


Valutazione delle prestazioni

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento Patterson: 1.6, 1.9, 1.10, 6.10


Sommario

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.

Le prestazioni migliorano perché:

- Incrementa la quantità di lavoro nell'unità di tempo (throughput, bandwidth). Quantità di dati elaborati nell'unità di tempo.
- Diminuisce il tempo di esecuzione (execution time, response time). Velocità di esecuzione.

Domande:

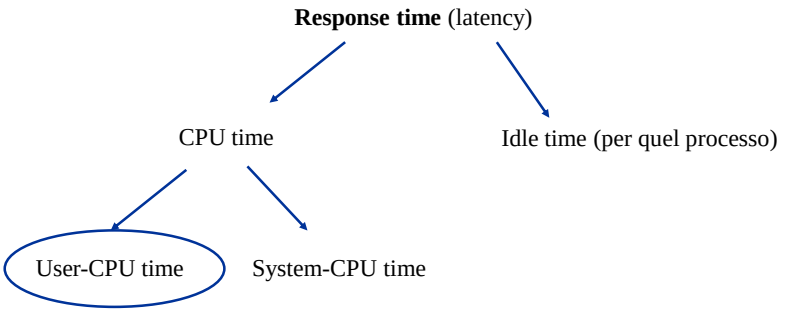
Un processore più veloce cosa influenza?
Più processori dedicati, cosa modificano?

Misura delle prestazioni (Fatturazione)

Compito non semplice per la stretta connessione tra HW, Software di sistema e Software Utente.

Latenza di un'operazione di I/O, di accesso a DRAM viene nascosta facendo altro.
Cosa valutiamo?

Chi valuta? Il progettista? L'utente finale?



Misura delle prestazioni (Progettista)

Execution time = Num_Cicli_Clock * T_{clock} = Num_Cicli_Clock / f_{clock}

CPI = Num_Cicli_Clock / Num_Istruzioni

IPC – Number of Instructions per Cycle

IPC = 1/ CPI

Execution time = (CPI * Num_Istruzioni) * T_{clock}

Equazione delle prestazioni di una CPU basata sul CPI

Execution time = (CPI * Num_Istruzioni) * T_{clock}

- Diminuzione del periodo di clock (aumento della frequenza)
- Diminuzione del numero di istruzioni
- Diminuire il numero di cicli di clock medi per ogni istruzione

Esempio - 1

Decisione tra due sequenze di istruzioni. Il progettista HW comunica che si identificano 3 cluster di istruzioni, dove ciascun cluster utilizza un numero di cicli di clock diverso:

- Cluster A: 1 ciclo di clock
- Cluster B: 2 cicli di clock
- Cluster C: 3 cicli di clock

Le due sequenze di istruzioni hanno un mix diverso delle istruzioni di tipo A, B e C.

Sequenza 1:
- Istruzioni di tipo A: 2
- Istruzioni di tipo B: 1
- Istruzioni di tipo C: 2

TOTALE: **5 istruzioni**

Sequenza 2:
- Istruzioni di tipo A: 4
- Istruzioni di tipo B: 1
- Istruzioni di tipo C: 1

TOTALE: **6 istruzioni**

Quale sequenza scegliere?

Esempio - 2

Decisione tra due sequenze di istruzioni. Il progettista HW comunica che si identificano 3 cluster di istruzioni, dove ciascun cluster utilizza un numero di cicli di clock diverso:

- Cluster A: 1 ciclo di clock
- Cluster B: 2 cicli di clock
- Cluster C: 3 cicli di clock

Le due sequenze di istruzioni hanno un mi diverso delle istruzioni di tipo A, B e C.

Sequenza 1:
- Istruzioni di tipo A: 2
- Istruzioni di tipo B: 1
- Istruzioni di tipo C: 2

TOTALE: **5 istruzioni**

Sequenza 2:
- Istruzioni di tipo A: 4
- Istruzioni di tipo B: 1
- Istruzioni di tipo C: 1

TOTALE: **6 istruzioni**

Quale sequenza scegliere?

T_{CPU_1} = 2*1 + 1*2 + 2*3 = 10 cicli clock
T_{CPU_2} = 4*1 + 1*2 + 1*3 = 9 cicli clock

CPI_1 = 10 cicli clock / 5 istruzioni = 2
CPI_2 = 9 cicli clock / 6 istruzioni = 1,5

Execution time = (CPI * Num_Istruzioni) / f_{clock}

CPI della sequenza 2 è molto minore e compensa l'aumento di un'istruzione a parità di clock.

Osservazioni

Execution time = (CPI * Num_Istruzioni) * T_{clock}

Execution time = (CPI * Num_Istruzioni) / f_{clock}

Cambiando l'insieme di istruzioni si può ottenere un cammino critico inferiore e quindi una frequenza di clock superiore -> un CPI inferiore (istruzioni più veloci), ma può aumentare il numero di istruzioni (e.g. togliendo l'istruzione movs dall'ISA Intel)

Occorre un compromesso.

Recap e osservazioni

Execution time = (CPI * Num_Istruzioni) / f_{clock}

Dato un programma Utente, possiamo calcolare il tempo di **risposta** (ma questo conterrà anche il tempo di esecuzione della parte di Sistema e il tempo in cui il processore è idle.

Alternativamente possiamo calcolare:

- f_{clock} -> dato dalle specifiche (in realtà esiste la possibilità di aumentare temporaneamente la frequenza, fino a un certo livello di calore. E.g. Core i-7, la frequenza aumenta del 10% - Turbo mode).
- Num_Istruzioni (in linguaggio macchina) -> dai tool di profilazione.
- CPI -> ?? Difficile da determinare.

