prestazioni CPU

M. Arrigoni Neri & *P. Borghese*

indice

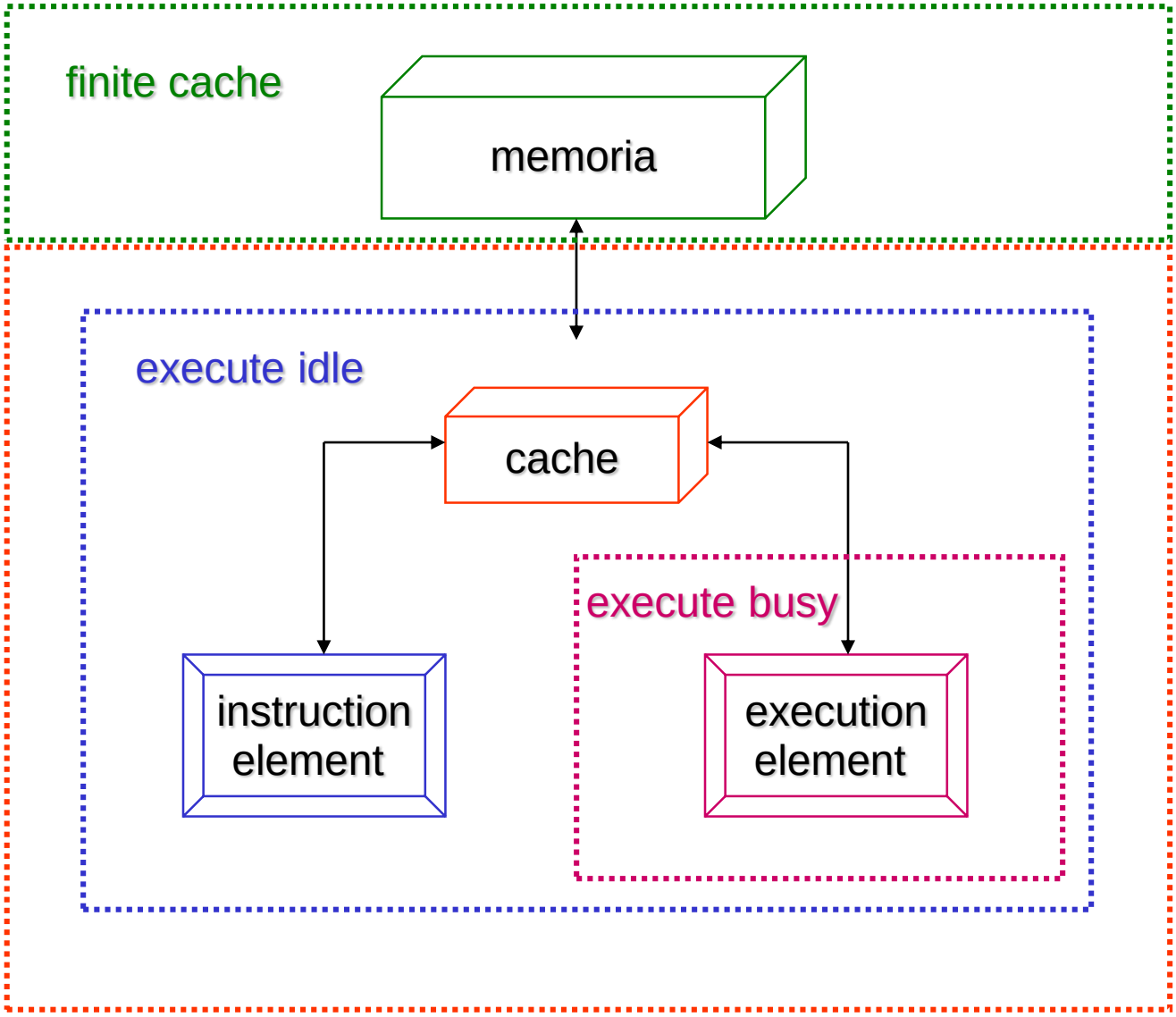
- potenza di CPU
  - metriche di misura
  - misure e confronti di potenza (*CPU e sistemi*)
- overhead
  - ripartizione dei consumi fra componenti del carico
- legge di Amdahl

potenza di CPU

Cicli Per Istruzione (CPI)

- Architettura: insieme delle regole che descrivono le funzioni logiche di una macchina, come sono viste da un programma
  - non specifica cosa l'hardware fa, ma che cosa *appare fare*
    - consente al programma di funzionare in modo consistente su macchine di diversa microarchitettura
  - consideriamo per ora un monoprocessore scalare cioè un processore singolo che esegue serialmente le istruzioni e i comandi che costituiscono il workload (programma)
    - superscalare decodifica ed esegue più istruzioni per ciclo
- CPI: metrica di prestazioni composta da tre addendi relativamente indipendenti:
  1. parte inerente all'esecuzione delle istruzioni
  2. parte relativa agli effetti della pipeline (eventuale attesa perché esistano le condizioni per l'esecuzione della istruzione: *execute idle*)
  3. parte dovuta alla gerarchia di memoria (cioè attesa perché istruzione e dati siano disponibili: *finite cache*)

Cicli Per Istruzione (CPI) (cont.)

