

Calcolo parallelo e distribuito mod.B

Lezione 2: Influenza della cache sugli algoritmi

Prestazioni (performance) del software

- La Top500 list classifica i supercalcolatori sulla base delle prestazioni ottenute con il LINPACK benchmark, un software per la risoluzione di sistemi di equazioni lineari
- Le prestazioni sono misurate contando il numero di operazioni f.p. eseguite per unita' di tempo, cioe'

$$\text{Perf} = \frac{N_{\text{flop}}}{T_{\text{sw}}}$$

Numero complessivo di operazioni

Tempo complessivo di esecuzione

Unita' di misura delle prestazioni dei supercomputer

prestazioni	• 1 Gflop/sec = 10 ⁹ op.f.p./sec	1 Gigaflop/sec (1985)
	• 1 Tflop/sec = 10 ¹² op.f.p./sec	1 Teraflop/sec (1997)
	• 1 Pflop/sec = 10 ¹⁵ op.f.p./sec	1 Petaflop/sec (2008)
	• 1 Eflop/sec = 10 ¹⁸ op.f.p./sec	1 Exaflop/sec (2019 ?)
Dim. memoria	• 1 GB = 10 ⁹ Bytes	1 Gigabytes
	• 1 TB = 10 ¹² Bytes	1 Terabytes
	• 1 PB = 10 ¹⁵ Bytes	1 Petabytes
	• 1 EB = 10 ¹⁸ Bytes	1 Exabytes

Complessita' di un algoritmo e prestazioni del software

- Tipicamente la bonta' di un algoritmo si misura mediante la
- Complessita' di tempo (numero di operazioni): N_{flop}
 - Complessita' di spazio (numero di locazioni id memoria)

T_{sw} dipende anche da altri fattori

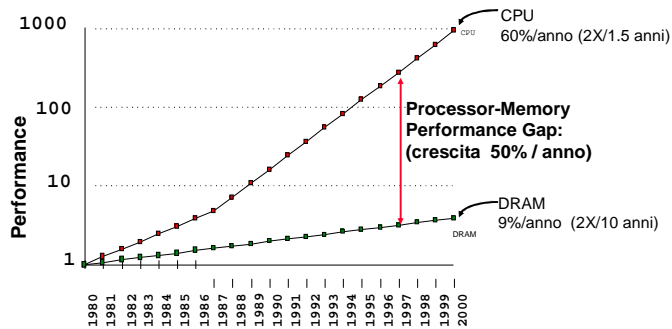
$$T_{\text{sw}} = N_{\text{flop}} \times t_{\text{flop}}$$

Algoritmo Hardware

Tempo per eseguire solo le operazioni f.p.

Così si trascura però il tempo di accesso alla memoria t_{mem}

E' ragionevole trascurare t_{mem} ?



Nel valutare il tempo di esecuzione **non e' realistico trascurare** il tempo di accesso alla memoria t_{mem}

Quindi...

tempo di esecuzione di un sw

$$T_{SW} = N_{mem} \times t_{mem} + N_{flop} \times t_{flop}$$

Algorithm Hardware Algorithm Hardware
 T_{mem} Tempo per accessi alla memoria T_{flop} Tempo per op. floating point

N_{mem} e' un fattore di amplificazione per t_{mem}

Valutiamo l'impatto di N_{mem} su Perf

Impatto di N_{mem} su Perf

- Calcoliamo

$$Perf = \frac{N_{flop}}{T_{SW}}$$

- caso **ideale** ($t_{mem} = 0$)
- (Gli accessi alla memoria non hanno nessun impatto)

$$Perf^* = \frac{N_{flop}}{T_{SW}} = \frac{N_{flop}}{N_{flop} \times t_{flop}} = \frac{1}{t_{flop}}$$

$Perf^* = Peak Performance$

Esempio : Intel Woodcrest (dual core)

