Calcolo Scientifico: II lezione

Introduzione a BLAS:

Ovvero lo sviluppo di software

efficiente

per le operazioni di base

del calcolo matriciale

L'efficienza di un software…

$$T_s \cong k \cdot \mu \cdot T(N)$$

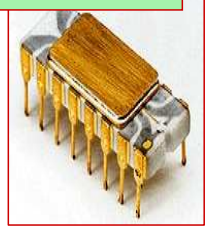
Tempo per l'esecuzione delle operazioni floating point
Tecnologia Hardware

Complessità di tempo dell'Algoritmo
Tecnologia Software

Evoluzione della tecnologia hardware

	1970	1995	2010
Velocità del proc.	20 M flops	6 G flops	10 Tera flops
Memoria RAM	200 K words	50 M words	100 G words
Hard disk	200 M words	500 G words	1 Peta words

Intel 4004



IBM® Power 3®

IBM RS/6000 SP

CRAY 1

ASCII Red

Evoluzione della tecnologia software

	1945	1955	1965	1975	1985
Tempo	2×10^6 anni	20 anni	1 giorno	12 ore	0.2 sec
Memoria	800 M words	5 M words	300 K words	170 K words	50 K words

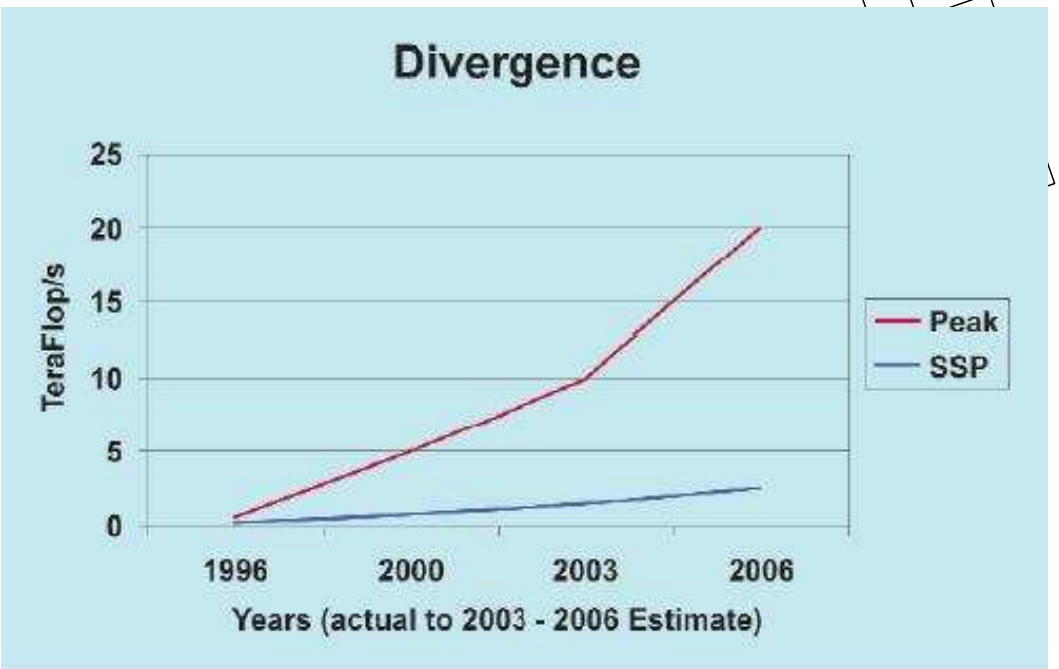
costo necessario alla risoluzione Computazinale di uno stesso problema

Obiettivo

- Si possono avere alte prestazioni
  - Migliorando la tecnologia hardware
  - Migliorando la tecnologia software

MA .....

Sono sempre maggiori le difficoltà nell'ottenere le massime prestazioni



Obiettivo:

ridurre il gap tra la *peak* performance

(la massima prestazione del calcolatore)

e la *sustained* performance

(prestazione massima del software su quel calcolatore)

Nel dettaglio...

T_S= tempo di esecuzione di un sw

$$T_S(N) = N_{mem} \times t_{mem} + N_{flop} \times t_{flop}$$

Algoritmo Hardware Algoritmo Hardware

$$T_S(N) = \mathbf{T_{mem}(N)} + \mathbf{T_{flop}(N)}$$

$\mathbf{T_{mem}(N)}$: Tempo per accessi alla memoria

$\mathbf{T_{flop}(N)}$: Tempo per op. floating point

Quando il sw raggiunge la massima prestazione ?

$T_S(N) \leq T_{flop}(N) \Rightarrow \frac{T_S(N)}{T_{flop}(N)} \leq 1$

OVVERO

quando il tempo di esecuzione di un software $T_s(N)$

è al più uguale al tempo $T_{flop}(N)$ dell'algoritmo

MA in generale.....

$$\frac{T_S(N)}{T_{flop}(N)} = \frac{N_{mem}t_{mem} + N_{flop} \times t_{flop}}{N_{flop} \times t_{flop}} = 1 + \frac{N_{mem} \times t_{mem}}{N_{flop} \times t_{flop}}$$

avviene che

$T_S(N) > T_{flop}(N)$

per migliorare l'efficienza dobbiamo

rendere "piccolo" il rapporto

Per migliorare l'efficienza…

$$\frac{T_S(N)}{T_{flop}(N)} = 1 + \frac{N_{mem} \times t_{mem}}{N_{flop} \times t_{flop}}$$

Problema 1

Ridurre il numero di accessi alla memoria

Problema 2

Ridurre il tempo di accesso alla memoria

Problema

$$\frac{T_S(N)}{T_{flop}(N)} = 1 + \frac{N_{mem} \times t_{mem}}{N_{flop} \times t_{flop}}$$

$$q = \frac{N_{mem}}{N_{flop}} = \frac{\#\text{ accessi in memoria}}{\#\text{ operazioni floating - point}}$$

Parametro di valutazione del traffico "parassita"

Quanto vale il parametro q

(misura del traffico parassita)

per le operazioni di base di algebra lineare

?