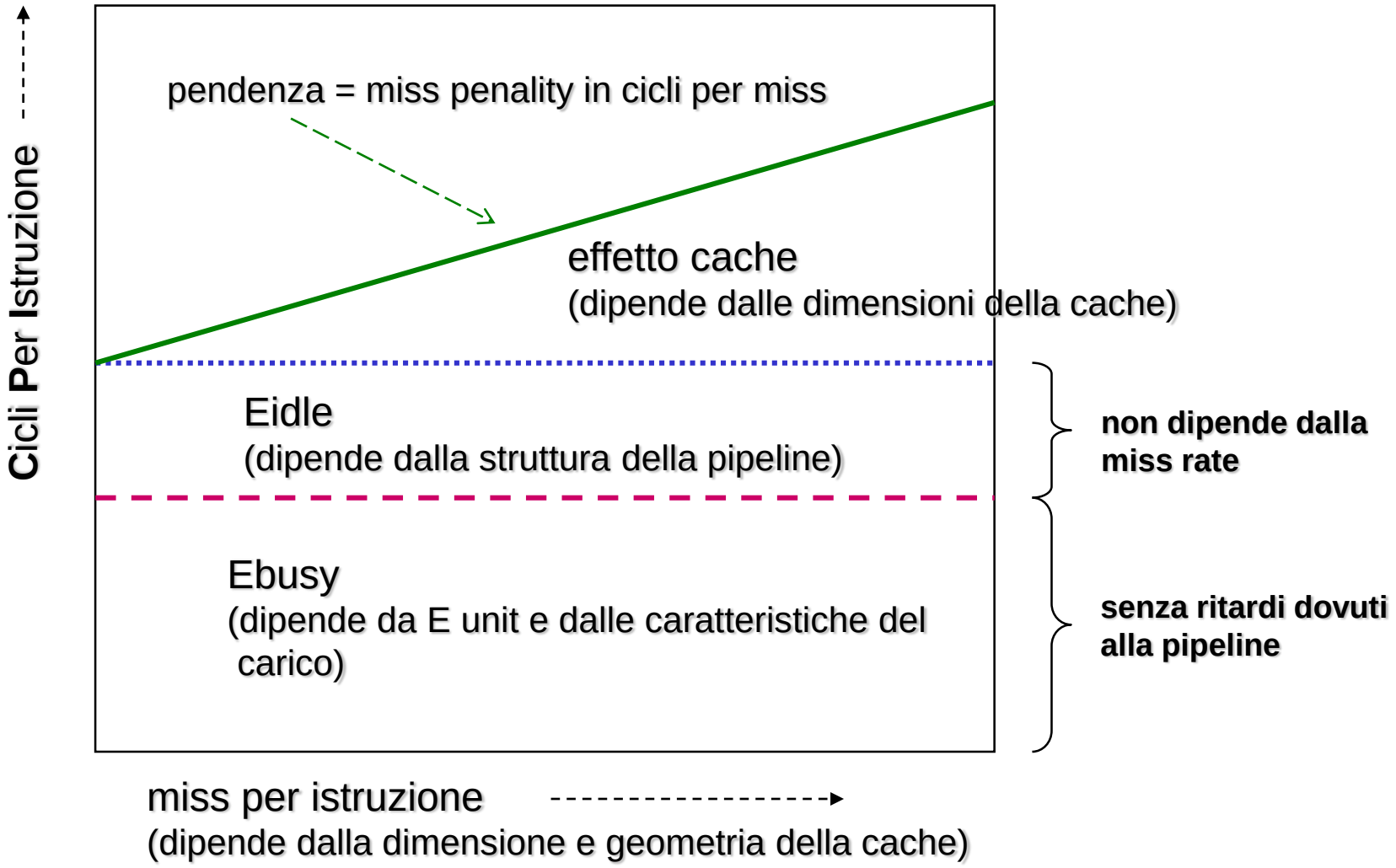


la memoria contiene dati e programmi

diagramma ad alto livello di un processore

istruzioni e dati (operandi) sono in cache distinte

Cicli Per Istruzione (cont.)



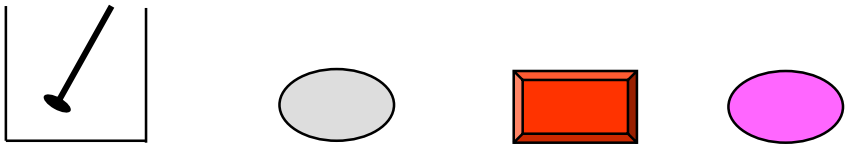
pipeline illustrazione metaforica (fornaio)

