


I multi processori

Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
borghese@dsi.unimi.it

Università degli Studi di Milano

Patterson, sezione 1.5, 1.6, 2.17, 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.9, 7.10

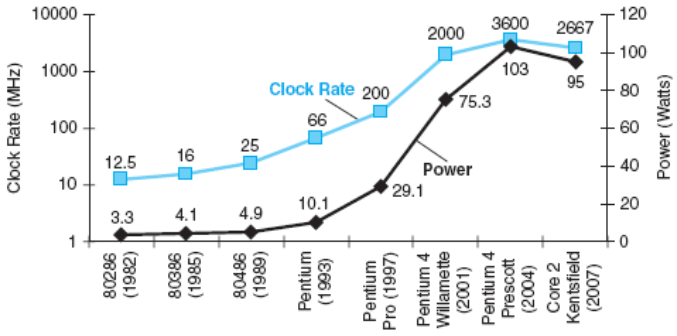

Sommario

Le architetture multi-processore

I cluster

Il modello "Roofline"

La barriera dell'energia

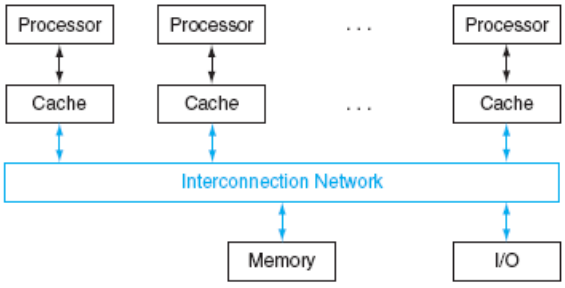


Power = Capacitive load × Voltage^2 × Frequency switched

stato delle porte CMOS

È aumentata di 1000 volte in 20 anni

I multi-processori

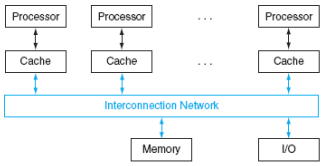


Chiama un parallelismo esplicito (la pipe-line multi-scalare è una forma di parallelismo implicito)

Un programma deve essere:
•Corretto
•Risolvere un problema importante (di grandi dimensioni)
•**Veloce** (altrimenti è inutile parallelizzare)

Esecuzione parallela

Esecuzione parallela richiede un Overhead:
- Scheduling.
- Coordinamento.



Scheduling:
•Analisi del carico di lavoro globale (scheduler).
•Partizionamento del carico sui diversi processori (scheduler).
•Coordinamento nella raccolta dei risultati (e.g. reorder station).

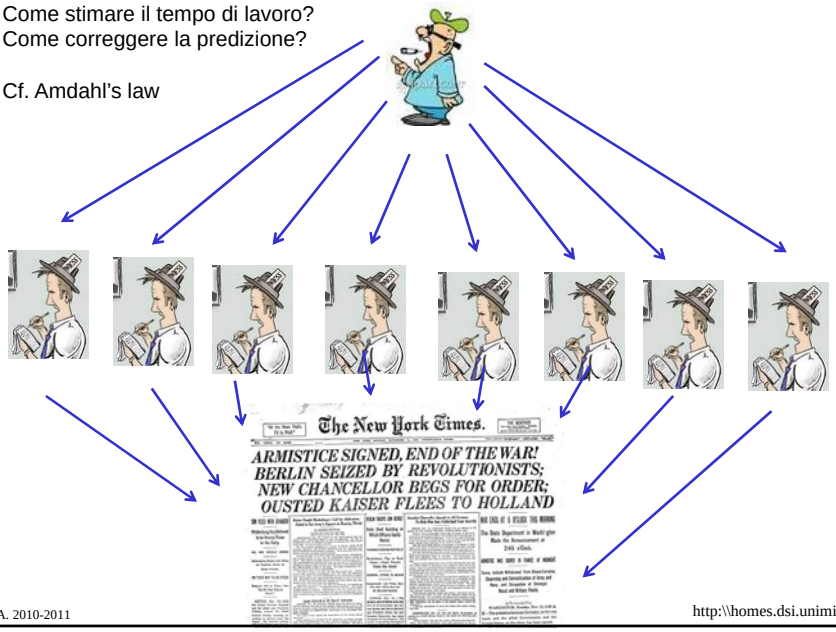
Lo scheduling può essere più (Pentium 4) o meno (CUDA) performante. Sempre più importante è il lavoro del programmatore / compilatore.