CPI di un instruction mix

Execution time = (CPI * Num_Istruzioni) / f_{clock}

Se tutte le istruzioni fossero dello stesso tipo -> CPI = Num Cicli clock di ogni istruzione

$$CPI_{medio} = \frac{\sum_{i=1}^{n}(CPI_i * l_i)}{\sum_{i=1}^{n} l_i} = \frac{\sum_{i=1}^{n}\frac{(CPI_i * l_i)}{l_{TOT}}}{\frac{\sum_{i=1}^{n} l_i}{l_{TOT}}} = \sum_{i=1}^{n}(CPI_i * f_i)$$

Operazioni intere più veloci delle operazioni in virgola mobile
Operazioni di salto condizionato più lente
…..

Nelle architetture super-scalari, cluster di istruzioni richiedono tempi diversi.

Quindi. Quale mix di istruzioni?

Esempio - 1

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 (T = 2 ns).

CPI = 0,43 * 1 + 0,21 * 4 + 0,12 * 4 + 0,12 * 2 + 0,12 * 2= 2,23

T_{CPU} = **200** istruzioni * 2,23 Clock / istruzione = 446 cicli di clock => 446 * 2 ns = 892 ns

Esempio - 2

Si consideri un calcolatore in grado di eseguire le istruzioni riportate in tabella (memory intensive):

	Frequenza	cicli di clock
ALU	23%	1
Load	31%	4
Store	22%	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 (T = 2 ns).

CPI = 0,23 * 1 + 0,31 * 4 + 0,22 * 4 + 0,12 * 2 + 0,12 * 2= 2,83

T_{CPU} = **200** istruzioni * 2,83 Clock / istruzione = 566 cicli di clock => 566 * 2 ns = 1.132 ns

Misura del tempo di esecuzione

Tempo di esecuzione -> dipende dal mix di istruzioni

$$CPI_{medio} = \frac{\sum_{i=1}^{n}(CPI_i * l_i)}{\sum_{i=1}^{n} l_i} = \frac{\sum_{i=1}^{n}\frac{(CPI_i * l_i)}{l_{TOT}}}{\frac{\sum_{i=1}^{n} l_i}{l_{TOT}}} = \sum_{i=1}^{n}(CPI_i * f_i)$$

$$t_{medio} = CPI_{medio} * T_{clock}$$

	Frequenza	cicli di clock
ALU	23%	1
Load	31%	4
Store	22%	4
Branch	12%	2
Jump	12%	2

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

Quale CPI scelgo?

Misure alternative

Execution time = (CPI * Num_Istruzioni) / f_{clock}

MIPS: milioni di istruzioni al secondo
MFLOPS: milioni di operazioni in virgola mobile al secondo

Problemi con il MIPS:

- Dipende dall'insieme di istruzioni, quindi è difficile confrontare computer con diverse ISA;
- 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 (dipende dall'architettura).
- Varia a seconda del programma considerato;
- Può variare in modo inversamente proporzionale alle prestazioni!
- V**alore di picco**, scelgo il mix di istruzioni per massimizzare il MIPS misurato (fuorviante).

Esempio di trabocchetto con i MIPS di picco

Macchina con hardware opzionale per virgola mobile (co-processore).

Le istruzioni in virgola mobile **sono più lente**: richiedono **più cicli di clock** rispetto a quelle che lavorano con interi,

➔ 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** (eseguono meno istruzioni!).

➔ 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 tempo di esecuzione **più elevato**!!

Problemi con MIPS di picco

Intel i860 (1989) dichiarava:
- 2 operazioni VM al secondo
- Clock di 50 Mhz

Prestazioni attese di 100 MFlops

MIPS R3000 (1989) dichiarava:
- 16 MFlops
- Clock a 33 Mhz

Su problemi reali l'i860 risultò 12% più lento del MIPS R3000!

Intel i860 dichiarava i MFlops di picco, difficilmente raggiungibili e sostenibili.

Legge di Amdhal: Come rendere più veloce un elaboratore

Execution time = (CPI * Num_Istruzioni) * T_{clock}

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 viene eseguito.

Esempio: Pentium e PentiumPro: a fronte di un raddoppio della frequenza di clock che è passata da 100 a 200 Mhz, si è registrato un aumento delle prestazioni misurate tramite SpecInt di 1,7 volte e di 1,4 volte misurate in SpecFloat. Come mai? Perché alcune istruzioni hanno richiesto più cicli di clock.

Metrica: speed-up: $T_{old} / T_{new} = V_{new} / V_{old}$

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 uno1 meno tale frazione: $Speedup_{globale} < 1/(1-F_m)$.

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. Divido T_{old} in tempo migliorato e tempo non migliorato:

$T_m = F_m * T_{old}$
$T_{nm} = (1- F_m) * T_{old}$

2. **$Speedup_{migliorato}$** ($S_m \geq 1$), ovvero il miglioramento ottenuto dal modo di esecuzione più veloce (T_{old}/T_{new}). Si applica solo al period di tempo T_m: $T_{old} = T_m + T_{nm}$

3. **Tempo migliorato**: $T_{new} = (F_m * T_{old}) / S_m + (1 – F_m) * T_{old}$

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

Per $S_m \to \infty$ $T_{old} / T_{new} = 1/ (1 – F_m)$

Esempio numerico

Somma di 10 variabili scalari e somma di una coppia di matrici bidimensionali 12x10

Supponiamo che solo la somma di matrici sia parallelizzabile e che abbiamo a disposizione 40 processori su cui parallelizzare. Ogni operazione di somma costa un tempo *t*.