impasto
45 min

lievitazione
60 min

forno
70 min

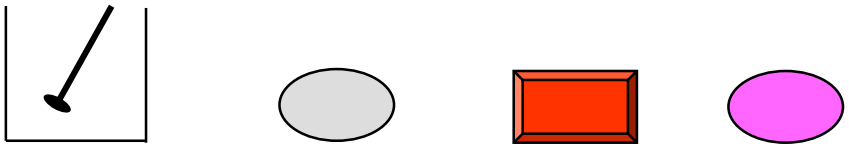
raffreddamento
60 min



durata processo = 235 min
produttività oraria = 60 /235 = 0.255

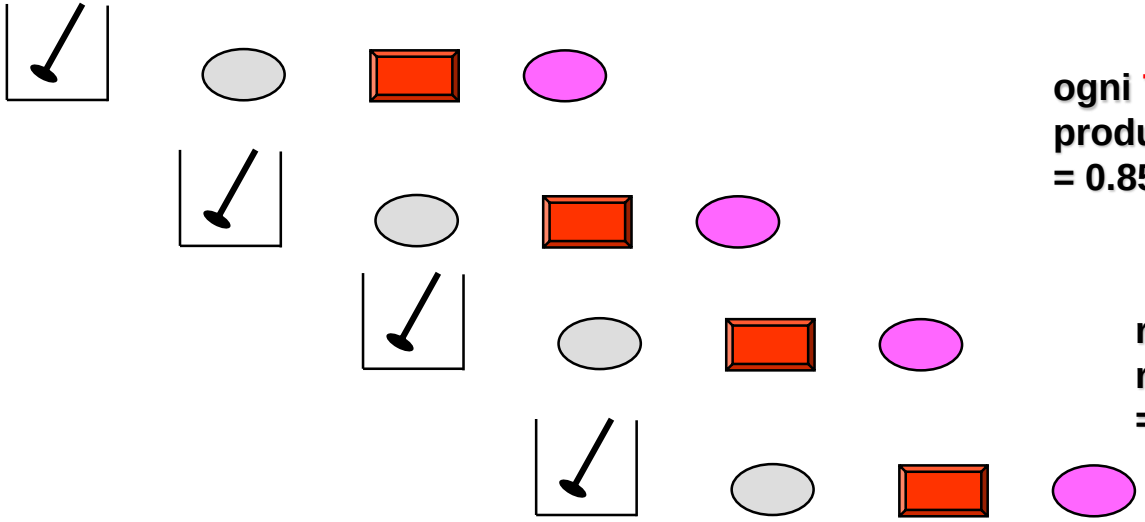
pipeline illustrazione metaforica (fornaio)

impasto **lievitazione** **forno** **raffreddamento**
45 min **60 min** **70 min** **60 min**



durata processo = 235 min
produttività oraria = 60 /235
= 0.255

alimentando i quattro componenti in parallelo (pipeline)



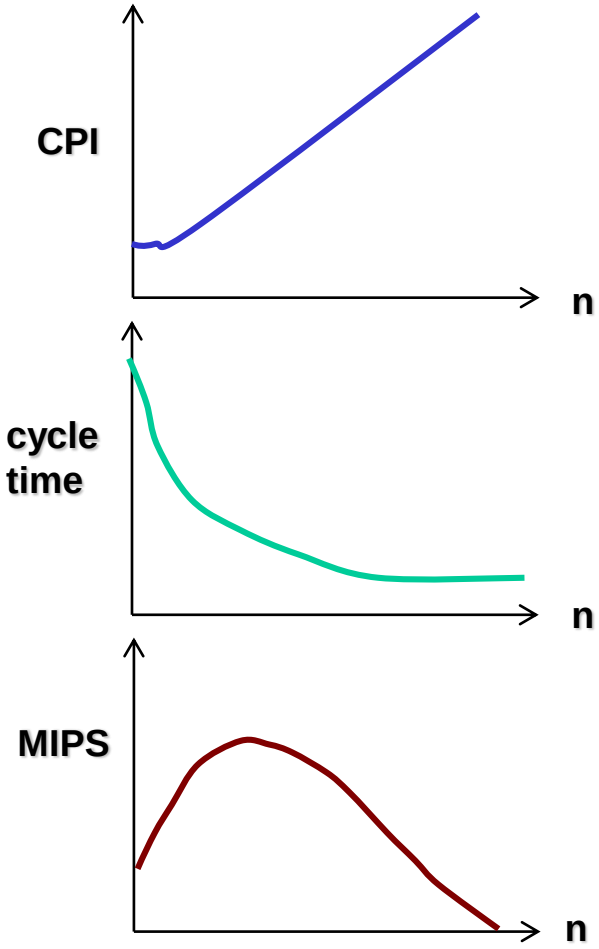
ogni 70 min termina un processo
produttività oraria = 60/70
= 0.857

rendimento della pipeline
rispetto al processo seriale
= 235 / 70 = 3.36

contributo a CPI per evento

- CPI = cicli_per_evento × eventi_per_istruzione
- cicli_per_evento: dipende dal tipo di evento e dalla particolare microarchitettura (può anche essere 0 per qualche tipo di evento)
- eventi_per_istruzione: si suppone noto per ogni *carico* (indipendente dalla microarchitettura)
  - esempio *branch/jump* in un programma (modifica del registro prossima istruzione da eseguire) considerato come evento non come istruzione
  - se ne avviene uno ogni 5 istruzioni, la frequenza dell'evento è 0.2, ognuno causa un ritardo di 3 cicli, allora ogni istruzione incorre mediamente in un 0.6 cicli di ritardo. (un meccanismo di previsione dei *branch* può ridurre la penalità media)
  - eventi che possono causare ritardi:
    - una istruzione necessita di risultati che non sono ancora pronti
    - cache miss - i dati referenziati devono essere caricati da un livello più basso della gerarchia di memoria
    - eventi di serializzazione - espliciti svuotamenti della pipeline per assicurare il corretto funzionamento di certe operazioni complesse

prestazioni della CPU (cont.)

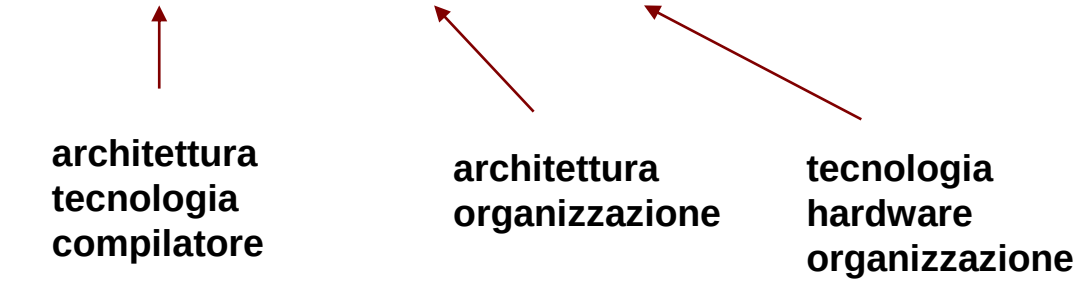


- effetto della granularità della pipeline
  - n = numero degli stadi
  - MIPS = 10^{-6} / (cycle_time $\times$ CPI)
- MIPS ha un andamento circa quadratico con un massimo
- $\Rightarrow$ non conviene aumentare la granularità della pipeline oltre un certo valore
  - un fenomeno di questo genere si verifica sempre all'aumento della complessità (es. col numero processori sul bus di memoria)