Aggiornamento di un vettore

Prodotto scalare

Prodotto Matrice vettore

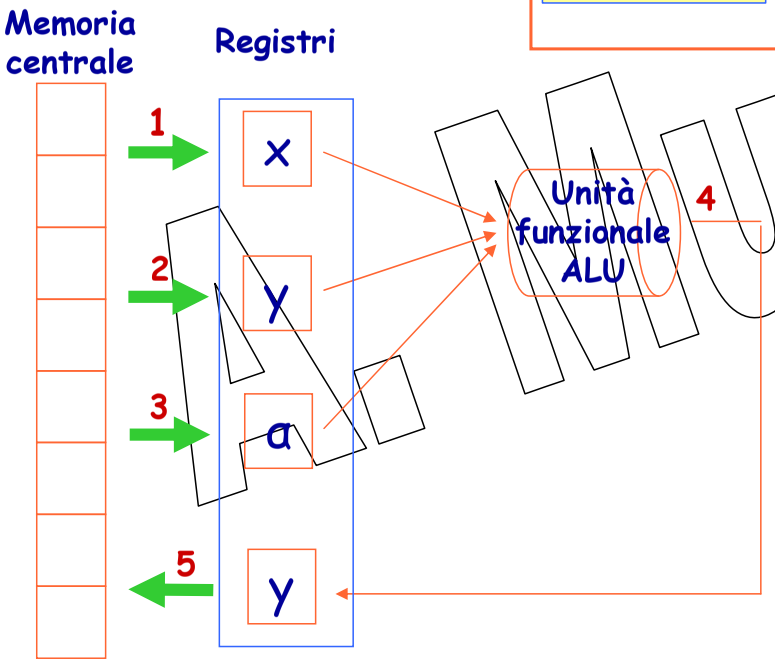
Prodotto Matrice -Matice

Esempio1: aggiornamento di un vettore (saxpy)

$\underline{y} = \underline{y} + \alpha \underline{x}$

$\underline{x}, \underline{y}$ Vettori di dim. n

α scalare



1 - preleva x → n accessi

2 - preleva y → n accessi

3 - preleva α → 1 accesso

4 - calcolo → 2n op. fl.p.

5 - riponi y → n accessi

Calcoliamo q....

SAXPY ⇒ $q = \frac{m}{f} = \frac{3n+1}{2n} \approx \frac{3}{2}$

1 - preleva x → n accessi

2 - preleva y → n accessi

3 - preleva α → 1 accesso

4 - calcolo → 2n op. fl.p.

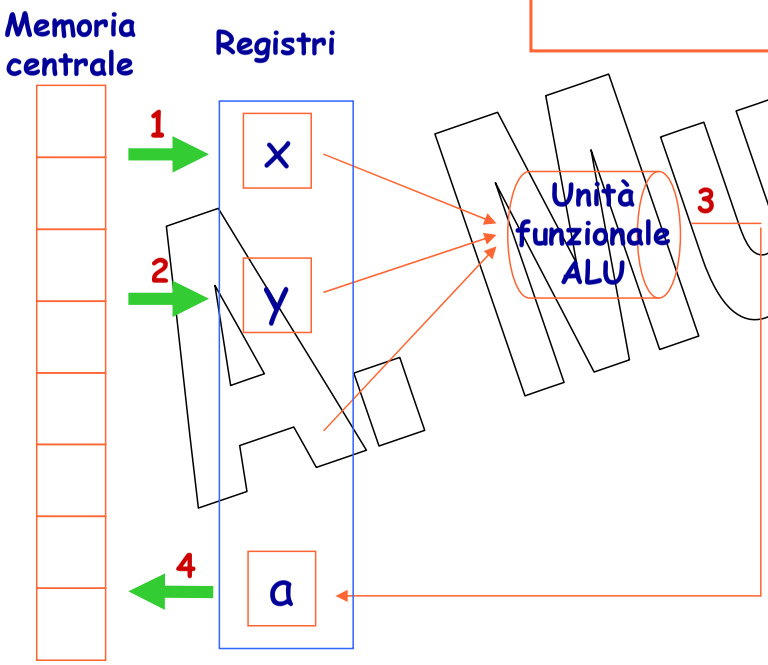
5 - riponi y → n accessi

accessi > # operazioni f.p.

Esempio2: prodotto scalare (dot)

$a = \underline{x}^T \underline{y}$ $\underline{x}, \underline{y}$ Vettori di dim. n

α scalare



1 - preleva x → n accessi

2 - preleva y → n accessi

3 - calcolo → 2n op. fl.p.

4 riponi α → 1 accesso

Calcoliamo q....

DOT ⇒ $q = \frac{m}{f} = \frac{2n+1}{2n} \approx 1$

1 - preleva x → n accessi

2 - preleva y → n accessi

3 - calcolo → 2n op. fl.p.

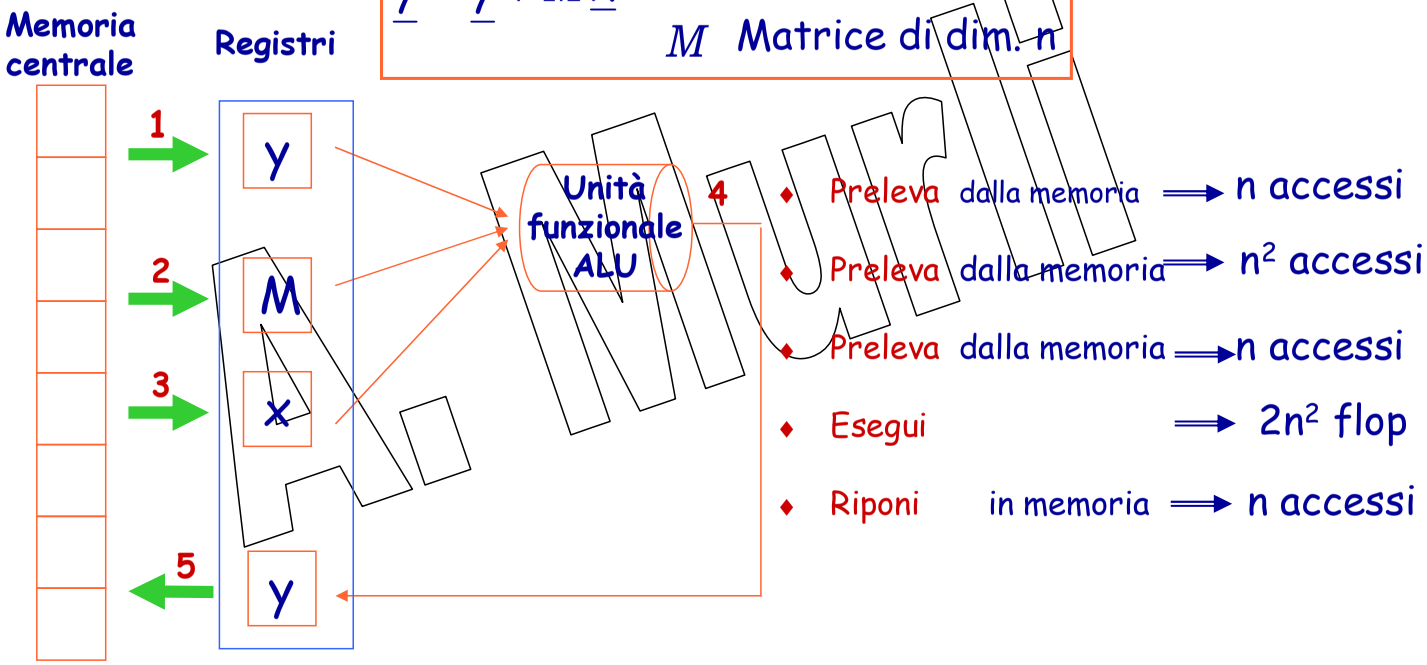
4 riponi α → 1 accesso

accessi = # operazioni f.p.