- CPU a 3 GHz
- 4 op.f.p. eseguite ogni ciclo di clock in ogni core
- 1 ciclo di clock = $\frac{1}{3 \times 10^9}$ sec = 0.33×10^{-9} sec

$$t_{flop} = 0.33 \times 10^{-9} / 4 \text{ sec} = 0.0825 \times 10^{-9} \text{ sec}$$

$$Perf^* = 1/t_{flop} = 12.12 \times 10^9 \text{ op.f.p./sec} = 12.12 \text{ Gflop/sec}$$

(peak performance di 1 core)

$$Perf^* = 24.24 \text{ Gflop/sec}$$

(peak performance dell'intera CPU 2-core)

Impatto di N_{mem} su Perf

- Caso reale: $t_{mem} > 0$

$$Perf = \frac{N_{flop}}{T_{SW}} = \frac{N_{flop}}{N_{mem} \times t_{mem} + N_{flop} \times t_{flop}}$$



Moltiplicando num. e den. per $1/(N_{flop} \times t_{flop})$

$$Perf = \frac{1/t_{flop}}{1 + \left(\frac{N_{mem} \cdot t_{mem}}{N_{flop} \cdot t_{flop}} \right)} = \frac{Perf^*}{1 + \left(\frac{N_{mem} \cdot t_{mem}}{N_{flop} \cdot t_{flop}} \right)}$$

Fattore di decadimento della performance

Impatto di N_{mem} su perf

$$\frac{N_{mem}}{N_{flop}} \quad \frac{t_{mem}}{t_{flop}}$$

Dipende dall'algoritmo

Dipende dall'hardware
(c'è poco da fare)

è il numero di accessi fatti alla memoria per ogni operazione eseguita

Ridurre $q = \frac{N_{mem}}{N_{flop}}$ significa avvicinare il caso reale al caso ideale (e quindi le prestazioni Perf alla Peak Performance Perf*)

osservazione

$$Perf = \frac{Perf^*}{1 + \left(q \frac{t_{mem}}{t_{flop}} \right)}$$

deve essere almeno $q \approx \frac{t_{flop}}{t_{mem}}$

Per ottenere meta' della peak performance

Riassumendo...

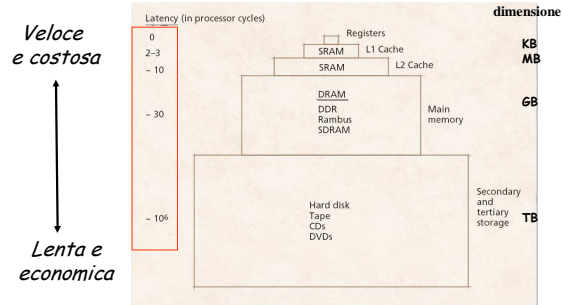
- Una metodologia chiave per avvicinare le prestazioni di una applicazioni a quelle della Peak Performance di un processore è'

Ridurre gli accessi alla memoria



Importanza della memoria gerarchica

La memoria gerarchica



Utilizzare i dati nei livelli alti della memoria (cache L1 e L2) significa ridurre gli accessi alla memoria e sostenere più facilmente la velocità operativa della CPU

esempio

- CPU capace di eseguire 1 flop in un ciclo di 10^{-9} sec
- Dati nella cache L1 (~ 2 cicli per l'accesso)
 - Necessarie $q=0.5$ (cioè 2 flop per dato trasferito) per avere il 50% di Perf*
- Dati nella cache L2 (~ 10 cicli per l'accesso)
 - Necessarie $q=0.1$ (cioè 10 flop per dato trasferito) per avere il 50% di Perf*
- Dati in memoria centrale (~30 cicli per l'accesso)
 - Necessarie $q=0.03$ (cioè 30 flop per dato trasferito) per avere il 50% di Perf*

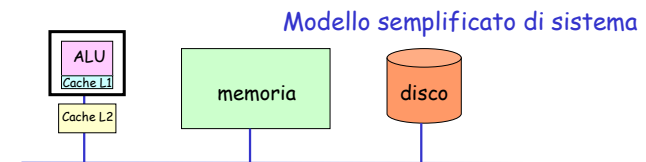
Il gap tra t_{mem} e t_{flop} viene colmato dalle cache