Qual è lo speed-up?

Il tempo senza parallelizzazione sarà: $10t + (12x10)t = 130\ t$

Il tempo dopo la parallelizzazione sarà: $10t + ((12x10) / 40)t = 13\ t$

Lo speed-up sarà quindi: $130\ t\ /\ 13\ t\ = 10$

La velocità aumenta di 10 volte e non di 40 come ci si poteva aspettare.

Ci sono delle parti di codice non parallelizzate (somma di scalari) che limitano il guadagno.

Stesso risultato con Amdhal: $Speedup_{globale} = 1 / (Fm / Sm + (1 – Fm)) =$

$1 / (120/130 / 40 + 10/130) = 520 / 5200 = 10$

Frazione di tempo: $F_m = 120/130$ $F_{nm} = 10/130$

Esempio numerico - 2

Somma di 1000 variabili scalari e somma di una coppia di matrici bidimensionali 12x10

Supponiamo che solo la somma di matrici sia parallelizzabile e che abbiamo a disposizione 40 processori su cui parallelizzare. Ogni operazione di somma costa un tempo *t*.

Qual è lo speed-up?

Il tempo senza parallelizzazione sarà: $1000t + (12x10)t = 1120\ t$

Il tempo dopo la parallelizzazione sarà: $1000t + ((12x10) / 40)t = 1003\ t$

Lo speed-up sarà quindi: $1120\ t\ /\ 1003\ t\ = 1,116$

La velocità aumenta di poco più di 1 volta e non di 40 come ci si poteva aspettare -> non è stata una scelta illuminata.

Ci sono delle parti di codice non parallelizzate (somma di scalari) che limitano il guadagno.

Sommario

Valutazione delle prestazioni

Benchmark

Il modello Roof-line

Esercizi

I benchmark

MIPS / MFLOPS di picco poco significativi. Sono metriche utilizzate dai progettisti per valutare il tempo di esecuzione.

CPI picco sono poco significativi.

Cosa può utilizzare l'Utente?

Benchmarks = Programmi per valutare le prestazioni.

Benchmarks: Whetstone, 1976; Drystone, 1984.

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

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

Evaluating Architecture performances

1.8 GHz

Descrizione	Nome	Numero di istruzioni × 10^9	CPI	Periodo di clock (secondi × 10^{-9})	Tempo di esecuzione (secondi)	Tempo di riferimento (secondi)	SPECratio
Interprete Perl	perlbench	2684	0,42	0,556	627	1774	2,83
Compilatore GNU C	gcc	2322	0,67	0,556	863	3976	4,61
Ottimizzazione combinatoria	mcf	1786	1,22	0,556	1215	4721	3,89
Libreria di simulazione di eventi discreti	omnetpp	1107	0,82	0,556	507	1630	3,21
Conversione da XML a HTML via XSLT	xalancbmk	1314	0,75	0,556	549	1417	3,21
Compressione video	x264	4488	0,32	0,556	813	1763	2,17
Intelligenza artificiale: ricerca su albero alpha-beta (scacchi)	deepsjeng	2216	0,57	0,556	698	1432	2,05
Intelligenza artificiale: ricerca ad albero Monte Carlo (Go)	leela	2236	0,79	0,556	987	1703	1,73
Intelligenza artificiale: generatore di soluzioni ricorsive (Sudoku)	exchange2	6683	0,46	0,556	1718	2939	1,71
Compressione generale di dati	xz	8533	1,32	0,556	6290	6182	0,98
Media geometrica							2,36

SPEC integer benchmarks – 2017 su Intel Xeon E5-2650L
SPEC (System Performance Evaluation Cooperative)

Indici SPEC (‘89, ‘92, ’95, ’00, ‘06, ‘16)

http://www.spec.org/. **The Standard Performance Evaluation Corporation (SPEC)** is a non-profit corporation formed to establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers. SPEC develops benchmark suites and also reviews and publishes submitted results from our member organizations and other benchmark licensees.

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.

SPEC INT 2000

Benchmark	Language	Category	Full Descriptions
164.gzip	C	Compression	HTML Text
175.vpr	C	FPGA Circuit Placement and Routing	HTML Text
176.gcc	C	C Programming Language Compiler	HTML Text
181.mcf	C	Combinatorial Optimization	HTML Text
186.crafty	C	Game Playing: Chess	HTML Text
197.parser	C	Word Processing	HTML Text
252.eon	C++	Computer Visualization	HTML Text
253.perlbmk	C	PERL Programming Language	HTML Text
254.gap	C	Group Theory, Interpreter	HTML Text
255.vortex	C	Object-oriented Database	HTML Text
256.bzip2	C	Compression	HTML Text
300.twolf	C	Place and Route Simulator	HTML Text

Modalità di incremento delle prestazioni su architetture parallele

Weak scaling: Il tempo di esecuzione rimane lo stesso, La dimensione dei dati e programma (working set), cioè (la dimensione del problema, cresce con il numero di nodi di elaborazione

Strong scaling: la dimensione del programma e dei dati è fissa e le prestazioni aumentano linearmente con il numero di processori, senza variare le dimensioni del problema.

Qual è più facile da ottenere?