Infatti. Non si può "perdere" troppo tempo nello scheduling e/o nella compilazione.

Un esempio

Come stimare il tempo di lavoro?
Come correggere la predizione?

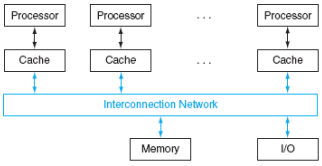
Cf. Amdahl's law



Esecuzione parallela con una memoria condivisa

La memoria viene suddivisa in memoria condivisibili e memoria non condivisibile o privata del singolo processo. In questo secondo caso la parte corrispondente della memoria principale non può essere utilizzata da altri processi.

- Competizione sui dati (data race).
- Occorre una coordinazione tra i diversi processi nell'accesso alla memoria (sincronizzazione) => **cache coherence**.
- Viene introdotta un'operazione atomica di scambio dei dati tra un registro ed una cella di memoria: nessun altro processssore o processo può inserirsi fino a quando l'operazione atomica non è terminata.
- Viene inserito un meccanismo hardware di blocco di una cella di memoria (lock o lucchetto).
- Viene gestito dal sotto-sistema di controllo della memoria dietro istruzioni della CPU.



La condivisione della memoria

Istruzioni corrispondenti al meccanismo hardware del lock:

- *load linked* (load collegata, `ll`) + *store conditional* (store condizionata, `sc`) che vengono utilizzate in sequenza.
- Se il contenuto di una locazione di memoria letta dalla load linked venisse alterato prima che la store condizionata abbia salvato in quella cella di memoria il dato, l'istruzione di store condizionata fallisce.
- L'istruzione di store condizionata ha quindi due funzioni: salva il contenuto di un registro in memoria *ed* imposta il contenuto di quel registro a 1 se la scrittura ha avuto successo e a 0 se invece è fallita.
- L'istruzione di load linked restituisce il valore letto e l'istruzione di store condizionata restituisce 1 solamente se la scrittura ha avuto successo.
- Controllo del contenuto del registro target associato all'istruzione di store condizionata.

Esecuzione parallela il quadro generale

		Software	
		Sequential	Concurrent
Hardware	Serial (pipeline)	Matrix Multiply written in MatLab running on an Intel Pentium 4	Windows Vista Operating System running on an Intel Pentium 4
	Parallel (pipeline)	Matrix Multiply written in MATLAB running on an Intel Xeon e5345 (Clovertown)	Windows Vista Operating System running on an Intel Xeon e5345 (Clovertown)

Alcuni task sono naturalmente paralleli (programmazione concorrente)

Altri task sono seriali e occorre capire come parallelizzarli in modo efficiente (moltiplicazione tra matrici)
Non è neppure facile parallelizzare task concorrenti in modo tale che le prestazioni aumentino con l'aumentare dei core

Esecuzione parallela e codice

		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: SSE instructions of x86
	Multiple	MISD: No examples today	MIMD: Intel Xeon e5345 (Clovertown)

SIMD (Single Instruction Multiple Data). Array Processor o calcolatori vettoriali. Istruzioni vettoriali: stessa istruzioni su più dati (prodotti di tutti gli elementi di un vettore per uno scalare). Funziona bene nella gestione dei vettori e matrici e con i cicli for. Funziona male quando ci sono branch (switch).

Parallelizzazione dell'esecuzione (multiple-issue) sono architetture MIMD.