Esempio3: prodotto matrice vettore (gemv)

$\underline{y} = \underline{y} + M\underline{x}$ $\underline{x}, \underline{y}$ Vettori di dim. n

M Matrice di dim. n



Calcoliamo q....

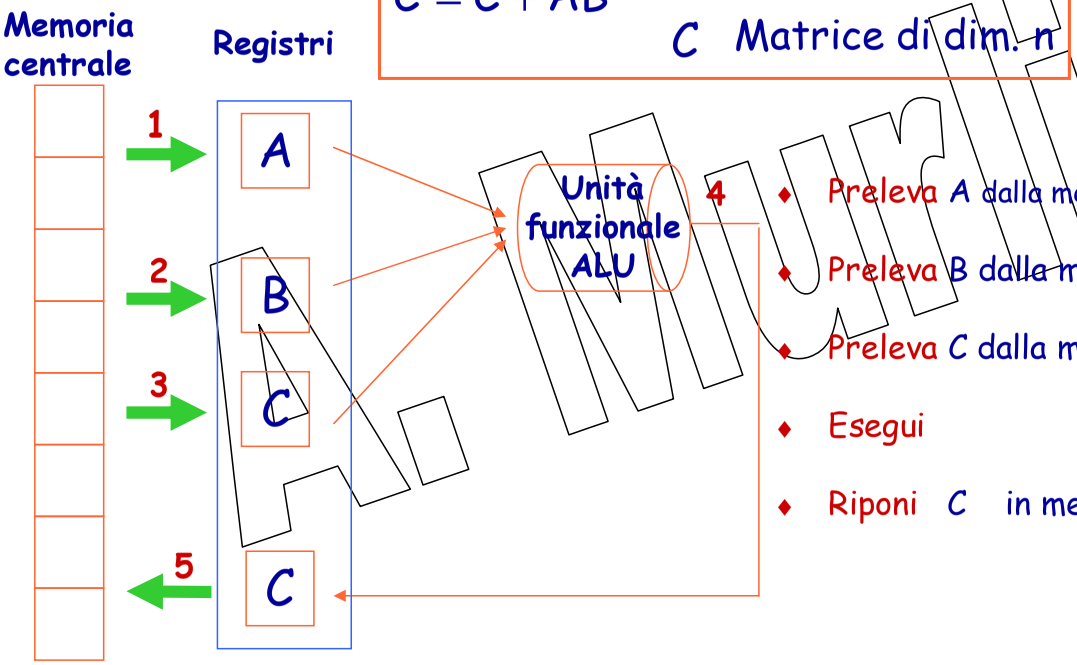
GEMV ⇒ $q_{gemv} = \frac{3n+n^2}{2n^2} = \frac{1}{2} + \frac{3}{2n}$

- Preleva dalla memoria il vettore x ⇒ n accessi
- Preleva dalla memoria la matrice M ⇒ n^2 accessi
- Preleva dalla memoria ilvettore y ⇒ n accessi
- Esegui ⇒ $2n^2$ flop
- Riponi y in memoria ⇒ n accessi

 # accessi > # operazioni f.p.

Esempio 4: prodotto matrice matrice (gemm)

$C = C + AB$ A, B Matrici di dim. n; C Matrice di dim. n



- Preleva A dalla memoria ⇒ n^2 accessi
- Preleva B dalla memoria ⇒ n^2 accessi
- Preleva C dalla memoria ⇒ n^2 accessi
- Esegui ⇒ $2n^3$ flop
- Riponi C in memoria ⇒ n^2 accessi

Calcoliamo q....

GEMM ⇒ $q_{gemm} = \frac{3n^2 + (n)^2}{2(n)^3} = \frac{2}{n}$

- Preleva dalla memoria ⇒ n^2 accessi
- Preleva dalla memoria ⇒ n^2 accessi
- Preleva dalla memoria ⇒ n^2 accessi
- Esegui ⇒ $2n^3$ flop
- Riponi in memoria ⇒ n^2 accessi

accessi < # operazioni f.p.

In conclusione....

$q_{saxpy} \approx \frac{3}{2}$

accessi > # operazioni f.p.

$q_{dot} \approx 1$

accessi = # operazioni f.p.

$q_{gemv} \approx \frac{1}{2}$

accessi < # operazioni f.p.

$q_{gemm} \approx \frac{2}{n}$

accessi << # operazioni f.p.

Il traffico parassita q è un parametro che DIPENDE dal nucleo computazionale di base

(un prodotto tra matrici è più conveniente in termini di traffico parassita)!!

In sintesi per migliorare l'efficienza…

$$\frac{T_S(N)}{T_{flop}(N)} = 1 + \frac{N_{mem} \times t_{mem}}{N_{flop} \times t_{flop}} = 1 + q\frac{t_{mem}}{t_{flop}}$$

BISOGNA
Ridurre il tempo di accesso alla memoria

$$\frac{T_S(N)}{T_{flop}(N)} = 1 + q\frac{t_{mem}}{t_{flop}}$$

numero di accessi alla memoria

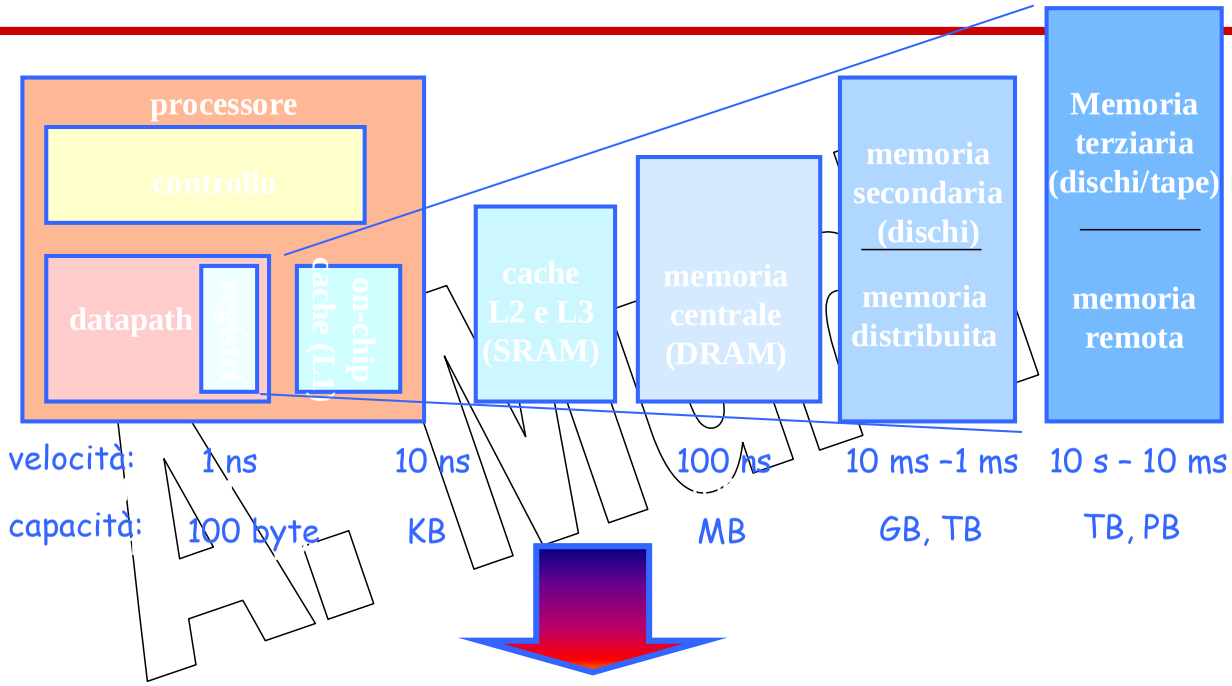
Ridurre il tempo di accesso alla memoria