Che tipo di aumento abbiamo ottenuto nell'esempio della somma degli elementi di un vettore parallelizzata su P procesori?

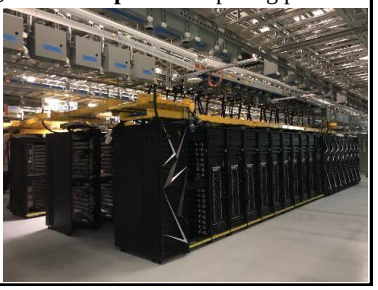
Descrizione dei benchmark

Linkpack. Programmi di **algebra lineare**. **Weak scaling.** DGEMM è il nucleo (e quello che costa di più intermini computazionali di questi programi). Linkpack determina il ***più veloce calcolatore al mondo (1.000,000 di miliardi di flops!)***.

https://www.top500.org/

Ultima competizione nel Maggio 2022

The No. 1 spot is now held by the **Frontier system at Oak Ridge National Laboratory (ORNL)** in the US. Based on the latest **HPE Cray EX235a** architecture and equipped with **AMD EPYC 64C 2GHz processors,** the system has **8,730,112 total cores, a power efficiency rating of 52.23 gigaflops/watt,** and relies on gigabit ethernet for data transfer**. 1,102 Exaflop/s** of computing power.



Summit (2019):

- 2,414,592 core
- 2,801,664 GB di Memoria Principale (>2 PetaByte)
- Potenza: 10,096.00 kW

Parallel SPEC

Anche valutazione dei cloud:
SPEC Cloud™ IaaS 2016

Anche Java server benchmark:
SPECjbb2015

Benchmark	Tipo di scaling	Riprogram-mazione	Descrizione
Linpack	Debole	Si	Algebra lineare con matrici dense (Dongarra, 1979)
SPECrate	Debole	No	Parallelismo di programmi indipendenti (Henning, 2007)
SPLASH 2, Stanford Parallel Applications for Shared Memory [Applicazioni parallele di Stanford per memoria condivisa] (Whoo et al., 1995)	Forte (anche se offre solo due dimensioni di problemi)	No	FFT 1D complessa Decomposizione LU a blocchi Fattorizzazione di Cholesky di matrici sparse a blocchi Ordinamento *Radix Sort* di interi *Barnes-Hut* Multipolo veloce adattativo Simulazioni oceaniche Radiosità gerarchica *Ray tracing* Rendering di volumi Simulazione di acqua con strutture dati spaziali Simulazione di acqua senza strutture dati spaziali
Benchmark paralleli NAS (Bailey et al., 1991)	Debole	Si (solamente C o Fortran)	EP: applicazioni «embarrassingly parallel».[1] MG: *Simplified Multigrid* (griglie multiple semplificate) CG: griglie non strutturate per il calcolo del gradiente coniugato FT: soluzione delle equazioni differenziali alle derivate parziali in 3D, utilizzando la FFT. IS: *Large integer sort*, ordinamento di un vettore di interi di grande dimensione.
Raccolta di benchmark PARSEC (Bienia et al., 2008)	Debole	No	*Blackscholes* – assegnamento dei prezzi utilizzando le equazioni differenziali alle derivate parziali Black-Scholes *Bodytrack* – tracking del corpo di una persona *Canneal* – versione dell'algoritmo di *simulated annealing* per l'ottimizzazione dell'instradamento su rete dei pacchetti, che tiene conto della presenza della cache. *Dedup* – algoritmo di compressione di ultima generazione, basato su deduplicazione dei dati *Facesim* – Simulazione della mimica di un volto umano *Ferret* – server di ricerca per similarità di contenuto *Fluidanimate* – fluidodinamica per l'animazione con il metodo SPH *Freqmine* – ricerca degli insiemi di elementi frequenti *Streamcluster* – clustering online di dati di input in stream *Swaptions* – assegnamento di un prezzo agli swap di un portafoglio titoli *Vips* – elaborazione delle immagini *X264* – codifica video H.264
Moduli algoritmici di Berkeley (Asanovic et al., 2006)	Forte e debole	Si	Macchine a stati finiti Logica combinatoria Attraversamento di grafi Griglie strutturate Matrici dense Matrici sparse Metodi spettrali (FFT) Programmazione dinamica N-corpi *MapReduce* *Backtrack/Branch and bound* Inferenza mediante modelli a grafi Griglie non strutturate

Descrizione dei benchmark paralleli

SPECrate. **Weak scaling.** Independent job parallelism. Vengono eseguite più copie di uno stesso programma.

Stanford parallel application for shared memory. **Strong scaling.** Problemi diversi simili agli SPEC CPU: FFT, LU decomposition. Fattorizzazione di matrici sparse. Ordinamento. Ray tracking. Volume rendering….

NAS parallel benchmarking. **Weak scaling.** Disegnati per la fluido-dinamica. Problemi multi-grid. Large integer sort. Equazioni differenziali parziali 3D risolte con FFT.

PARSEC. **Weak scaling.** Utilizzano i Pthread (POSIX threads) e OPENMP. Propongono applicazioni di frontiera. Tracking di persone (da video). Routing ottimizzato. Compressione dei dati. Mimica facciale. Ricerca di contenuti simili in un server. Fluido-dinamica. Image processing. Video encoding.

Berkeley design pattenrs. **Weak scaling.** Macchine a Stati Finiti. Logica combinatoria. Attraversamento di grafi. Matrici dense e sparse. Metodi di analisi spettrale (FFT). Programmazione dinamica. Problemi N-body. Ottimizzazione.