equazione delle prestazioni

- tempo_CPU = cicli_CPU(programma) × tempo_ciclo(clock)
  = cicli_CPU(programma) / clock_rate
- IC = Instruction_Count = path_length = numero_istruzioni(prog.)
- CPI = cicli_clock_CPU_programma / IC : cicli di clock per istruzione
- tempo_CPU = IC × CPI × tempo_ciclo = IC × CPI / clock_rate

$$\frac{istruzioni}{programma} \times \frac{cicli\ clock}{istruzione} \times \frac{secondi}{ciclo\ clock} = \frac{secondi}{programma} = tempo\ di\ CPU$$



architettura
tecnologia
compilatore

architettura
organizzazione

tecnologia
hardware
organizzazione

effetto cache

- *cicli effettivi di esecuzione* = CPI + cicli attesa memoria

  CPI + miss_per_istruzione × miss_penality

  CPI + memory_reference_per_istr × miss_rate × miss_penality
- esempio di calcolo:
  - CPI = 2
  - il 40% delle istruzioni accede anche ai dati
  - memory_reference_per_istr = 1 + 0.4
  - miss_rate = 5%
  - miss_penality = 25 cicli
  - cicli attesa memoria = (1+ 0.4) × 0.05 × 25 = 1.75
  - cicli effettivi = 2 + 1.75 = 3.75
  - quanto più veloce sarebbe la macchina se il tasso di hit fosse 100%?

3.75 / 2 = 1.875

la miss penality tende ad essere più grande quando la miss rate è molto bassa (non ci sono porzioni che si sovrappongono) o molto alta (effetto di accodamento)

prestazioni della CPU

$$CPI = \sum CPI_i \frac{IC_i}{IC}$$

- IC_i/IC rappresenta la frequenza dell'istruzione *i* in un programma e CPI_i è il numero medio di cicli per eseguire l'istruzione *i*
- CPI_i è difficile da misurare e non è costante
  - dipende dai dettagli dell'organizzazione del processore che usa tecniche di *pipelining* e di *gerarchie di memoria*
  - non è possibile assegnare a un'istruzione un numero fisso di cicli perché questo *dipende dallo stato del processore* al momento della sua esecuzione
- le istruzioni eseguite nell'unità di tempo (MIPS: Milioni di istruzioni per secondo) non sono un indicatore significativo delle prestazioni ma hanno senso solo come valore di *picco*

prestazioni della CPU (cont.)

- *non esiste una scala universale di prestazioni*
  - MIPS: Milioni di Istruzioni Per Secondo =

    10^{-6} / tempo_per_istruzione
  - MFLOPS: Milioni di Operazioni FLoating point Per Secondo
- sono *indici* di *scarso significato* infatti dipendono oltre che dal *carico* dall'ambiente di esecuzione
- il numero di istruzioni (IC) varia per lo stesso programma in funzione delle condizioni di esecuzione, così il numero di cicli per istruzione (CPI) dipende, anche per la stessa istruzione, dallo stato del sistema.
- CPI $\times$ IC (e quindi il tempo di CPU) non è una grandezza invariante dell'esecuzione di un programma
  - due processori con lo stesso ciclo base non hanno necessariamente le stesse prestazioni
  - il numero di cicli per istruzione non può essere calcolato e nemmeno misurato facilmente

prestazioni della CPU (cont.)

- *ogni ambiente di produzione tende ad avere il suo insieme dominante di istruzioni.*
  - I rapporti di potenza fra processori dipendono dal tipo di carico
- *Microcode*: una tipica istruzione fa il lavoro di un'intera routine di software.
  - in questo caso si possono riscontrare prestazioni superiori con valori di MIPS inferiori.
- *Cache*: le macchine attuali hanno più cache in cui sono mantenute le istruzioni e i dati precedentemente referenziati e che saranno probabilmente referenziati in seguito.
  - il movimento fra la memoria centrale e cache è sovrapposto alla normale attività.
  - le dimensioni e l'organizzazione delle cache hanno un ruolo importante sulle prestazioni di una macchina relativamente a un carico, (lavori con cache hit più elevata hanno una maggiore velocità di esecuzione per istruzione).
- *Effetto multi-processore*: il software di controllo è progettato per gestire queste funzioni.
  - il numero di istruzioni che un programma esegue (in identiche condizioni) dipende anche dal numero di processori a causa della loro interferenza.

esempio pratico: calcolo della potenza di un sistema (per un dato carico)

- definiamo come *potenza* il *numero di blocchi elaborati al secondo da un programma* che ripete indefinitamente il seguente ciclo di tre passi:
  - lettura di un blocco di 8KB su disco
  - elaborazione in CPU (30 M cicli)
  - scrittura di un altro blocco di 8KB su disco
    - il programma è l'unico processo attivo
    - non c'è alcuna sovrapposizione fra operazioni su disco e CPU
- caratteristiche CPU
  - frequenza 5 GHz
  - tempo di elaborazione in CPU per ciclo di programma = $30 \times 10^6 / 5 \times 10^9$ = 0.006 s.

esercizio riassuntivo: calcolo della potenza (cont.)

