 = 1.5

Esempio: scelta di compilatori per una macchina

- Per le istruzioni di tipo A, B, C: $/CPI_A=1$ $/CPI_B=2$ $/CPI_C=3$

COMPILATORE	Istr. Tipo A	Istr. Tipo B	Istr. Tipo C
CC1	2	1	2
CC2	4	1	1

- Quale compilatore produce il minor numero di istruzioni?
  - $N_{CPU}(1) = 5$ $N_{CPU}(2) = 6$
  - il compilatore 1 produce il codice piu' corto
- Quale compilatore produce il codice piu' veloce?
  - $C_{CPU}(1) = 2*1 + 1*2 + 2*3 = 10$ cicli
  - $C_{CPU}(2) = 4*1 + 1*2 + 1*3 = 9$ cicli
  - il compilatore 2 produce, pero', il codice piu' veloce!
- Qual e' il /CPI di ciascuno dei due compilatori?
  - $/CPI(1) = C_{CPU}(1)/N_{CPU}(1) = 10/5 = 2$
  - $/CPI(2) = C_{CPU}(1)/N_{CPU}(1) = 9/6 = 1.5$
  - il compilatore 2 produce il codice con il miglior /CPI

Scalabilita'

$$P \propto 1/T_{CPU} = \frac{1}{N_{CPU} \cdot \overline{CPI}} \cdot f_C$$

- **Ad un raddoppio di f_c ci aspettiamo un raddoppio di P, ma cio' non avviene… (cfr. SPEC Pentium)**
  - **Il sottosistema di memoria 'non tiene il passo del processore'**
- **La capacita' di fornire maggiori prestazioni al variare di un parametro viene chiamata *scalabilita' rispetto a quel parametro***
  - **Il processore PentiumPro risulta *piu' scalabile* del processore Pentium *con la frequenza***
  - **In particolare:** $P_{200}/P_{100}=f_{200}/f_{100} * CPI_{100}/CPI_{200}$ → $CPI_{200} \neq 2*CPI_{100}$

Legge di Amdhal

- In generale puo' essere difficile stabilire l'esatta dipendenza da un dato parametro (anche quando questo compare esplicitamente nell'equazione!)

- Per questo motivo viene utilizzata la seguente formula, per mettere in relazione lo speedup di una parte del sistema con lo speedup dell'intero sistema:

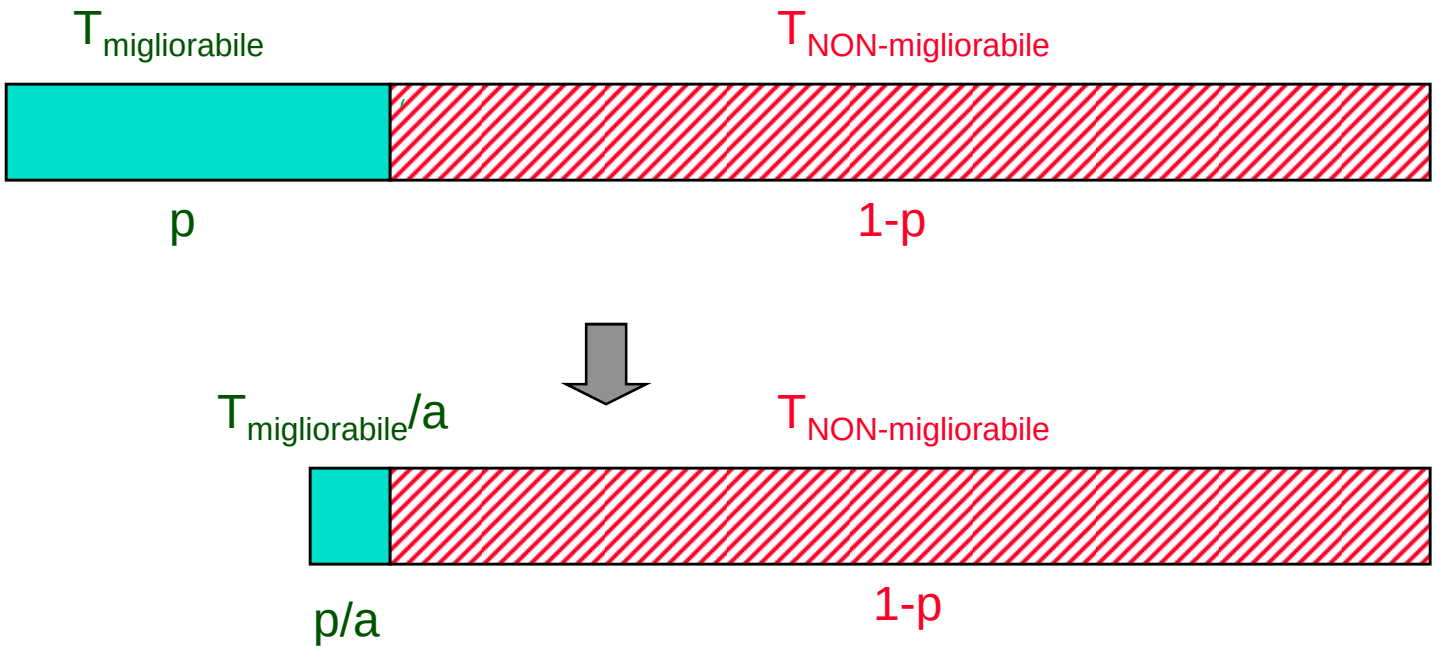
$$S = \frac{P_{dopo}}{P_{prima}} = \frac{T_{prima}}{T_{dopo}} = \frac{T_{prima}}{\frac{T_{migliorabile}}{a} + T_{NON-migliorabile}} = \frac{1}{\frac{p}{a} + (1-p)}$$