Sommario

Valutazione delle prestazioni

Benchmark

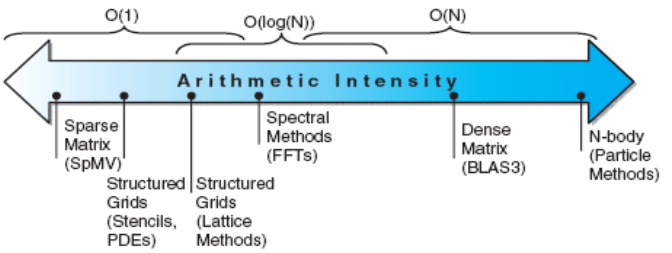
Il modello Roof-line

Esercizi

Arithmetic intensity

Quante operazioni facciamo per ogni Byte trasferito dalla memoria?

Obiettivo nella parellelizzazione è riutilizzare il più possibile i dati presenti in cache per evitare le miss penalty (e.g. blocchettizzazione dei dati).

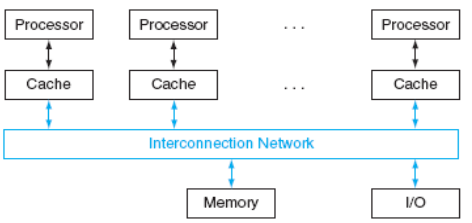


weak scaling, less memory requests per byte per task con intensità maggiore

Ruolo dell'intensità aritmetica

Velocità di calcolo di picco: riempio al massimo tutti i cori (tutti gli issue)

Velocità di trasferimento dalla gerarchia di memoria.



Se effettuiamo N_{op} >> 1 operazioni su ogni byte letto dalla memoria, avremo la velocità di calcolo massima.

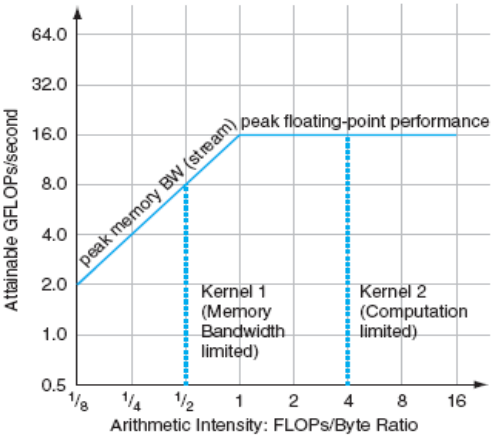
Il modello “roofline”

Nessun benchmark può essere contenuto interamente in cache.

2 elements:
- Computation
- Memory transfer

Per programmi con bassa intensità aritmetica (elevati accessi alla memoria per dato), il limite è offerto dal sistema di memoria.

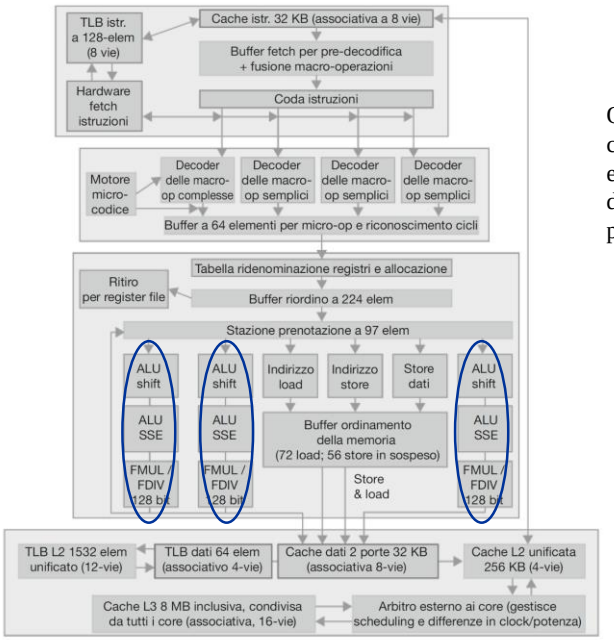
Per programmi ad alta intensità, il limite è dato dalla capacità di elaborazione della CPU.



AMD Opteron X2

Memory speed = 16GByte/s

Il modello “roofline”: Computation



Ogni core, a ogni ciclo di clock deve riuscire ad eseguire il massimo numero di istruzioni FP previste, 3 per un Core i7

Il modello “roofline”: Computation

Se non ci fossero problemi con la memoria le prestazioni sarebbero una linea orizzontale pari alla massima velocità di calcolo V_{calc}:

V_{calc} = P * V_{core} con P numero di core

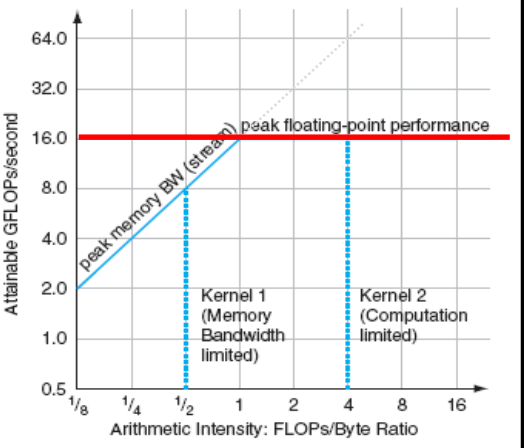
V_{calc} = costante = 16 Gflops per Opteron X2

Tutto quello che viene letto dalla memoria può essere elaborato senza stalli.