- caratteristiche operazione I/O disco

	caratteristiche	tempo (ms)	tempo (ms) disco	tempo (ms) cache	
seek		8	8	0	
RPM	15000	2	2	0	
trasf MB/sec	80	0,0977	0,0977	0	
cache MB/sec	320	0,0244	0	0,0244	
controller		1	1	1	
serv. time (ms)			11,0977	1,0244	**7,0684**

- latenza = (60000/15000)/2 = 2 ms.
- trasf (disco) = 8 × 1000 / (80 × 1024) = 0.0977 ms.
- trasf (cache) = 8 × 1000 / (320 × 1024) = 0.0244 ms.
- tempo di servizio I/O = 0.6 × 11.0977 + 0.4 × 1.0244 = 7.0684 ms.
  - (si assume che il dato si trovi nella cache nell' 80% delle letture, pari al 40% del totale delle operazioni)

esercizio riassuntivo: calcolo della potenza (cont.)

- l'esecuzione di un ciclo di elaborazione richiede perciò:
- $6 + 2 \times 7.0684 = 20.1367$ ms.

- la potenza è quindi:
- 1000 / 20.1367 = 49.66 blocchi/sec.

- Utilizzo CPU = 6 / 20.1367 = 29.80%

- Utilizzo disco = 2×7.0684 / 20.1367 = 100 - 29.80 = 70.20%

confronto delle prestazioni di sistemi diversi per un certo programma (metodo fenomenologico)

- X è più veloce di Y - (in particolare le prestazioni di X sono n volte quelle di Y)

$$\Rightarrow \frac{tempo\ di\ esecuzione\ Y}{tempo\ di\ esecuzione\ X} = n$$

$$tempo\ di\ esecuzione = \frac{1}{prestazione}$$

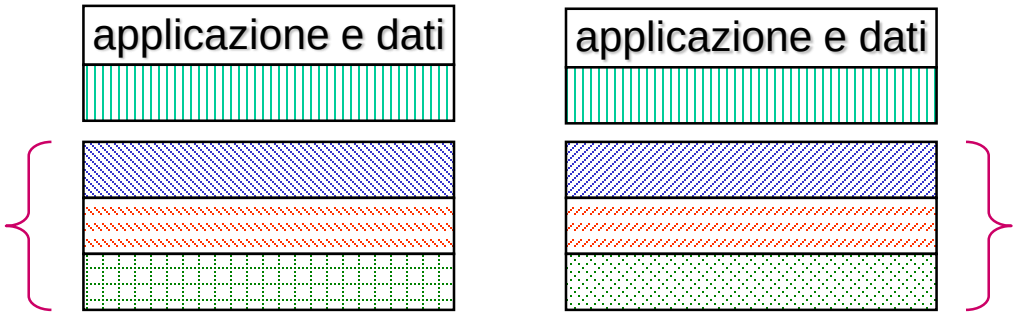
$$n = \frac{T_Y}{T_X} = \frac{\frac{1}{P_Y}}{\frac{1}{P_X}} = \frac{P_X}{P_Y}$$

problema del "programma campione"

tempo di esecuzione: include tutti i tempi di attività e di attesa

confronto delle prestazioni (cont.)

- il processore più veloce, *per una certa applicazione*, è quello che la esegue nel minore tempo di CPU;
- il confronto è a partire dal primo strato diverso verso il basso:
  - per esempio, si possono confrontare applicazioni *identiche* per l'utente, dal punto di vista della logica e dei dati, anche se eseguite su processori forniti di architetture diverse;
    naturalmente in questo modo si paragonano non solo i processori, ma anche le architetture in senso lato (sistemi operativi, applicazioni, linguaggi, codifica di programmazione, tuning, ...)



algebra di confronto per più applicazioni

tempi medi totali: $\frac{1}{n}\sum T_i$

programma i-esimo

$$\frac{P_X}{P_Y} = \frac{T_Y}{T_X} = \frac{\sum T_i(Y)}{\sum T_i(X)} = \frac{\sum \frac{1}{P_i(Y)}}{\sum \frac{1}{P_i(X)}} = \frac{\frac{n}{\sum \frac{1}{P_i(X)}}}{\frac{n}{\sum \frac{1}{P_i(Y)}}}$$

- il rapporto fra le prestazioni medie di due macchine è il rapporto fra le medie armoniche delle prestazioni dei singoli programmi
- i pesi vanno determinati in base a considerazioni sulle frequenze dei carichi (nella formula per semplicità valgono *1/n* - vedi alla pagina seguente)

algebra di confronto (cont.)

	A	B	C
P1	1	10	20
P2	1000	100	20
somma	1001	110	40

Tempi di esecuzione di 2 programmi (P1, P2) su tre macchine (A, B, C)

- rapporti di potenza cioè rapporti inversi dei tempi:

  A = 10 B (P1)
  B = 10 A (P2)

  A = 20 C (P1)
  C = 50 A (P2)

  B = 2 C (P1)
  C = 5 B (P2)

- i rapporti sono associativi (per lo stesso carico)

  C/A = C/B B/A

  B = 9.1 A (P1 + P2)

  C = 25 A (P1 + P2)

  C = 2.75 B (P1 + P2)

- i rapporti si conservano solo se si usano le medie geometriche:

  $MG(P1,P2)_B = \sqrt{(100 \times 10)} = 31.62$

  $MG(P1,P2)_C = 20$

  $MG(P1,P2)_B / MG(P1,P2)_C = 1.58$

  $MG((B/C)_1, (B/C)_2) =$

  $MG(0.5, 5) = 1.58$

  (questa proprietà ha giustificato talvolta l'uso della MG per valutare le prestazioni)

algebra di confronto (cont.)