Memorizzazione delle matrici

- Una matrice A è un array 2-D di elementi ma la memoria è un array 1-D di locazioni di memoria
- Matrici in memoria
 - Per colonna, o "column major" (es. Fortran)
 - $A(i,j)$ si trova in $\&A(0,0) + i + j*n$
 - Per riga, o "row major" (es. C)
 - $A(i,j)$ si trova in $\&A(0,0) + i*n + j$

	per colonna				per riga			
↓	0	5	10	15	0	1	2	3
	1	6	11	16	4	5	6	7
	2	7	12	17	8	9	10	11
	3	8	13	18	12	13	14	15
	4	9	14	19	16	17	18	19

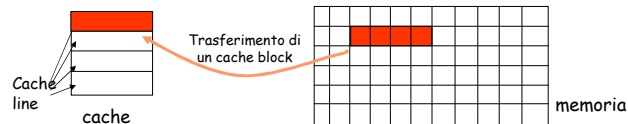
Come funzionano le cache



- i dati passano dalla memoria alle cache L1 e L2
 - la cache L1 è sempre sul chip della CPU
 - la cache L2 può essere interna o esterna al chip della CPU
 - esistono CPU con cache L3 (sempre esterna al chip)
- tali dati permangono nelle cache per i riferimenti successivi
 - quando la CPU richiede un dato viene interrogata la cache L1
 - se il dato è presente in cache L1 viene utilizzato, se è assente (L1-miss) si interroga la cache L2
 - se il dato è presente in cache L2 viene utilizzato, se è assente (L2-miss) si interroga la memoria centrale

Come funzionano le cache

- Le cache sono di **piccole dimensioni**
 - Esempio Intel Gainestown 4core (2009)
 - L1 di 64 KB, L2 di 256 KB, L3 di 4 MB
 - quando la CPU richiede nuovi dati e la cache e' piena, i vecchi dati sono sostituiti dai nuovi nelle cache (con algoritmi LRU-like)
- Le cache sono organizzate in "cache line" e il trasferimento dei dati dalla memoria avviene in blocchi di **dati contigui** (cache block)
 - quando un dato e' richiesto viene trasferito un **blocco** di elementi vicini
 - se tutti i dati del blocco vengono utilizzati si **ammortizza il costo** del cache miss



Come funzionano le cache

- Ogni livello di cache ha differenti
 - latenza (tempo di accesso alla cache)
 - bandwidth (numero di bytes trasferiti nell'unita' di tempo)
- Numerosi livelli di cache



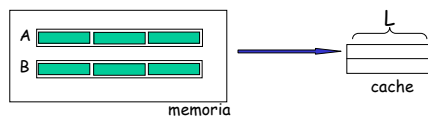
Una gestione efficiente della cache e' possibile solo attraverso linguaggi a basso livello e dipende fortemente dal sistema operativo, ma qualcosa e' possibile fare anche a livello di applicazione



Principale metodologia
Ristrutturare gli algoritmi in maniera da **riutilizzare** piu' volte **tutti** i dati del cache block

Esempio (un livello di cache)

- Due array A e B di lunghezza N,
- Sistema con memoria centrale e una sola cache
- Lunghezza della Cache Line = L
- Cache con due cache line (una per A e una per B)
- A e B sono formati da N/L blocchi



Problema:
ogni elemento di A deve essere sommato
a tutti gli elementi di B

Due possibili algoritmi

Forma i j

```
for i = 0 to N-1
  for j = 0 to N-1
    A[i] = A[i] + B[j]
  endfor
endfor
```

- gli elementi di A sono portati in cache e poi rimemorizzati
 - per $i = 0, L, 2L \dots$ ($2N/L$ accessi)
- per ogni valore di i gli elementi di B sono portati in cache
 - per $j = 0, L, 2L \dots$ (N volte N/L accessi)