Parallelizzazione dell'esecuzione - 1

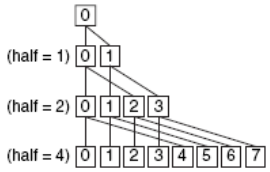
- Somma di 100,000 numeri (N=100,000) su un'architettura 100-core (P=100).
- Sommo N/P (=1,000) numeri su ciascun processore
  - Partizionamento dei dati in ingresso
  - Stessa memoria fisica

```
/* Execute in parallel on each Pn processor */
sum[Pn] = 0;
for (i = 1000*Pn; i < 1000*(Pn+1); i = i + 1)
       sum[Pn] = sum[Pn] + A[i]; /* sum the assigned areas*/
```

- Ottengo P = 100 somme parziali. Per ottenere la somma finale devo sommare tra loro le somme parziali (**riduzione**). Come?

Parallelizzazione dell'esecuzione - reduction

- Sommo i numeri a due a due in modo ricorsivo e gerarchico (**divide and conquer**)




```
half = 100; /* 100 processors in multiprocessor*/
repeat
   synch(); /* wait for partial sum completion */
       /* Conditional sum needed when half is odd */
       /* Processor0 gets missing element */
   if (half%2 != 0 && Pn == 0)
       sum[0] = sum[0] + sum[half-1];
   half = half/2; /* dividing line on who sums */
   if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn+half];
until (half == 1); /* exit with final sum in Sum[0] */
```

Osservazioni

`half = 100;` consente di scalare con il numero di processori

`if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn+half];`

Assegna ai diversi processori il ruolo (accumulatore o semplice memoria)

`synch(); /* wait for partial sum completion */`

Sincronizzazione esplicita alla fine di ogni livello di somme parziali

Distribuzione e sincronizzazione sono problematiche già viste nelle pipe-line superscalari dove venivano risolte dall'HW e/o dal compilatore. Qui distribuzione e sincronizzazione vengono eseguiti a livello di codice. Potrebbero essere eseguiti dai compilatori. Non sono ancora così "smart" per sfruttare appieno il parallelismo ...

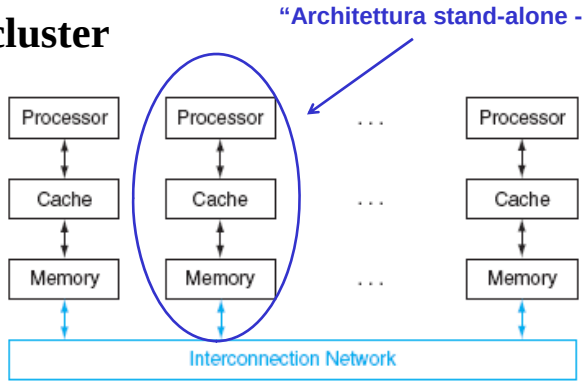
Sommario

Le architetture multi-processore

I cluster

Il modello "Roofline"

I cluster



Synchronization through **message passing** multiprocessors (sender – receiver).

Modalità tipica delle architetture SW concorrenti (Robot: AIBO Sony, File servers)
Ogni architettura ha la sua memoria, il suo SO. La rete di interconnessione non può essere così veloce come quella dei multi-processori.

E' un'architettura molto più robusta ai guasti, facile da espandere.

Massive parallelism -> data center -> Grid computing (SETI@home, 257 TeraFLOPS-> Cloud computing.

Parallelizzazione dell'esecuzione - 1

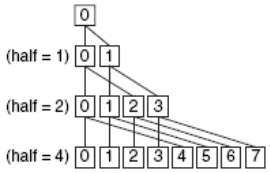
- Somma di 100,000 numeri (N=100,000) su un cluster con P=100 macchine.
- Sommo N/P (=1,000) numeri su ciascuna macchina
  - Partizionamento dei dati in ingresso inviandoli alle diverse macchine.
  - Ciascun gruppo di 1000 numeri risiede fisicamente su una macchina diversa.

```
/* Execute in parallel on each Pn processor */
sum = 0;
for (i = 1000; i < 1000; i = i + 1)
       sum = sum + A[i]; /* sum the assigned numbers */
```

- Ottengo P = 100 somme parziali. Per ottenere la somma finale devo sommare tra loro le somme parziali (**riduzione**). Come?