rapporto delle potenze fra macchina X e macchina Y

medie pesate con pesi w_i

P_i : peso = importanza che assegnamo al carico (i)

$$\frac{\sum w_i \cdot T_i(Y)}{\sum w_i \cdot T_i(X)} = \frac{\dfrac{1}{\sum \dfrac{w_i}{P_i(X)}}}{\dfrac{1}{\sum \dfrac{w_i}{P_i(Y)}}}$$

w_i: importanza relativa del carico i

	A	B	C	w(1)	w(2)	w(3)
P1	1	10	20	0.5	0.909	0.999
P2	1000	100	20	0.5	0.091	0.001
w(1)	500.50	55.00	20.00			
w(2)	91.82	18.18	20.00			
w(3)	2.00	10.09	20.00			

la riga w(i) contiene le medie di P1 e P2 con i pesi della colonna w(i)

500.5/55

1- pesi uguali
2- pesi inversi alle durate della macchina B
3- pesi inversi alle durate della macchina A

	Rapporti di potenza		
	B/A	C/A	C/B
1	9.1	25	2.75
2	5.05	4.59	0.91
3	0.2	0.1	0.5

metriche di confronto fra macchine diverse

- un singolo numero non è sufficiente per confrontare macchine diverse, è importante conoscere quale *tipo di carico* è usato nel confronto

  dato un certo carico sia:
  - T : tempo di osservazione
  - B : tempo di CPU
  - C : unità_di_lavoro_completate nel tempo T

  - U = B/T: utilizzo

- ETR = C/T : (External Throughput Rate) – indice di potenza di *sistema*
- ITR = C/B : (Internal Throughput Rate) = ETR/U – indice di potenza di *CPU*
  - (valore di picco - si suppone cioè che non ci siano colli di bottiglia)
- i rapporti ETR(A) / ETR(B) e ITR(A) / ITR(B) sono metriche di confronto delle prestazioni delle macchine A e B per lo stesso tipo di carico
- ITR & ETR hanno dimensione di una potenza cioè [tempo] $^{-1}$

esempio numerico – un solo carico

(si misura lo stesso carico sulle macchine A e B)

	A	B	B/A
misure			
Tempo di esecuzione	720.54	720.32	
Tempo di processor	630.11	627.72	
Conteggio trans.	727736	1273150	
calcoli			
Utilizzo %	87.45	87.14	
ETR	1010.0	1767.5	1.75
ITR	1154.9	2028.3	1.76

- se il numero p di processori per macchina è > 1
  il valore ITR si ottiene ancora da ETR/U dove U è l'*utilizzo medio per processore*
- le condizioni di confronto devono essere paragonabili per quanto riguarda l'utilizzo

esempio numerico - più carichi

- La tabella contiene i valori ITR di 3 tipologie di carico (1, 2, 3) misurati su 4 macchine (A, B, C, D); l'ultima colonna riporta il valore medio (media armonica con uguali pesi)

	prestazioni interne X			
	tipo di carico			
macchina	1	2	3	media
A	0,02506	0,901	41,69	0,073103
B	0,04835	2,091	102,16	0,141706
C	0,05305	2,334	118,23	0,155545
D	0,06296	2,713	137,36	0,184513

- un ITR maggiore (minore) corrisponde a una macchina di maggiore (minore) potenza.
- il rapporto di potenza fra due macchine dipende essenzialmente dal *tipo di carico*

esempio numerico - più carichi (cont.)

- nella tabella di esempio:
- 0.073103 = 3/(1/0.02506 + 1/0.901 + 1/41.69) e analoghi calcoli.
- i pesi dovrebbero essere scelti in base alle relative frequenze (o consumi di risorse) dei carichi. Se, per esempio, volessimo calcolare il valore medio di potenza nella situazione in cui i carichi *consumassero* la risorsa CPU rispettivamente per il 10, 20 e 60 percento, pesata su tali consumi allora:
- 9/(1/0.02506 + 2/0.901 + 6/41.69) = 0.2129

- *I valori ITR non hanno un significato assoluto (come si sarebbe voluto con i MIPS, Mega Istruzioni Per Secondo) ma relativo, cioè solo come confronto fra macchine diverse che elaborano la stessa tipologia carico*

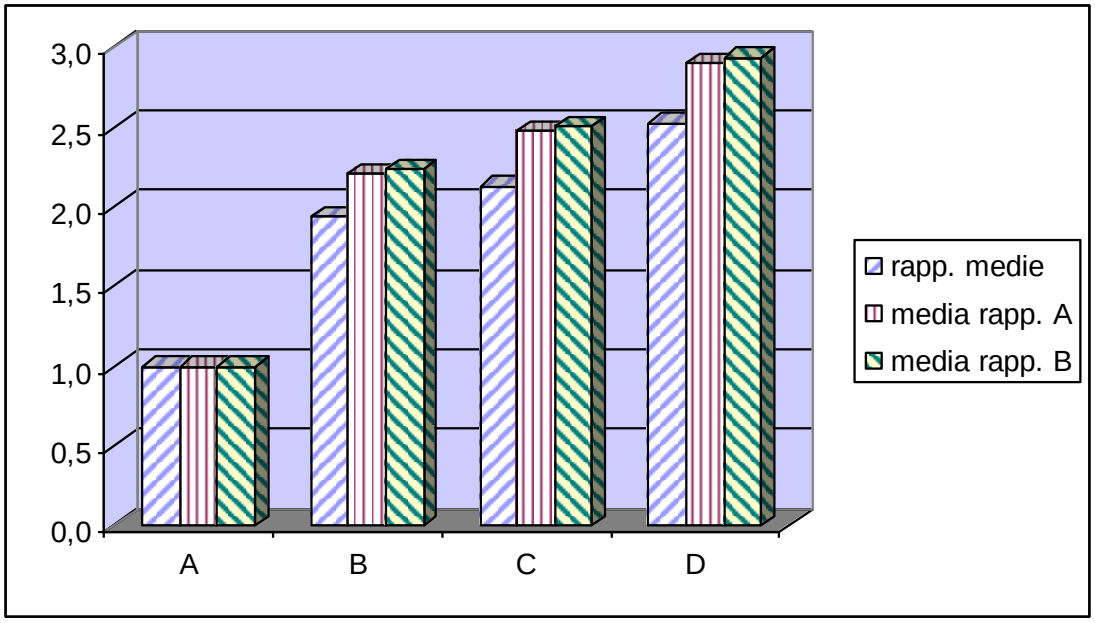
esempio numerico - più carichi (cont.)