Le prestazioni non dipendono dall’intensità aritmetica. Ma solo da P e V_{core}.

V_{core} = $N_{cammini\_FP}$ * N_{Dati} / cammino / sec

Il limite coputazionale si può calcolare dalle caratteristiche strutturali della CPU.



AMD Opteron X2

Il modello “roofline”: Memory transfer

La memoria rifornisce la CPU.

I dati vengono letti dalla memoria con una certa velocità massima: V_{mem} = [Byte]/[s].

Le prestazioni di memoria si valutano con un benchmark particolare: streaming benchmark.

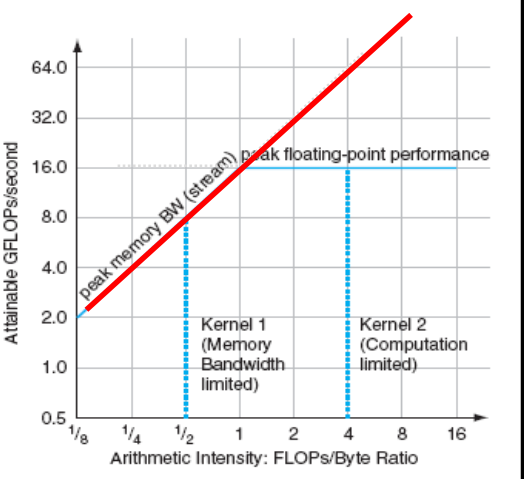
Tutti i dati letti dalla memoria vengono elaborati: per ogni byte effettuo un numero di operazioni N_{flop} definite dall’intensità aritmetica.

Maggiore è l’intensità, maggiore la velocità di calcolo.

Come si inserisce nel grafico questo vincolo?

All’aumentare della velocità del sistema di memoria, la retta trasla verso sx.

Il sottosistema di memoria associato all’AMD Opteron X2 ha una velocità di trasferimento di picco di 16GByte/s.



AMD Opteron X2

Il modello “roofline”: calcolo della pendenza

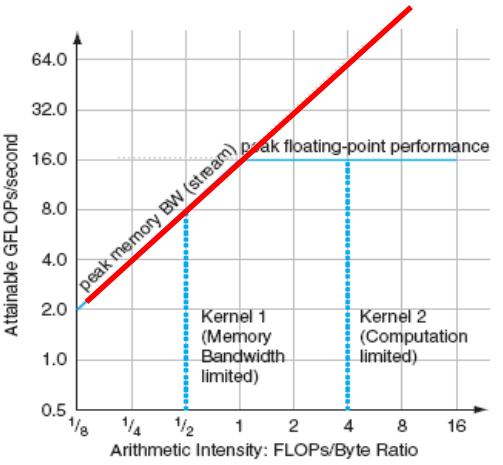
Il sottosistema di memoria associato all’AMD Opteron X2 ha una velocità di trasferimento di picco di 16GByte/s.

Intensità aritmetica 1 => 16 Gflops (1 Byte letto dalla memoria -> 1 Flops)
Intensità aritmetica 0.5 => 8 Gflops
Intensità aritmetica 0.25 => 4 Gflops

Intensità aritmetica 2 => 32 GFLops

….

Non si può superare la velocità massima offerta di calcolo della CPU



AMD Opteron X2

Il modello “roofline”: riassunto

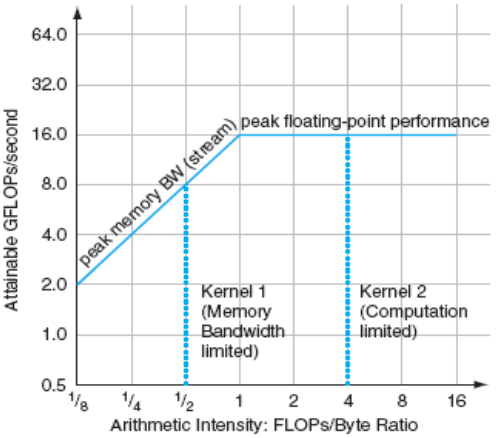
Nessun benchmark può essere contenuto interamente in cache.

AMD Opteron X2

2 elements:
- Computation
- Memory transfer

Per programmi con bassa intensità aritmetica (elevati accessi alla memoria per dato), il limite è offerto dal sistema di memoria.

Per programmi ad alta intensità, il limite è dato dalla capacità di elaborazione della CPU.



Attainable GFLOPs/sec =
Min {Peak Memory BW × Arithmetic Intensity, Peak Floating-Point Performance}

Dipende dall’instruction mix, ovverosia dal tipo di kernel.

Dall'Opteron X2 all'Opteron X4

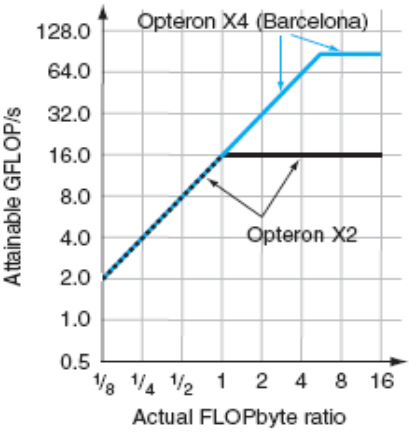
Opteron X4 vs Opteron X2:

•Stesso sistema di memoria (16 Gbyte/s)

•Numero doppio di processori (core)

•Numero quadruplo di operazioni in virgola mobile al secondo

- Doppia capacità aritmetica della pipeline
- Doppia capacità di fetch.