- Dove p=% del tempo 'migliorabile'=$T_{migliorabile}/T_{prima}$

  a=speed-up della parte di sistema migliorabile

Legge di Amdhal (2)

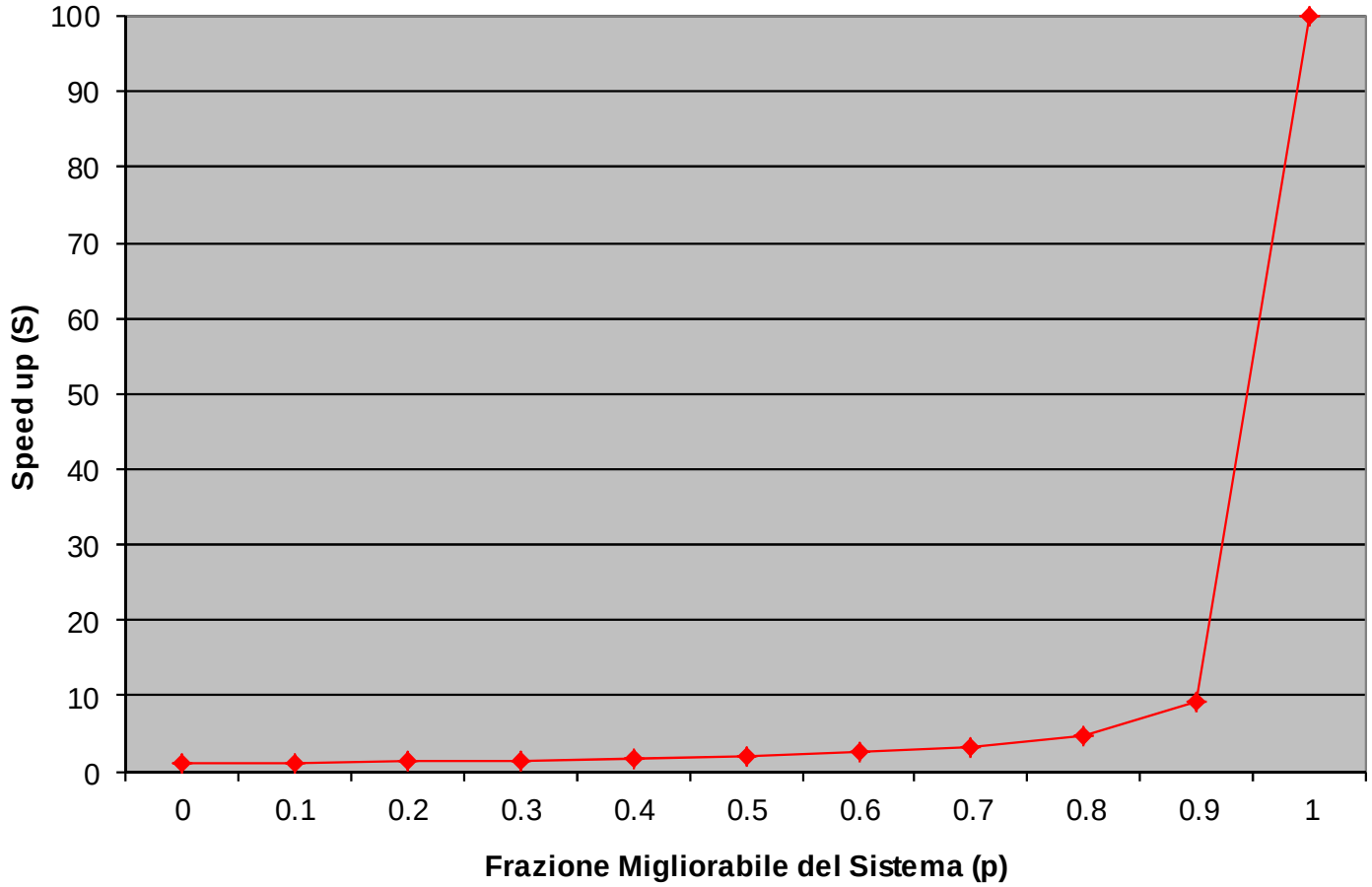
$$S = \frac{1}{\frac{p}{a} + (1 - p)}$$



p=% del tempo 'migliorabile'=$T_{migliorabile}/T_{prima}$

a=speed-up della parte di sistema migliorabile

Legge di Amdhal (3)



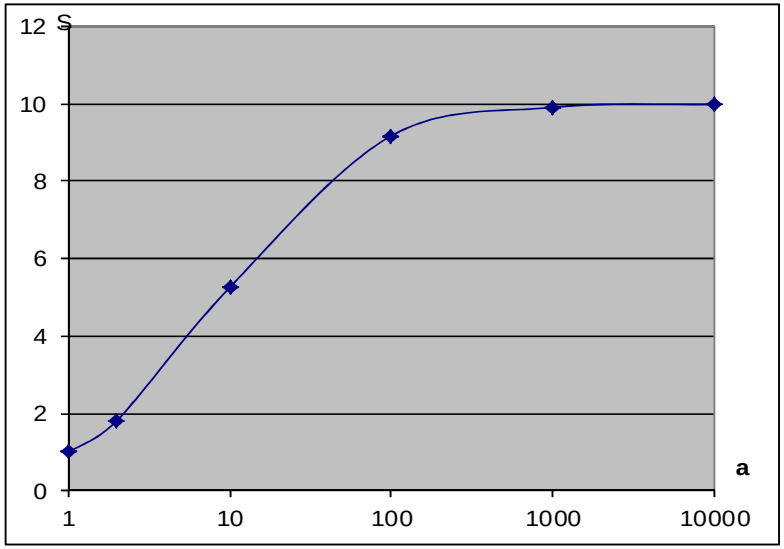
- Ipotesi: la frazione migliorabile ha uno speed-up (a) pari a 100

→ E' importante migliorare una parte che e' consistente (>0.8)

Esempio

- **Supponiamo che il 90% di un programma possa essere eseguita in parallelo (es. aggiungendo piu' processori)→p=0.9**

a	S	p
1	**1.000**	0.90
2	**1.818**	0.90
10	**5.263**	0.90
100	**9.174**	0.90
1000	**9.911**	0.90
10000	**9.991**	0.90



Per a→ ∞ il massimo speed-up ottenibile
e' pari a 1/0.1=10

→Invece di aggiungere 9999 processori conviene parallelizzare anche una parte del restante 10%

Conclusioni dalla legge di Amdhal

1) E' molto importante migliorare la parte piu' utilizzata del sistema!

2) A un certo punto e' importante anche migliorare la parte meno utilizzata del sistema…

Definizioni

- **WORKLOAD**
  insieme di programmi
  abitualmente usati su un dato calcolatore
- **BENCHMARK**
  programma campione che gli utenti sperano abbia
  un comportamento simile al proprio workload (e.g. SPEC2000)
- **SMALL-KERNEL**
  programma "giocattolo" usato per le fasi di testing
  di prototipi (e.g. Lawrence Livermore Loops, Linpack)
- **MICRO-BENCHMARK**
  sequenze di istruzioni ritenute significative (e.g. REPS MOVE)
- I migliori benchmark sono i programmi reali
  - campo ingegneristico: programmi scientifici/ingegneristici
  - software developers: compilatori, sistemi di documentazione
- I moderni compilatori possono ottimizzare il codice
  in base al tipo di programma (scientifico, office, …)
  - nota: ottimizzazioni troppo spinte possono produrre codice assembly NON CORRETTO