Numero di cache miss (N_{mem}) = $2N/L + N^2/L$

Forma j i

```

for j = 0 to N-1
  for i = 0 to N-1
    A[i] = A[i] + B[j]
  endfor
endfor
    
```

- per ogni valore di j gli elementi di A sono portati in cache e poi rimemorizzati
 - per $i = 0, L, 2L \dots$ (N volte $2N/L$ accessi)
- gli elementi di B sono portati in cache
 - per $j = 0, L, 2L \dots$ (N/L accessi)

$$\text{Numero di cache miss } (N_{\text{mem}}) = 2N^2/L + N/L$$

E' possibile ridurre N_{mem} ?

```

for j=0 to N-1 step L           // per un fissato indice j ...
  for i=0 to N-1               // ... tutti gli elementi di A...
    for jj=j to min(j+L, N)-1 // ... sono sommati ad un blocco di B
      A[i] = A[i] + B[jj];
    endfor
  endfor
endfor
    
```

		Accessi su		
		A	B	
j=0	i=0	jj=0,...,j+L-1	2	1
	i=1	jj=0,...,j+L-1	0	0
	...	jj=0,...,j+L-1	2	...
	i=N-1	jj=0,...,j+L-1	0	0
	Tot		2N/L	1

ripetere per $j=L, 2L, \dots, N$ (N/L volte)

$$\text{Numero di cache miss } (N_{\text{mem}}) = 2N^2/L^2 + N/L$$



Alcune semplificazioni del seguito

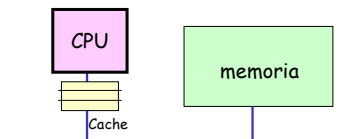
- Sistema con 2 livelli di memoria (Memoria Veloce e Memoria Lenta)
- Si ignora un eventuale parallelismo tra accesso in memoria e ALU
- Memoria veloce di dimensioni tali da contenere almeno 3 righe
 - Ragionevole se si pensa alle cache, ma limitativo
 - Non ragionevole se pensiamo ai registri della CPU (~32 words)
- Tempo di accesso alla memoria veloce trascurabile
 - Ragionevole se pensiamo ai registri
 - Non ragionevole se pensiamo alle cache (1-2 cicli per L1)

23

Esempio: operazione $C = \beta C + \alpha AB$

Modello semplificato di sistema

- Sistema con 2 livelli di memoria (Cache e Memoria centrale)
- Cache di dimensioni tali da contenere 3 righe (una riga per ogni matrice)
- Tempo di accesso alla memoria veloce trascurabile



Esempio: prodotto tra matrici

A, B e C matrici di ordine N
 α e β scalari

```

for __ = 1 to N
  for __ = 1 to N
    for __ = 1 to N
      C(i,j) =  $\beta$ *C(i,j) +  $\alpha$ *A(i,k)*B(k,j)
    endfor
  endfor
endfor
    
```

6 combinazioni di indici

- i j k
- j i k
- i k j
- j k i
- k i j
- k j i

Per ogni combinazione di indici sono sempre $4N^2$ operazioni
 (ma il numero di accessi?)

Osservazione: C richiede 2 accessi; A e B un solo

Prestazioni delle sei versioni in Gflop/sec

- Esecuzione su Intel core i7 (Bloomfield) 3GHz (Perf*=12 Gflops)
- Compilatore cc con opzione -O3, S.O. Linux 6.2
- Matrici double di ordine **N=50**

	Gflop/sec
i j k	3.03
j i k	2.90
i k j	3.49
j k i	2.52
k i j	3.42
k j i	2.59

Max perf } + 38%
 Min perf } sul min !!