Principio di Località dei dati

Utlizzando la memoria "gerarchica"

Rivisitazione degli algoritmi al fine di…

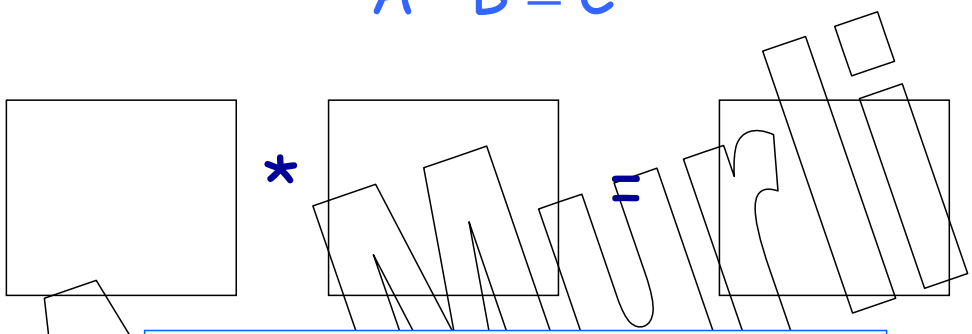
…Utilizzare la localita' dei dati …



progettare sw che riusi i dati nei "Livelli alti della memoria"

Esempio: il prodotto di due matrici:

$$A * B = C$$



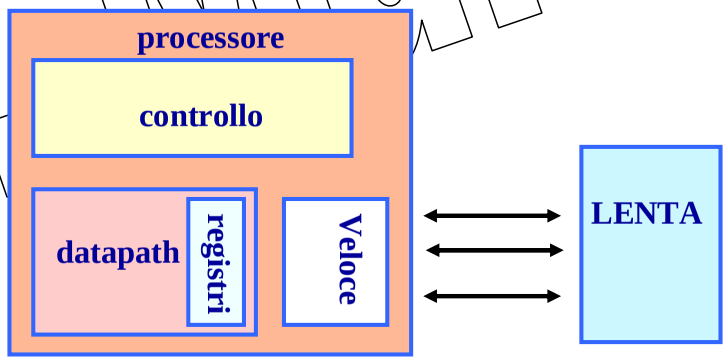
```
for …
    for …
        for …
            c(i,j)=c(i,j)+a(i,k)*b(k,j)
        endfor
    endfor
endfor
```

Esempio: il prodotto di due matrici:

- Calcoliamo il tempo richiesto dall'algoritmo matricexmatrice solo per gli accessi in memoria se rorganizziamo le operazioni
  - In termini di saxpy
  - In termini di gemv
  - In termini di gemm

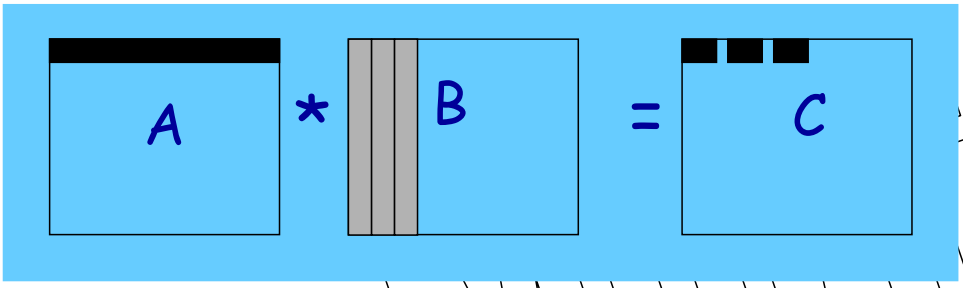
Alcune Ipotesi:

1. l'ambiente di calcolo ha 2 livelli di memoria
   1. Memoria "veloce": $t_{mem} = t_{flop}$ → registri
   2. Memoria "lenta" : $t_{mem} / t_{flop} = 10^{-1}$ → memoria principale



Esempio: A,B,C matrici quadrate di dimensione 10

Strategia I



Ciascun elemento di C e' ottenuto da 1 saxpy

(trasferimento memoria Lenta-Veloce I riga di A e I colonna di B)

La prima riga di C e' ottenuta mediante 10 saxpy

(trasferimento memoria Lenta-Veloce 10 righe di A e 10 colonne di B)

Gli elementi di C sono ottenuti mediante 100 saxpy

(trasferimento memoria Lenta-Veloce di 10 righe di A e 10 colonne di B, per 10 volte!)

Strategia I:

... quanto costa in termini di

ACCESSI alla memoria ?

Consideriamo gli accessi alla memoria

$$\frac{T_S(N)}{T_{flop}(N)} = 1 + q\frac{t_{mem}}{t_{flop}}$$

Sono necessarie 100 saxpy per ottenere C:

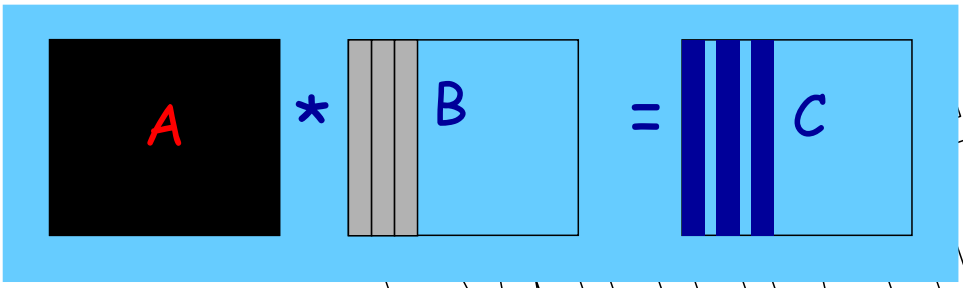
$$100 x q_{saxpy} x \frac{t_{mem}}{t_{flop}} = 150 \times 10^{-1}\, sec = 15\, sec$$

10^{-1}

La strategia I richiede 15 sec per gli accessi alla memoria!