Parallelizzazione dell'esecuzione - reduction

- Sommo i numeri a due a due in modo ricorsivo e gerarchico (**divide and conquer**) ma le somme parziali sono qui su macchine fisicamente diverse




```
limit = 100; half = 100;/* 100 macchine */
repeat
       /* suddivido i processori in sender e receiver */
       half = (half+1)/2;
       if (Pn >= half && Pn < limit) send(Pn - half, sum);
       if (Pn < (limit/2)) sum = sum + receive();
       limit = half; /* ultimo sender */
until (half == 1); /* esci con la somma finale */
```


Sommario

Le architetture multi-processore

I cluster

Il modello "Roofline"

Evaluating Architecture performances

Throughput, Response time, Execution time

Small programs can be incredibly fast (kernel benchmarks)

Description	Name	Instruction Count × 10^9	CPI	Clock cycle time (seconds × 10^9)	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2,118	0.75	0.4	637	9,770	15.3
Block-sorting compression	bzip2	2,389	0.85	0.4	817	9,650	11.8
GNU C compiler	gcc	1,050	1.72	0.4	724	8,050	11.1
Combinatorial optimization	mcf	336	10.00	0.4	1,345	9,120	6.8
Go game (AI)	go	1,658	1.09	0.4	721	10,490	14.6
Search gene sequence	hmmer	2,783	0.80	0.4	890	9,330	10.5
Chess game (AI)	sjeng	2,176	0.96	0.4	837	12,100	14.5
Quantum computer simulation	libquantum	1,623	1.61	0.4	1,047	20,720	19.8
Video compression	h264avc	3,102	0.80	0.4	993	22,130	22.3
Discrete event simulation library	omnetpp	587	2.94	0.4	690	6,250	9.1
Games/path finding	astar	1,082	1.79	0.4	773	7,020	9.1
XML parsing	xalancbmk	1,058	2.70	0.4	1,143	6,900	6.0
Geometric Mean							11.7

SPEC (System Performance Evaluation Cooperative)

Parallel SPEC

Weak scaling: la dimensione del programma e dei dati è fissa

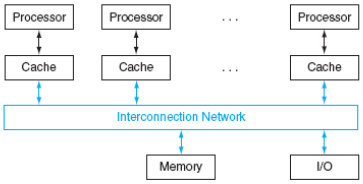
Strong scaling: la dimensione del programma e dei dati cresce proporzionalmente al numero dei processori.

Benchmark	Scaling?	Reprogram?	Description
Linpack	Weak	Yes	Dense matrix linear algebra [Dongarra, 1979]
SPECrate	Weak	No	Independent job parallelism [Henning, 2007]
Stanford Parallel Applications for Shared Memory SPLASH 2 [Woo et al., 1995]	Strong (although offers two problem sizes)	No	Complex 1D FFT Blocked LU Decomposition Blocked Sparse Cholesky Factorization Integer Radix Sort Barnes-Hut Adaptive Fast Multipole Ocean Simulation Hierarchical Radiosity Ray Tracer Volume Renderer Water Simulation with Spatial Data Structure Water Simulation without Spatial Data Structure
NAS Parallel Benchmarks [Bailey et al., 1991]	Weak	Yes (C or Fortran only)	EP: embarrassingly parallel MG: simplified multigrid CG: unstructured grid for a conjugate gradient method FT: 3-D partial differential equation solution using FFTs IS: large integer sort
PARSEC Benchmark Suite [Bienia et al., 2008]	Weak	No	Blackscholes—Option pricing with Black-Scholes PDE Bodytrack—Body tracking of a person Canneal—Simulated cache-aware annealing to optimize routing Dedup—Next-generation compression with data deduplication Facesim—Simulates the motions of a human face Ferret—Content similarity search server Fluidanimate—Fluid dynamics for animation with SPH method Freqmine—Frequent itemset mining Streamcluster—Online clustering of an input stream Swaptions—Pricing of a portfolio of swaptions Vips—Image processing x264—H.264 video encoding
Berkeley Design Patterns [Asanovic et al., 2006]	Strong or Weak	Yes	Finite-State Machine Combinational Logic Graph Traversal Structured Grid Dense Matrix Sparse Matrix Spectral Methods (FFT) Dynamic Programming N-Body MapReduce Backtrack/Branch and Bound Graphical Model Inference Unstructured Grid