Forma i k j (caso N=L: riga intera in cache)

```

for i = 1 to N
  for k = 1 to N
    for j = 1 to N
      C(i,j) =  $\beta$ *C(i,j) +  $\alpha$ *A(i,k)*B(k,j)
    endfor
  endfor
endfor
    
```

		Accessi su			
		C	A	B	
i=1	k=1	j=1,...,n	1	1	1
	k=2	j=1,...,n	0	0	1
	...	j=1,...,n
	k=N	j=1,...,n	1	0	1
tot			2	1	N

(ripetere per i=2,...,N)

Totale accessi $N_{mem} = N(3+N) = 3N+N^2$

Forma j k i (caso N=L: riga intera in cache)

```

for j = 1 to N
  for k = 1 to N
    for i = 1 to N
      C(i,j) =  $\beta$ *C(i,j) +  $\alpha$ *A(i,k)*B(k,j)
    endfor
  endfor
endfor
    
```

		Accessi su			
		C	A	B	
j=1	k=1	i=1,...,n	2N	N	1
	k=2	i=1,...,n	2N	N	1
	...	i=1,...,n
	k=N	i=1,...,n	2N	N	1
Tot			$2N^2$	N^2	N

(ripetere per j=2,...,N)

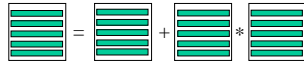
Totale accessi $N_{mem} = N(3N^2+N) = 3N^3+N^2$

Forma i j k (caso N=L: riga intera in cache)

```

for i = 1 to N
  for j = 1 to N
    for k = 1 to N
      C(i,j) = beta*C(i,j) + alpha*A(i,k)*B(k,j)
    endfor
  endfor
endfor

```



Accessi su

		C	A	B
j=1	k=1,...,n	1	1	N
j=2	k=1,...,n	0	0	N
...	k=1,...,n
j=N	k=1,...,n	1	0	N
tot		2	1	N ²

(ripetere per i=2,...,N)

Totale accessi $N_{mem} = N(3+N^2) = 3N+N^3$

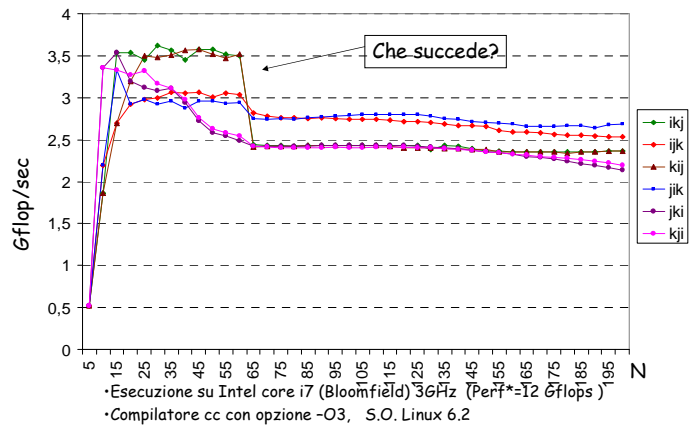
Confronto Gflop/sec vs q (N=50)

Per una operazione $C=\beta C+\alpha AB$ di ordine $N=L$ si ha $N_{flop}=4L^3$

	Gflop/sec	N_{mem}	$q=N_{mem}/N_{flop}$
i j k	3.03	$3N+N^3$	$\frac{3}{4L^2} + \frac{1}{4}$
j i k	2.90	$3N^2+N^3$	$\frac{3}{4L} + \frac{1}{4}$
i k j	3.49	$3N+N^2$	$\frac{3}{4L^2} + \frac{1}{4L}$
j k i	2.52	$3N^3+N^2$	$\frac{3}{4} + \frac{1}{4L}$
k i j	3.42	$3N^2+N$	$\frac{3}{4L} + \frac{1}{4L^2}$
k j i	2.59	$3N^3+N$	$\frac{3}{4} + \frac{1}{4L^2}$

Max perf → (for i j k, j i k, i k j)
 Min perf → (for j k i, k i j, k j i)
 Min q ← (for i j k, j i k)
 Max q ← (for j k i, k i j, k j i)

Prodotto $C=\beta C+\alpha AB$ al variare di N



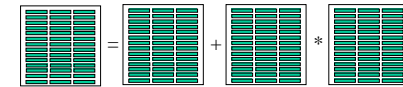
Calcolo dei cache miss (N / L intero)