Esempio: A,B,C matrici quadrate di dimensione 10

Strategia II



- Ciascuna colonna di C e' ottenuta mediante un prodotto matrice vettore (GEMMV)

(trasferimento memoria Lenta-Veloce di 1 colonna di B e tutta la matrice A)

- Gli elementi di C sono ottenuti mediante 10 GEMV

(trasferimento memoria Lenta-Veloce di 10 colonne di B e 10volte A)

Strategia II:

...quanto costa in termini di

ACCESSI alla memoria ?

$$\frac{T_S(N)}{T_{flop}(N)} = 1 + q\frac{t_{mem}}{t_{flop}}$$

Sono necessarie 10 GEMV per ottenere C ...

$$10 \times q_{gemmv} \times \frac{t_{mem}}{t_{flop}} = 10 \times \frac{1}{2} \times 10^{-1} sec = 0.5 sec$$

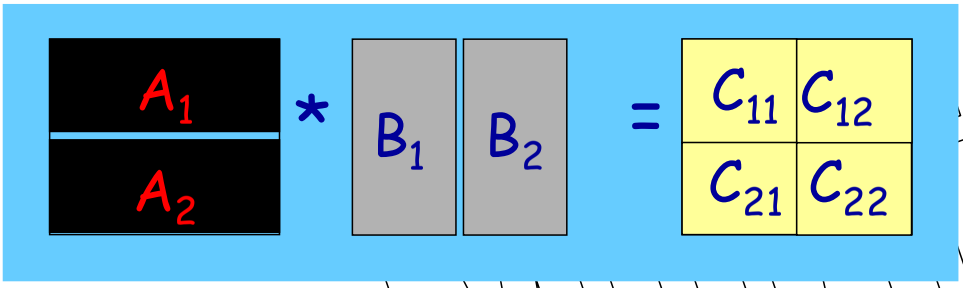
$q_{gemmv} \approx 1/2$

$\frac{t_{mem}}{t_{flop}} = 10^{-1}$

La strategia II richiede 0.5 sec per gli accessi alla memoria!

Esempio: A,B,C matrici quadrate di dimensione 10

Strategia III



- Ciascun blocco di C e' ottenuto mediante 1 prodotto matrice matrice (GEMM)

(trasferimento memoria Lenta-Veloce di 1 blocco di A e di 1 blocco di B)

- Gli elementi di C sono ottenuti mediante 4 GEMM

(trasferimento memoria Lenta-Veloce di 4 blocchi di A e di 4 blocchi di B)

Strategia III:

quanto costa in termini di

ACCESSI alla memoria ?

$$\frac{T_S(N)}{T_{flop}(N)} = 1 + q\frac{t_{mem}}{t_{flop}}$$

Sono necessarie 4 GEMM per ottenere C dunque...

$$4 \times q_{gemm} \times \frac{t_{mem}}{t_{flop}} = 4 \times \frac{1}{10} \times 10^{-1} = 4 \times 10^{-2}\ \text{sec}$$

$q_{gemm} \approx 1/10$; $\frac{t_{mem}}{t_{flop}} \approx 10^{-1}$

La strategia III richiede 4×10^{-2} sec per gli accessi alla memoria!

In conclusione

Strategia I (saxpy)

15 sec per gli accessi alla memoria!

Strategia II (GEMV)

1 sec per gli accessi alla memoria!

Strategia III (GEMM)

4×10^{-2} sec per gli accessi alla memoria!

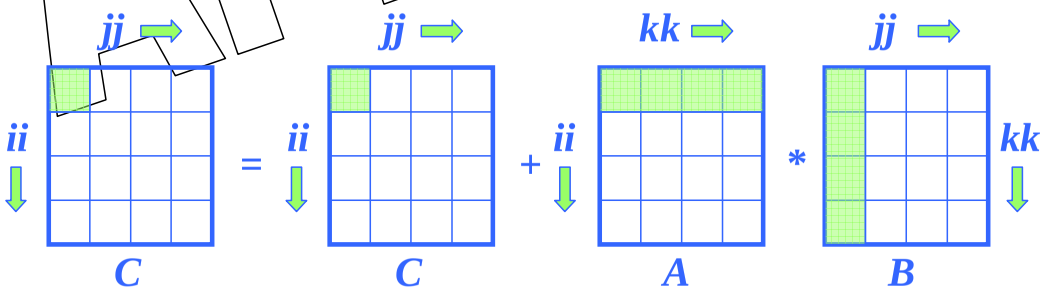
riorganizzare gli algoritmi a blocchi (nulceo computazionale Matrice -Matrice)

In conclusione

Algoritmi a blocchi

Riuso "ottimale" dei dati nei

"Livelli alti della memoria"



BLAS:

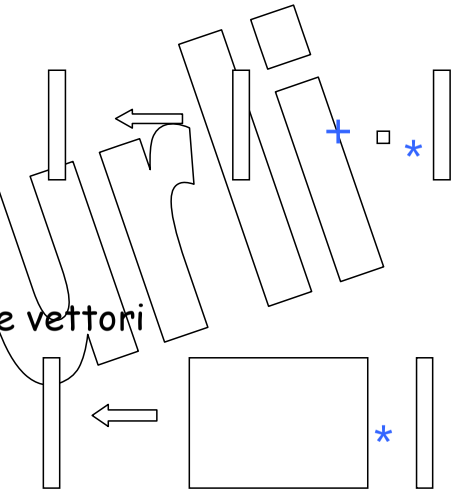
Basic Linear Algebra Subroutines:

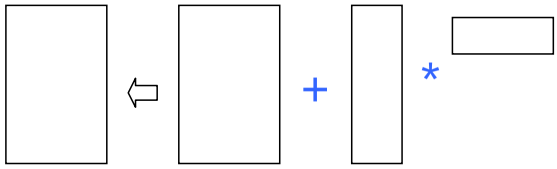
Libreria di software matematico per l'esecuzione di operazioni di base del calcolo matriciale che

ottimizza gli accessi alla memoria

BLAS

- BLAS 1:
  - Operazioni di base tra vettori
    - Somma
    - Aggiornamento
    - Norma
    - Prodotto scalare
    - .....
- BLAS 2
  - Operazioni di base tra matrice e vettori
  - Prodotto matrice –vettore
  - Aggiornamento
  - .....
- BLAS 3
  - Operazioni di base tra matrici
    - Prodotto tra matrici
    - Aggiornamento
    - Norma
    - Somma
    - .....