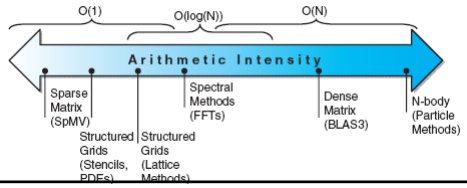
Arithmetic intensity

Velocità di calcolo FLOPS (floating point per second): velocità del singolo core: V_{core}.
In un'achitettura multi-core con N core la velocità di calcolo $V_{calc} = P * V_{core}$

Velocità di trasferimento dalla gerarchia di memoria. Per calcolarla dobbiamo capire quante operazioni devono essere fatte per ciascun byte caricato in cache, $N_{op} / _{Byte}$ **(Arithmetic Intensity)**



Se effettuiamo N_{op} operazioni su ogni byte letto dalla memoria, avremo una velocità di calcolo massima pari a: $V_{max} = V_{mem} * N_{byte}$ = [Byte /s].



Weak scaling, less memory request per byte

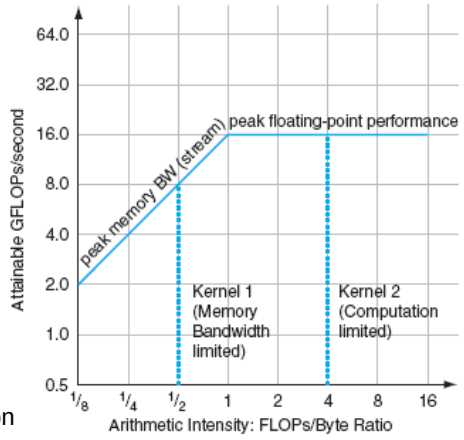
Il modello "roofline"

2 elements:
- Computation
- Memory transfer

Nessun benchmark può essere contenuto interamente in cache.

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

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



AMD Opteron X2

Il modello “roofline”: osservazioni

2 elements:
- Computation
- Memory transfer

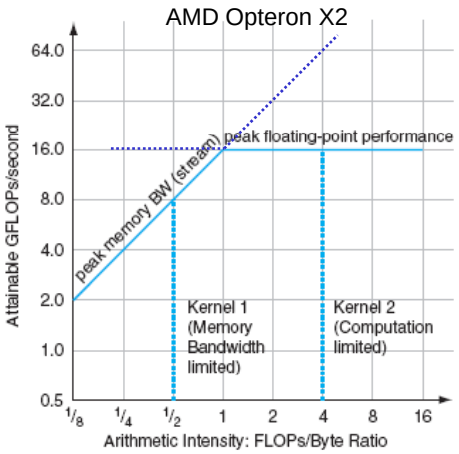
Se non ci fossero problemi con la memoria le prestazioni sarebbero una linea orizzontale pari alla massima capacità di calcolo:

V_{calc} = cost =
16 GFLOPS sull’Opteron X2.

Se tutto quello che viene letto dalla memoria alla massima velocità potesse essere calcolato, si avrebbe una velocità di calcolo pari a:

$$V_{calc} = V_{mem} * N_{op} / {}_{Byte}$$

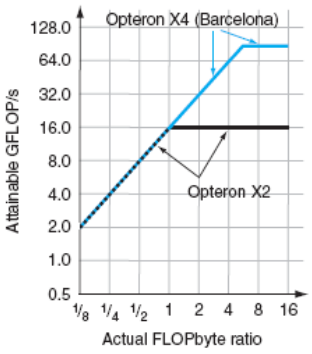
V_{mem} = 16GFlop/s



Si disegnano le due curve e le prestazioni sono limitate dalla curva più bassa è sufficiente conoscere l’intensità aritmetica del programma.