Forma i k j

```

for i = 1 to N
  for k = 1 to N
    for j = 1 to N
      C(i,j) = beta*C(i,j) + alpha*A(i,k)*B(k,j)
    endfor
  endfor
endfor

```



Accessi su

		C	A	B
k=1	j=1,...,n	$2N/L$	1	N/L
k=2	j=1,...,n	$2N/L$	0	N/L
...	j=1,...,n	...	1	...
k=N	j=1,...,n	$2N/L$	0	N/L
tot		$2N^2/L$	N/L	N^2/L

(ripetere per i=2,...,N)

Totale accessi = $N(3N^2/L + N/L) = 3N^3/L + N^2/L$

Calcolo di $q = N_{\text{mem}} / N_{\text{flop}}$

- Operazione tra matrici $C = \beta C + \alpha AB$
 - $N_{\text{flop}} = 4N^3$

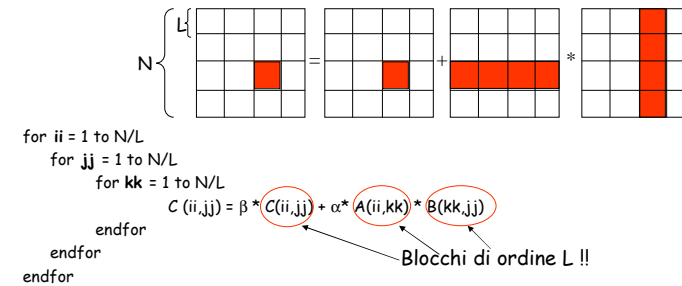
$$q = \frac{3N^3/L + N^2/L}{4N^3} = \frac{3}{4L} + \frac{1}{4NL}$$

Tale valore e' maggiore del corrispondente q del caso $N=L$

$$q = \frac{3}{4L^2} + \frac{1}{4L}$$

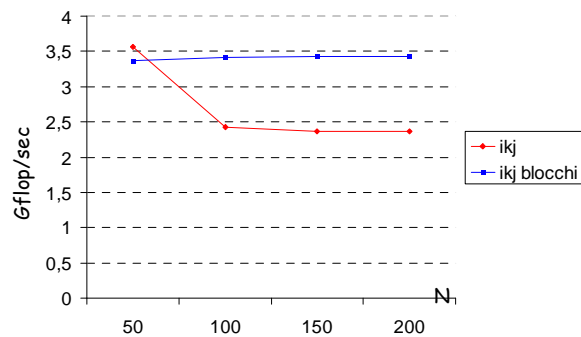
Possibile soluzione (N/L intero)

- Suddividere le matrici in blocchi di ordine L cosi' da poter applicare il caso $N=L$
- Ogni blocco di C e' il prodotto di un blocco di righe di A e un blocco di colonne di B



6 cicli innestati !!!

Prodotto matrici a blocchi



Calcolo di q

```

for ii = 1 to N/L
  for jj = 1 to N/L
    for kk = 1 to N/L
      C(ii,jj) = beta * C(ii,jj) + alpha * A(ii,kk) * B(kk,jj)
    endfor
  endfor
endfor
    
```

$$N^3/L^3 \text{ operazioni di ordine } L \Rightarrow N_{\text{mem}} = \frac{N^3}{L^3} (3L + L^2)$$

$$q = \frac{N^3}{L^3} \frac{(3L + L^2)}{4N^3} = \frac{3L + L^2}{4L^3} = \frac{3}{4L^2} + \frac{1}{4L} < \frac{3}{4L} + \frac{1}{4NL}$$

Cosa accade se ci sono 2 o piu' livelli di cache?

- E' necessario
 - Minimizzare la comunicazione tra tutti i livelli
 - trovare le giuste dimensioni dei blocchi
- Fortemente dipendente dall'architettura
 - Solo memoria centrale \Rightarrow 3 cicli innestati
 - 1 livello di cache \Rightarrow 6 cicli innestati
 - 1 livello di cache \Rightarrow 9 cicli innestati
- Necessita' di strumenti di piu' "basso livello"

37

• FINE