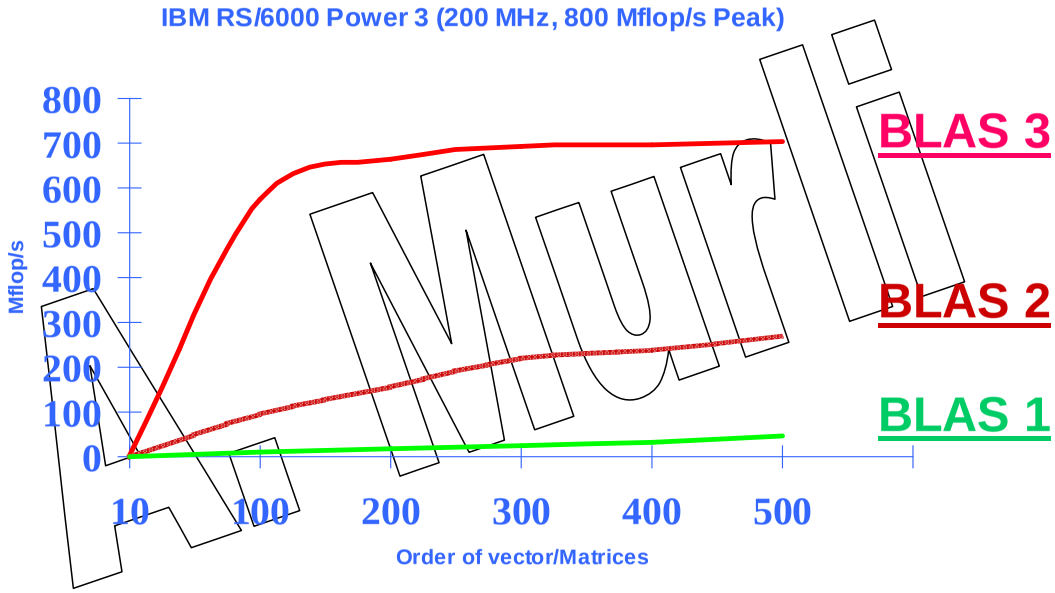




Confronto BLAS1, BLAS2, BLAS3

- BLAS 1:
  - Ottimizza le operazioni tra vettori (loop unrolling)
- BLAS 2
  - Ottimizza il riuso dei dati che risiedono nei registri (riduce lo spostamento dei dati dalla cache ai registri)
- BLAS 3
  - Ottimizza il riuso dei dati che risiedono nella cache (riduce lo spostamento dei dati dalla memoria centrale alla cache)

Un confronto tra BLAS1, BLAS2, BLAS3



Solo BLAS 3 riesce a raggiungere

la peak performance del processore IBM /RS 6000

perché ottimizza l'uso di tutti i livelli della memoria gerarchica

"As machines become more powerful, the efficiency of algorithms grows more important, not less."

Nick Trefethen, 1997

Quanto più potente è l'ambiente di calcolo tanto più diventa complesso e difficile lo sviluppo di software in grado di sfruttarne appieno le capacità.

FINE LEZIONE

Esercitazione

Come sviluppare software efficiente per il calcolo matriciale ?

Case study n. 1: Prodotto matrice x matrice

Calcolo di

$A \cdot B = C$

$A, B, C \in \Re^{n \times n}$

```
for _ = 1:n;
  for _ = 1:n;
    for _ = 1:n;
      C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
    end
  end
end
```

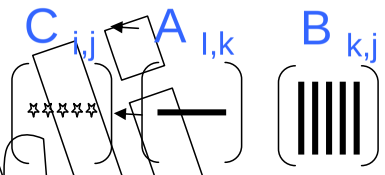
Indipendentemente dall' ordine dei cicli abbiamo sempre $O(n^3)$ operazioni f.p.

6 Varianti della moltiplicazione di matrici

```
for _ = 1:n;
  for _ = 1:n;
    for _ = 1:n;
      C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
    end
  end
end
```

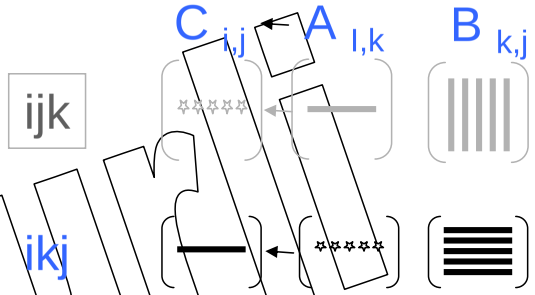
6 Varianti della moltiplicazione di matrici

```
for i = 1:n;
   for j = 1:n;
      for k = 1:n;
         C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
      end
   end
end
```



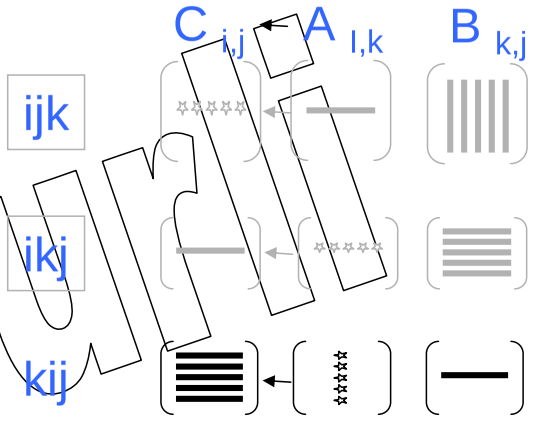
6 Varianti della moltiplicazione di matrici

```
for i = 1:n;
   for k = 1:n;
      for j = 1:n;
         C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
      end
   end
end
```



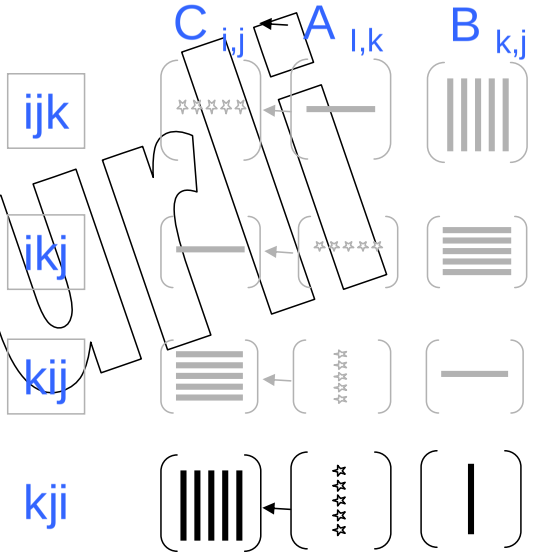
6 Varianti della moltiplicazione di matrici

```
for k = 1:n;
   for i = 1:n;
      for j = 1:n;
         C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
      end
   end
end
```