La velocità di elaborazione aumenta, ma solo per intensità aritmetiche superiori ad 1 quando si possono sfruttare maggiormente i dati già presenti in cache.

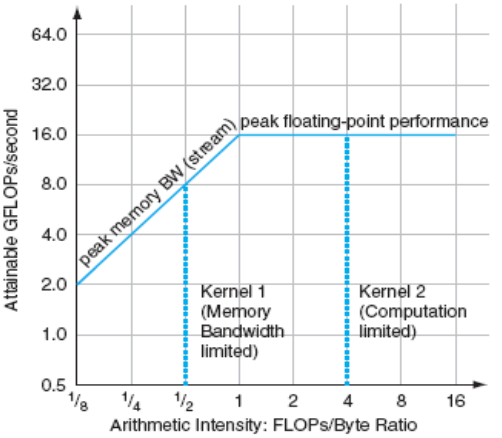
La velcocità di calcolo massima di 80 Gflops si raggiunge solo per un'intensità aritmetica pari a 5.

Roof model e ottimizzazioni

Cosa succede se le prestazioni del vostro programma risultano scadenti rispetto alle prestazioni attese?

Riempire meglio le pipeline.

Caricare meglio I dati in memoria.



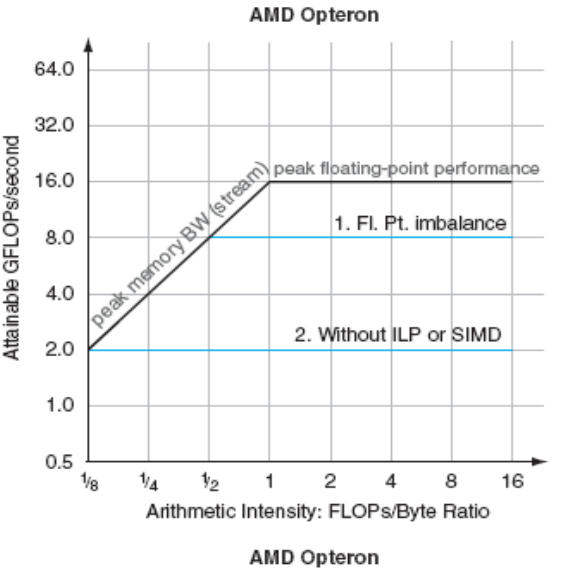
Ottimizzazioni sulla parte di calcolo - 1

Migliorare il mix di operazioni:

- Significativa percentuale di operazioni floating point

- Bilanciamento tra Moltiplicazioni e addizioni (e.g. fused add-multiplications, or equal numeber of execution paths).

-With imbalance, peak at 8 Gflops.



Ottimizzazioni sulla parte di calcolo - 2

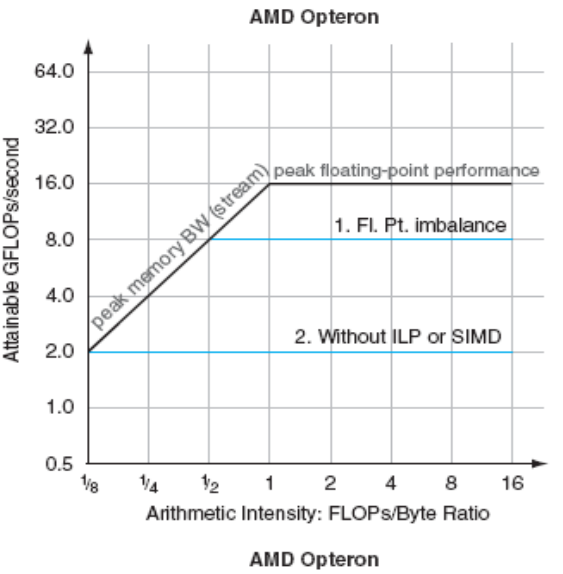
Aumentare la parallelizzazione dell'esecuzione (ILP): 3-4 istruzioni per ciclo di clock.

- Srotolamento dei cicli

- Utilizzo delle istruzioni vettoriali (AVX)

- Ottimizzazione del mix delle istruzioni.

Without ILP or SIMD adequantely exploited, 2 GFlops.



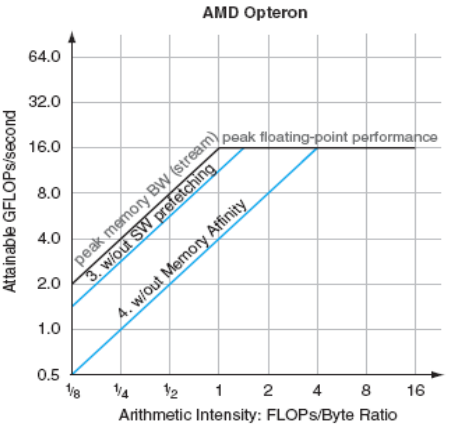
Ottimizzazioni sulla parte di memoria - 3

Software pre-fetching:

- Precaricamento dei dati in cache (speculazione efficace).