	prestazioni interne X				rapporti prestazioni con base A			
	tipo di carico				tipo di carico			
macchina	1	2	3	media	1	2	3	media
A	0,02506	0,901	41,69	0,07310	1,00	1,00	1,00	1,00
B	0,04835	2,091	102,16	0,14171	1,93	2,32	2,45	1,94
C	0,05305	2,334	118,23	0,15554	2,12	2,59	2,84	2,13
D	0,06296	2,713	137,36	0,18451	2,51	3,01	3,29	2,52

questa colonna si ottiene da quest'altra

- i rapporti di potenza sono ottenuti dai valori della tabella delle prestazioni interne (ITR) divisi per quelli corrispondenti della riga A considerata come *base*
- attenzione!
  - *il rapporto delle medie NON è uguale alla media dei rapporti*

esempio numerico - più carichi (cont.)



- le colonne a sinistra sono quelle corrette
  - si può osservare che la media dei rapporti dipende anche dalla macchina presa come base (colonne 2 e 3)

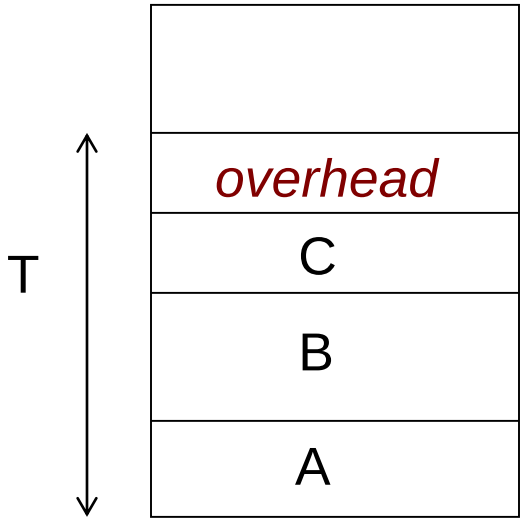
esempio numerico - più carichi (cont.)

- Conclusione
- non si possono utilizzare i soli rapporti (normalizzati e adimensionali) ma bisogna:
  1. partire dai valori effettivi di potenza (ITR), eventualmente ricalcolati da quelli di una macchina base e dai rapporti con quella che ci interessa;
  2. costruire le medie armoniche pesate;
  3. ottenere i rapporti di queste (colonne “carico medio con base X”)

- solo questi ultimi sono *corretti e coerenti*

overhead e caratterizzazione del carico

- non tutto l'*overhead* viene assegnato in modo automatico a un *workload*. Una parte non addebitata è per esempio il primo livello di gestione delle interruzioni di I/O. Un maggior numero di processori produce una maggiore "rate" di eventi di sincronizzazione)
- nasce perciò il problema di come ripartirlo fra i carichi se questi sono più di uno. Il non catturato è di solito intorno al 10%. (*capture ratio* (C.R.) = 90%)
- il totale è misurato correttamente ma i parziali lo sono per difetto



$$A + B + C < T$$

$$\frac{A}{CR_A} + \frac{B}{CR_B} + \frac{C}{CR_C} = \frac{A+B+C}{CR} = T$$

overhead e caratterizzazione del carico (cont.)

- noti A, B, C, T, CR bisogna determinare CR_A, CR_B, CR_C
  - 1) $CR_A = CR_B = CR_C = CR$
  - 2) $1/CR_A$, $1/CR_B$, $1/CR_C$ vengono calcolati dalla regressione lineare attraverso numerose misure o misure isolate di un solo carico alla volta. Questi metodi sono generalmente insoddisfacenti e gli errori non sono trascurabili anche perché il CR non dipende solo dal tipo di carico ma da altre variabili incontrollabili.
  - 3) Si cerca di legare le quantità *non catturate* a una caratteristica del carico, per esempio al numero di *interrupt* allora:
  - $(1-CR_A)A = KI_A$; $(1-CR_B)B = KI_B$; $(1-CR_C)C = KI_C$;
  - $(1-CR)(A+B+C) = K(I_A+I_B+I_C)$; (K = costante)
- (tutti questi metodi sono solo approssimazioni)

Legge di Amdahl

(verrà ripresa in modo più dettagliato nel “calcolo parallelo”)

speedup

- un principio generale della progettazione è quello di favorire gli stati che si verificano più frequentemente (rendere più veloce il caso più comune)
- il confronto fra la situazione variata e quella iniziale si misura mediante una grandezza adimensionale detta *speedup* e così definita:

- Speedup

  = prestazione_finale / prestazione_iniziale

  = tempo_iniziale / tempo_finale

  $s = P1 / P_0 = T_0 / T_1$

Legge di Amdahl (1967)

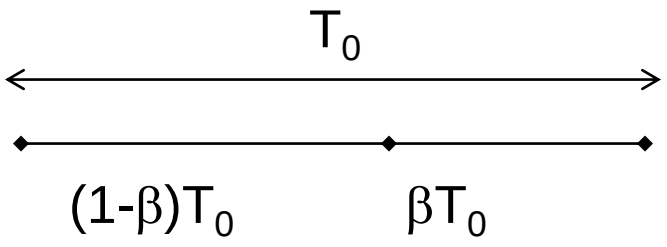
- la legge di Amdahl riguarda il guadagno di prestazioni che può essere ottenuto riducendo la durata di elaborazione di una porzione dell'intero processo