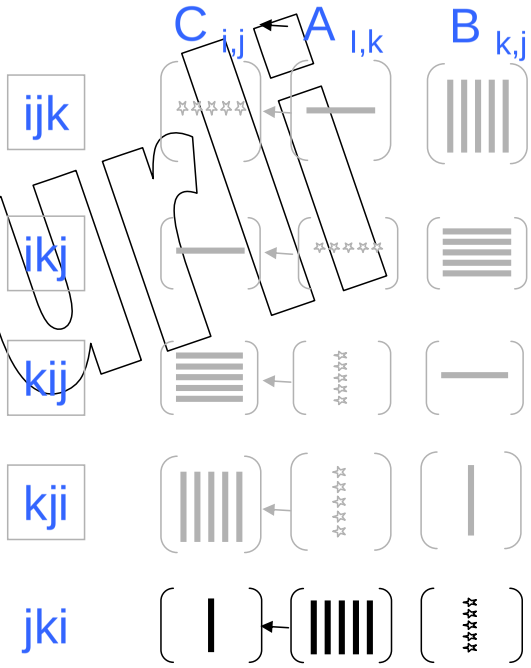
6 Varianti della moltiplicazione di matrici

```
for k = 1:n;
   for j = 1:n;
      for i = 1:n;
         C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
      end
   end
end
```



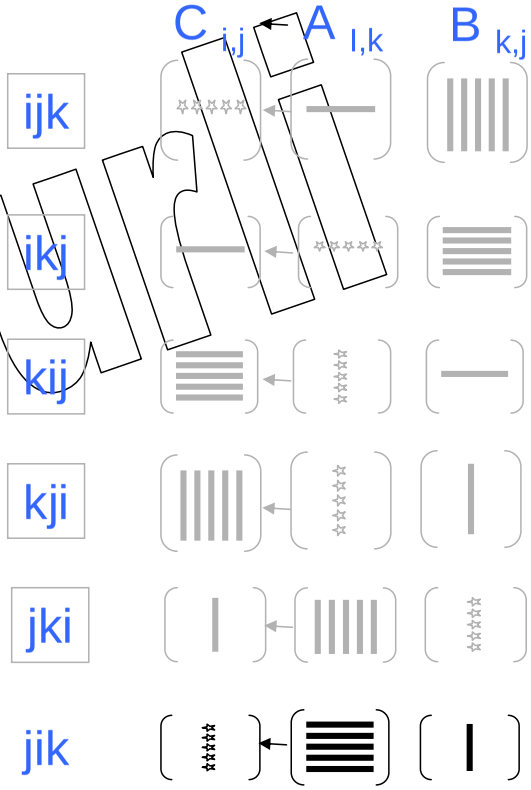
6 Varianti della moltiplicazione di matrici

```
for j = 1:n;
   for k = 1:n;
      for i = 1:n;
         C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
      end
   end
end
```



6 Varianti della moltiplicazione di matrici

```
for j = 1:n;
   for i = 1:n;
      for k = 1:n;
         C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
      end
   end
end
```

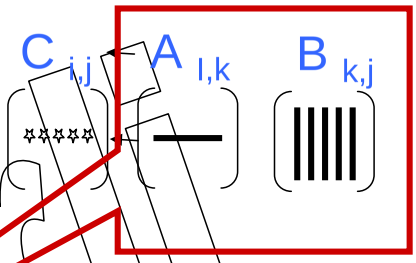


Quale versione implementare?

6 Varianti della moltiplicazione di matrici

```
for i = 1:n;
   for j = 1:n;
      for k = 1:n;
         C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
      end
   end
end
```

ijk



Operazione di base:

prodotto scalare

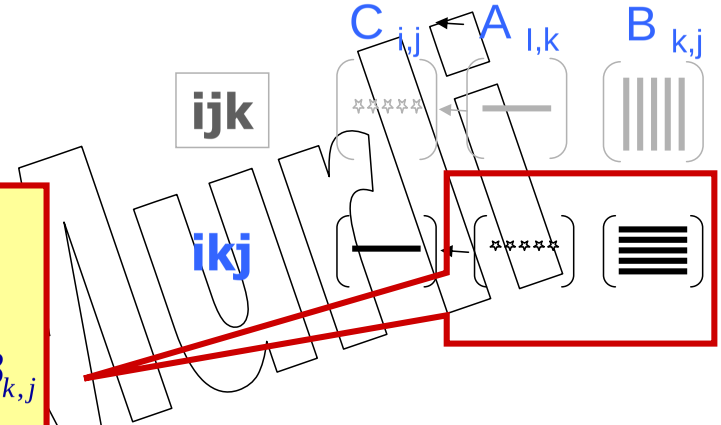
(BLAS1)

della riga i-ma di A e

della colonna j -ma di B

6 Varianti della moltiplicazione di matrici

```
for i = 1:n;
    for k = 1:n;
        for j = 1:n;
            C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
        end
    end
end
```



Operazione di base:

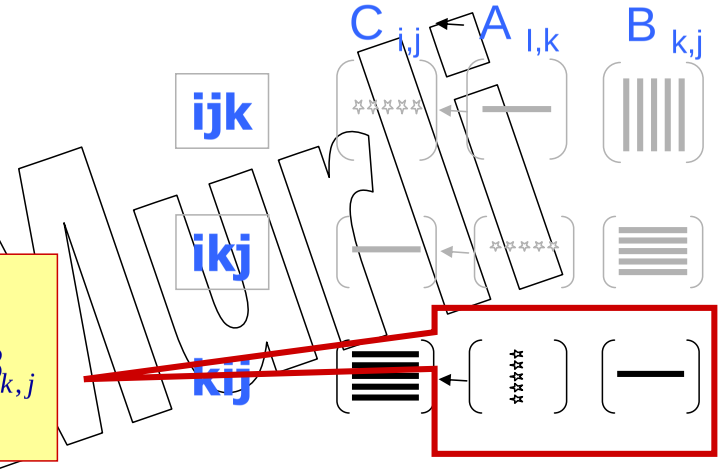
Matrice B x vettore(riga) A_{ik}

(aggiornamento per riga)

(BLAS 2)

6 Varianti della moltiplicazione di matrici

```
for k = 1:n;
    for i = 1:n;
        for j = 1:n;
            C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
        end
    end
end
```



Operazione di base: saxpy

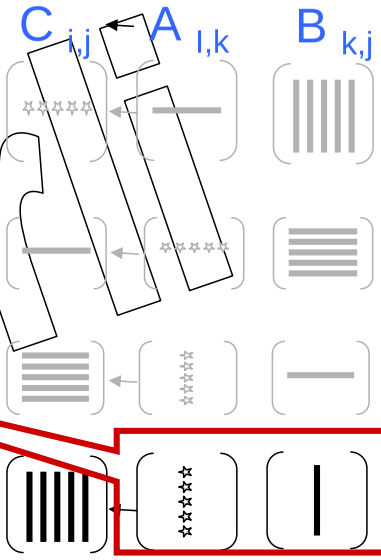
della riga k -ma di B

con lo scalare A_{ik}

(BLAS 1)

6 Varianti della moltiplicazione di matrici

```
for k = 1:n;
   for j = 1:n;
      for i = 1:n;
         C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
      end
   end
end
```