- Precaricamento delle istruzioni in cache (predizione dei salti)

Spostamento verso destra della curva della memoria: si reduce a velocità di trasferimento tra Memoria Principale e Cache



Ottimizzazioni sulla parte di memoria - 4

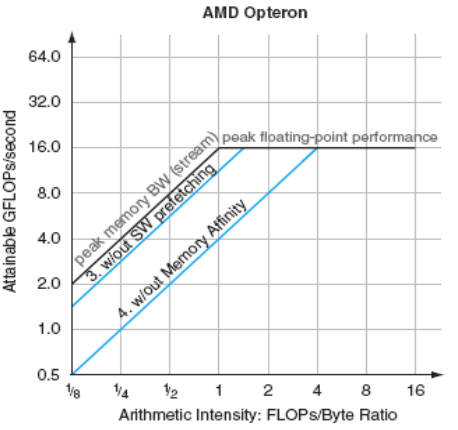
Affinità della memoria:

- Massimizzare gli hit.

- Separare il codice nei diversi core in modo che gli accessi in memoria siano all'interno della cache associata.

- Minimizzazione degli «invalidate».

Spostamento verso destra della curva della memoria: si reduce a velocità di trasferimento tra Memoria Principale e Cache.



Quali ottimizzazioni?

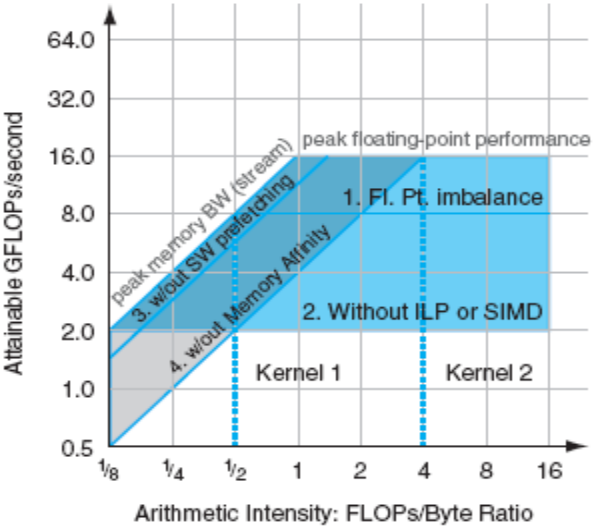
Occorre migliorare il codice perchè venga raggiunto il «tetto». Massima velocità di calcolo raggiungibile.

La distanza dal tetto indica quando si guadagna.

Ottimizzazioni eseguite in sequenza facilitano il compito.

Kernel 2 non ha problemi di memoria -> ottimizzazioni 1 e 2.

Kernel 1 -> ottimizzazioni 1, 2, 3 e 4.



Sommario

Valutazione delle prestazioni

Benchmark

Il modello Roof-line

Esercizi

Legge di Amdhal: Come rendere più veloci i calcolatori

Execution time = (CPI * Num_Istruzioni) * T_{clock}

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 viene eseguito.

Metrica: speed-up: $T_{old} / T_{new} = V_{new} / V_{old}$

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

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

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 uno1 meno tale frazione: $Speedup_{globale} < 1/(1-F_m)$.

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. Divido T_{old} in tempo migliorato e tempo non migliorato:

$T_m = F_m * T_{old}$
$T_{nm} = (1- F_m) * T_{old}$

2. **$Speedup_{migliorato}$** ($S_m \geq 1$), ovvero il miglioramento ottenuto dal modo di esecuzione più veloce (T_{old}/T_{new}). Si applica solo al period ti tempo T_m: $T_{old} = T_m + T_{nm}$

3. **Tempo migliorato**: $T_{new} = (F_m * T_{old}) / S_m + (1 - F_m) * T_{old}$

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

Per $S_m \to \infty$ $\quad T_{old} / T_{new} = 1/ (1 - F_m)$

Speed-up 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?

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

$Frazione_{migliorato}$ = 0.4

$Speedup_{migliorato}$ = 10

$Speedup_{globale}$ = $1/ [1-0,4+ 0,6 / 10)]$ = **1.56**

Speed-up 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?

$Speedup_{globale} = 1/ [1-0,5+ 0,5 / 5)] = 1.67$
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.

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.

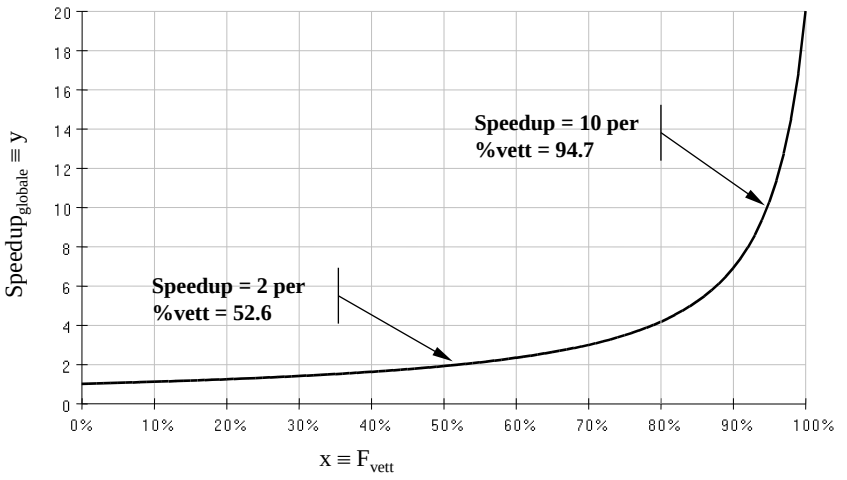
• Disegnare un grafico che riporti lo speedup come percentuale della computazione effettuata in modo vettoriale.
• Quale percentuale di vettorizzazione è necessaria per uno speedup di 2?
• Quale per raggiungere la metà dello speedup massimo?

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%

Sommario

Valutazione delle prestazioni

Benchmark

Il modello roof-line