- in particolare la legge è espressa dallo *speedup* che è funzione di:

  1. quanto più veloce è il nuovo modo di operare rispetto a quello iniziale, rappresentato dal parametro **k**. (k > 1)

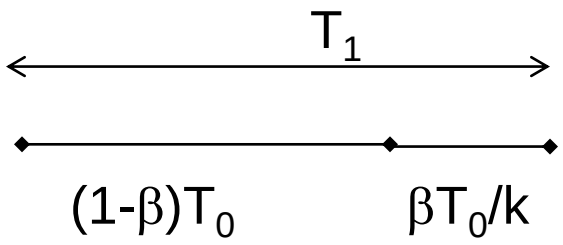
  2. della *frazione* del processo a cui si applica tale modalità, data dal parametro **β**. ($\beta \leq 1$)

Legge di Amdahl (cont.)



β = frazione del tempo che può essere modificata

k = fattore di modifica



speedup

$s = T_0 / T_1 = 1/((1-\beta) + \beta/k)$

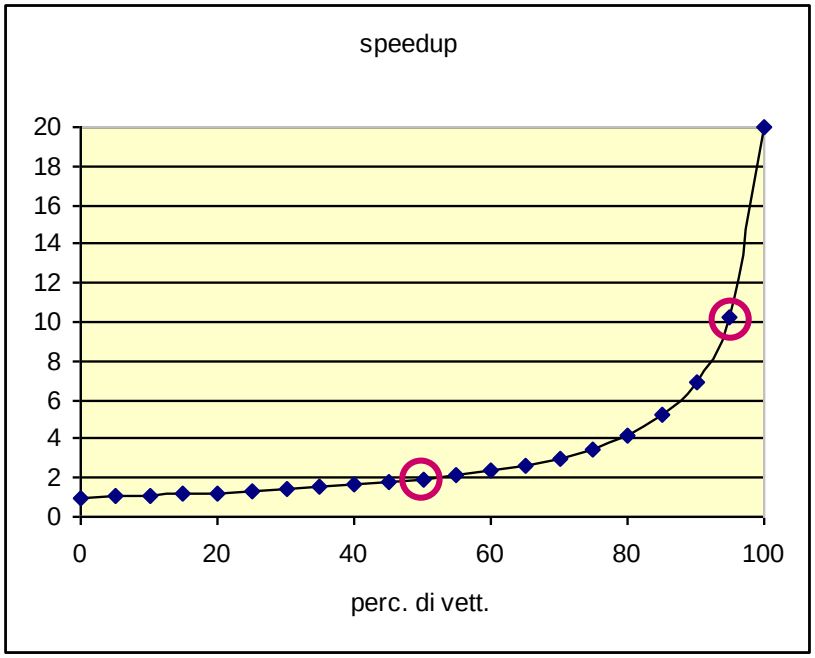
massimo raggiungibile: $s \rightarrow 1/(1-\beta)$
(per $k \rightarrow \infty$)

massimo raggiungibile: $s \rightarrow k$
(per $\beta \rightarrow 1$)

un esempio

- un impianto può essere perfezionato introducendo la modalità di lavoro vettoriale, nella quale il calcolo è *20* volte più veloce che in modo scalare. Si chiede:
  - la percentuale di vettorizzazione che un programma deve possedere perché lo speedup sia pari a 2
  - la percentuale necessaria per raggiungere la metà del massimo speedup possibile
- si è raggiunta una percentuale di vettorizzazione del *70%*
  - con un notevole investimento addizionale si può raddoppiare la velocità vettoriale ottenendo un certo speedup;
  - potendo migliorare ancora tale percentuale, quale è la percentuale che produce lo stesso speedup senza modificare la velocità vettoriale?

un esempio (cont.)



s = s(β) per k = 20

$$\beta = k \cdot \frac{s-1}{s \cdot (k-1)}$$

s = 2, β = 52.63%
s = 1/2 smax = 10, β = 94.74%

β = 70%, k = 20, s = 2.98
β = 70%, k = 40, s = 3.15

β = 71.84%, k = 20, s = 3.15

Il guadagno s diventa significativo solo per valori elevati di β

esempio: effetto della gerarchia di memoria

- effetto della *cache*
- cache hit: 90% cache miss: 10%
- velocità cache: 10 x velocità memoria
- speedup: 5.3

$$S = \frac{1}{(1-0.9)+\frac{0.9}{10}} = \frac{1}{0.19} = 5.3$$

confronto delle prestazioni a livello “sistema”

- il confronto richiede:
  - scelta dei programmi e dei dati per la valutazione
    - programmi reali / kernel da essi estratti / programmi speciali…
  - collezione di programmi: *benchmark suite*
  - principio guida: *riproducibilità*
- all’*utente* interessano in primo luogo i *tempi di risposta*

speedup indotto dalla sostituzione della CPU

- la durata di un lavoro non dipende linearmente dall'inverso della potenza, infatti:
- un carico di lavoro ha durata *x* eseguito su un certo sistema con utilizzo *u(x)*
- portato su altro sistema, la cui CPU ha potenza *n* volte la prima, avrà durata *x'* con utilizzo *u'(x')*
- x = x (1- u(x) + u(x))
- x' = x (1- u(x) + u(x)/n)
- per esempio:
  - u(x) = 52.44%; *n* = 2
  - s = x/x' = 1.355 (incremento di produttività < 2)
  - x·u(x) $\Rightarrow$ x'·u'(x') = x·u(x) / *n* (tempi di CPU busy)
  - u'(x') = s·u(x) / *n* = 35.52% (nuovo utilizzo)