Operazione di base: saxpy

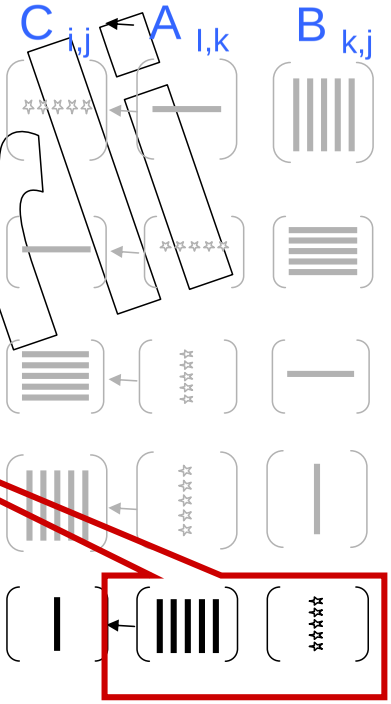
della colonna k -ma di *A*

con lo scalare B_{kj}

(BLAS 1)

6 Varianti della moltiplicazione di matrici

```
for j = 1:n;
   for k = 1:n;
      for i = 1:n;
         C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
      end
   end
end
```



Operazione di base:

Matrice x vettore

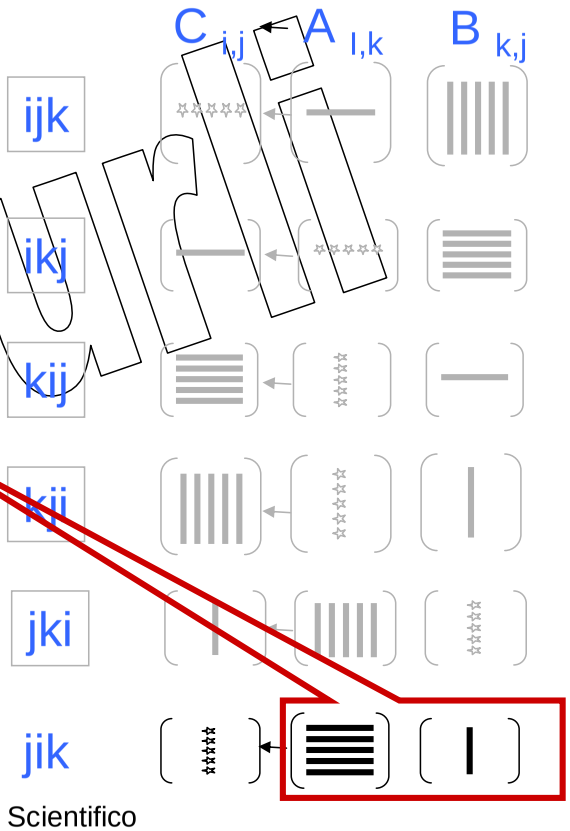
(aggiornamento per colonna)$_j$

(BLAS 2)

6 Varianti della moltiplicazione di matrici

```
for j = 1:n;
   for i = 1:n;
      for k = 1:n;
         C_{i,j} ← C_{i,j} + A_{i,k} B_{k,j}
      end
   end
end
```

Operazione di base:
prodotto scalare
(BLAS1)
della riga i-ma di *A* e
della colonna j -ma di *B*



Ciascuna permutazione degli indici implica l'esecuzione di una diversa operazione di base tra vettori (BLAS 1) o tra vettori e matrici (BLAS 2)

Case study 2

Risoluzione di

$$Ax = b$$

$$A \in \Re^{n \times n}, \quad x, b \in \Re^{n}$$

Mediante
L' algoritmo di
eliminazione di Gauss

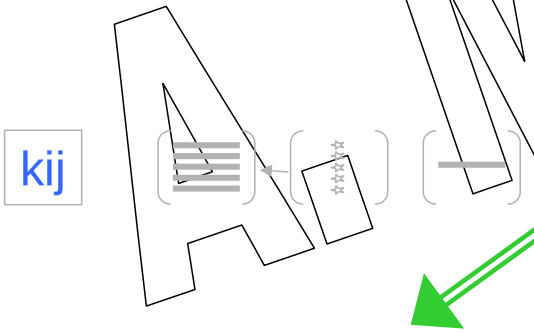
```
for k = 1 to n-1
    for i = k+1 to n
        a(i,k) = a(i,k)/a(k,k)
        for j = k+1 to n
            a(i,j) = a(i,j) – a(i,k) * a(k,j)
        endfor
    endfor
endfor
```

Indipendentemente dall' ordine dei cicli abbiamo sempre $O(n^3)$ operazioni f.p.

Versione k i j :

Operazione di base?

$$\underline{a}^{i} = \underline{a}^{i} - a_{ik} \cdot \underline{a}^{k}$$




```
for k = 1 to n-1
    for i = k+1 to n
        a(i,k) = a(i,k)/a(k,k)
        for j = k+1 to n
            a(i,j) = a(i,j) – a(i,k) * a(k,j)
        endfor
    endfor
endfor
```

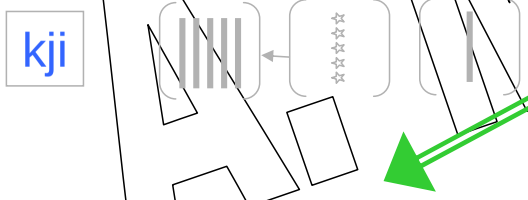
Aggiornamento di un vettore (**riga**) mediante il prodotto di uno scalare per un vettore (**O(n)** op. f.p.)

Invertendo i cicli ……Versione (k j i):

Operazione di base?

$$\underline{a}^j = \underline{a}^j - a_{kj} \cdot \underline{a}^i$$

```
for k = 1 to n-1
    for j = k+1 to n
        for i = k+1 to n
            a(i,j) = a(i,j) – a(i,k) * a(k,j)
        endfor
```



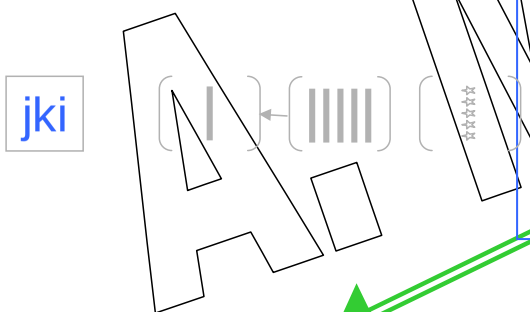
Aggiornamento di un vettore (**colonna**) mediante il prodotto di uno scalare per un vettore (**O(n)** op. f.p.)

Invertendo i cicli ….Versione (j k i):

Operazioni di base?

$$\underline{a}^j = \underline{a}^j - M \cdot \underline{a}^k$$

```
for j = 1 to n-1
    for k = … to n
        ……
        for i = k+1 to n
            a(i,j) = a(i,j) – a(i,k) * a(k,j)
        endfor
    endfor
```



Aggiornamento di un vettore mediante il prodotto di una matrice per un vettore (**O(n²)** op. f.p.)

Quale versione dello stesso algoritmo conviene implementare?

In generale,

la scelta dipende dall'ambiente di calcolo …