Tipi di Benchmark

Vantaggi		Svantaggi
• rappresentativo	Carico effettivo (workload)	• molto specifici • non portabili • difficili da eseguire, o da misurare • difficile identificare cause
• portabilita' • ampiamente usati • catturano i miglioramenti	Set di Benchmark (SPEC2000,TPC)	• meno rappresentativo
• facili da seguire, usati nelle prime fasi del ciclo di progetto	Small "Kernel" (e.g. matrix loop)	• facili da fuorviare
• identificare prestazioni di picco e potenziali colli di bottiglia	Micro-benchmarks (e.g REP MOVS)	• il "picco" puo' essere distante dalle prestazioni delle reali appliacazioni

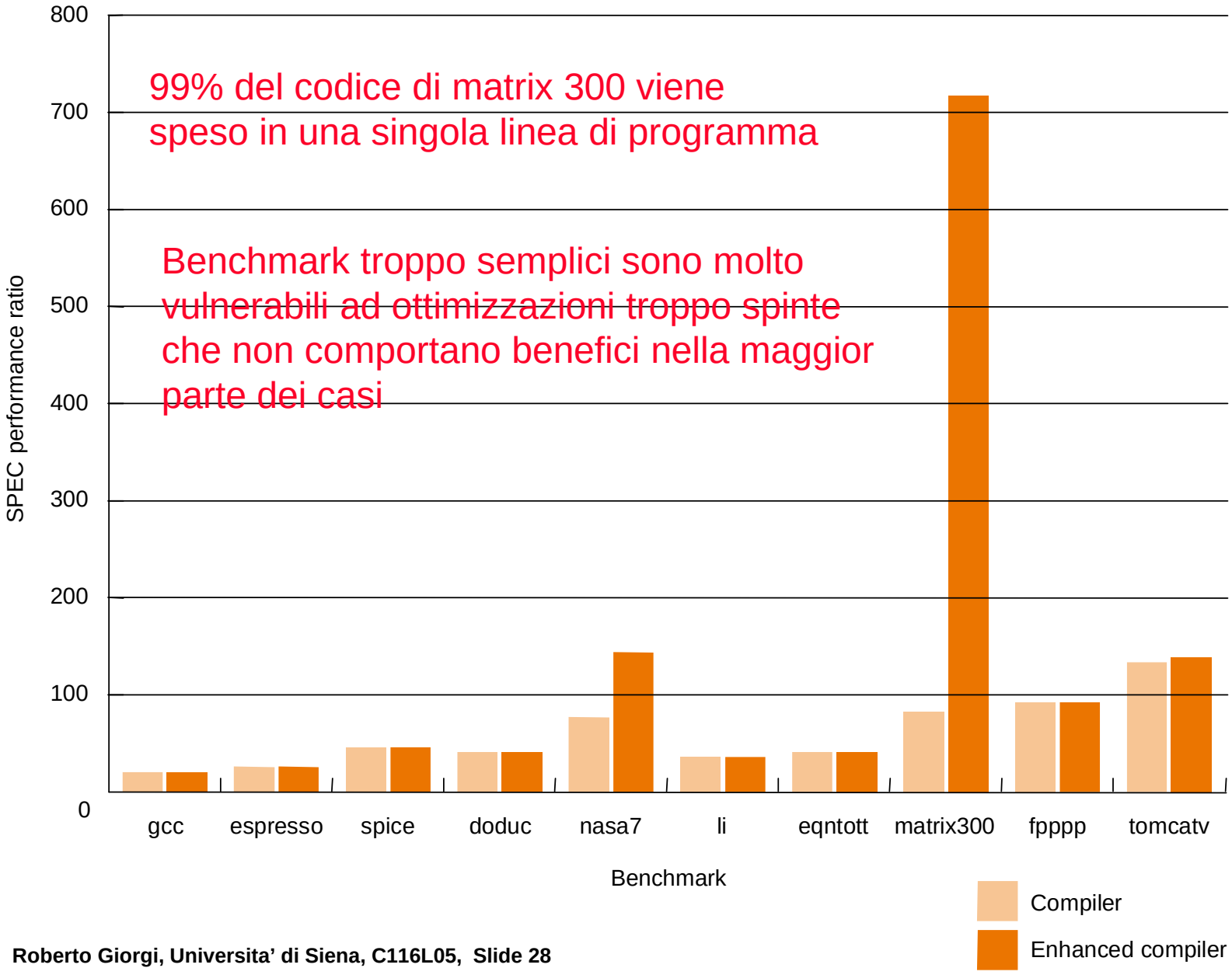
Benchmark Suites

- Insieme di programmi usati per valutare un computer
- Sviluppati e gestiti da organizzazioni non-profit (e.g. Standard Performance Evaluation Corp=SPEC, PC Magazine, …) → www.spec.org www.tpc.org
- Mettono sotto test ben definiti aspetti del sistema:
  - performance delle applicazioni da ufficio
  - performance delle applicazioni database
  - prestazioni CPU+memoria
  - Performance grafiche
- Esempi
  - SPEC95, SPEC2000, SPEC2006
  - TPC-A, TPC-B (carichi transazionali puri)
  - TPC-C (On-Line Transaction Processing o OLTP = Database + WEB server) →TPC-H, TPC-S, TPC-E
  - TPC-D (Decision Support Systems o DSS)
  - TPC-W (Applicazioni per commercio elettronico e Internet)

SPEC Benchmarks

- Prima versione: SPEC89
  - 10 programmi forniscono un singolo numero (SPECmark)
  - Macchina di riferimento: VAX 11/780
- Seconda versione: SPEC92
  - SPECint92 (6 programmi con operazioni su interi)
  - SPECfp92 (14 programmi con operazioni su floating point)
  - Le opzioni del compilatore NON possono essere qualsiasi
    - spice:
      unix.c:/def=(sysv,has_bcopy,"bcopy(a,b,c)=memcpy(b,a,c)")
    - wave5:
      /ali=(all,dcom=nat) /ag=a /ur=4 /ur=200
    - nasa7:
      /norecu /ag=a /ur=4 /ur2=200 /lc=blas
- Terza versione: SPEC95