Dall’Opteron X2 all’Opteron X4

Opteron X4 vs Opteron X2:
- Stesso sistema di memoria
- Numero doppio di processori (core)
- Numero quadruplo di operazionio 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.

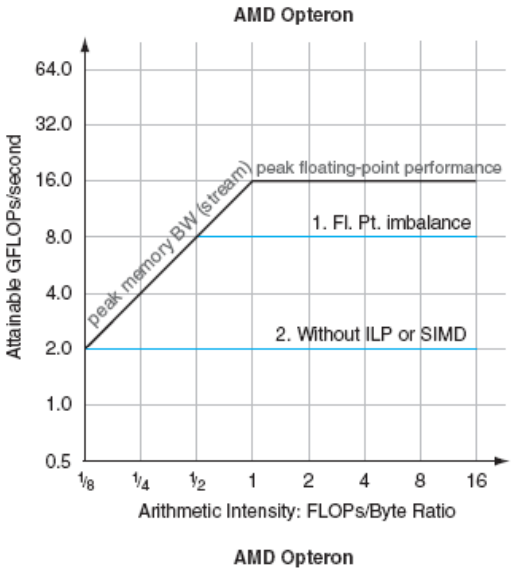
La velcocità di calcolo massima di 128GFLOPS si raggiunge solo per un’intensità aritmetica pari a 5.

Ottimizzazioni sulla parte di calcolo

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

Mix di operazioni:
- Floating point
- Moltiplicazioni e addizioni

Parallelizzazione dell'esecuzione
- Srotolamento dei cicli



Ottimizzazioni sulla parte di calcolo

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

Mix di operazioni:
- Floating point
- Moltiplicazioni e addizioni

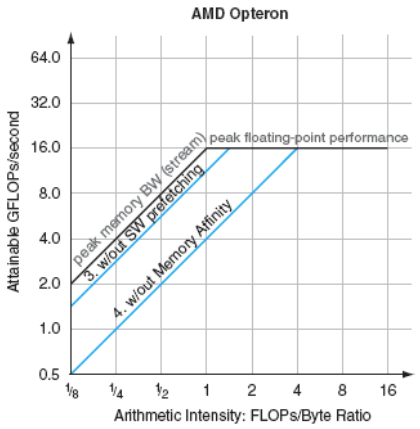
Parallelizzazione dell'esecuzione
- Srotolamento dei cicli

Ottimizzazioni sulla parte di memoria

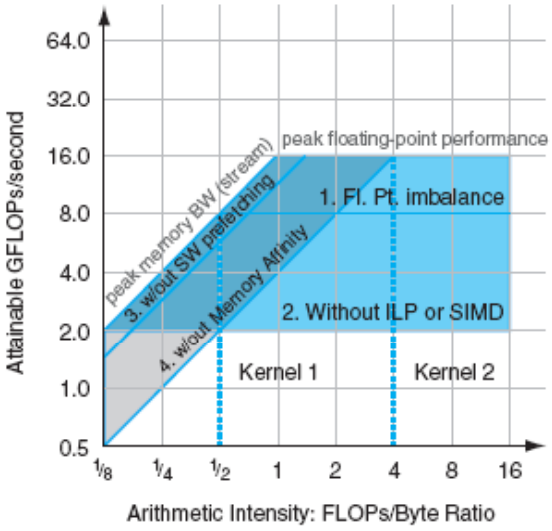
Cosa succede se le prestazioni del vostro programma risultano scadenti rispetto alle prestazioni attese?

Software pre-fetching:
-Precaricamento dei dati in cache

Affinità della memoria:
- Massimizzare gli hit



Quali ottimizzazioni?



Sommario

Le architetture multi-processore

I cluster

Il modello “Roofline”