  - Macchina di riferimento SPARCStation 10/40
- Quarta versione: SPEC2000
- Quinta versione: SPEC2006

Risultati SPEC89: IBM PowerStation550, 2 compilatori



SPEC 95

- 18 applicazioni (benchmark) con i relativi dati di ingresso che riflettono uno workload di carattere tecnico

- Otto benchmark con prevalenti operazioni su interi
  - `go, m88ksim, gcc, compress, li, ijpeg, perl, vortex`
- Dieci benchmark con prevalenti operazioni su floating point
  - `tomcatv, swim, su2cor, hydro2d, mgrid, applu, turb3d, apsi, fppp, wave5`
- Devono essere eseguiti con le opzioni standard dei compilatori

Caratteristiche da definire

- Codice del benchmark
- Dati di ingresso
- Dati attesi in uscita
- Opzioni del compilatore
- Caratteristiche della macchina
- Tipo di sistema operativo

Esempio caratteristiche prova

- **Hardware**
  - numero del modello — Powerstation550
  - CPU — 41.67MHz POWER 614
  - FPU — Integrata
  - Numero di CPU — 1
  - Dimensione Cache — 64Kbyte dati, 8Kbyte istruzioni
  - Memoria — 64MB
  - Sottosistema dischi — 2.4 Gbyte SCSI
- **Software**
  - tipo e versione del Sistema Operativo — AIX 3.1.5
  - versione compilatore — AIX XL/C 1.1.5, AIX XL Fortran 2.2
  - Altro software — Nessuno
  - Tipo di filesystem — AIX
  - Livello di firmware — -
- **Sistema**
  - Parametri Sis. Op. — Nessuno
  - carico background — Nessuno
  - Stato del sistema — Multiuser

Ricapitolazione

- Metriche a tempo di progetto del sistema
  - Il sistema puo' essere implementato, in quanto tempo e a che costo?
  - Il sistema puo' essere programmato, facilita' di compilazione
- Metriche statiche
  - Quanti byte occupa il programma?
- Metriche dinamiche
  - Quante istruzioni vengono eseguite?
  - Quanti byte il processore preleva per eseguire il programma?
  - Quanti cicli servono mediamente per una istruzione?
  - Quanto e' efficace "tirare" sul clock?
- Miglior metrica:
  - Tempo di esecuzione del mio programma!