

Calcolo Parallelo e Distribuito

matriceXvettore

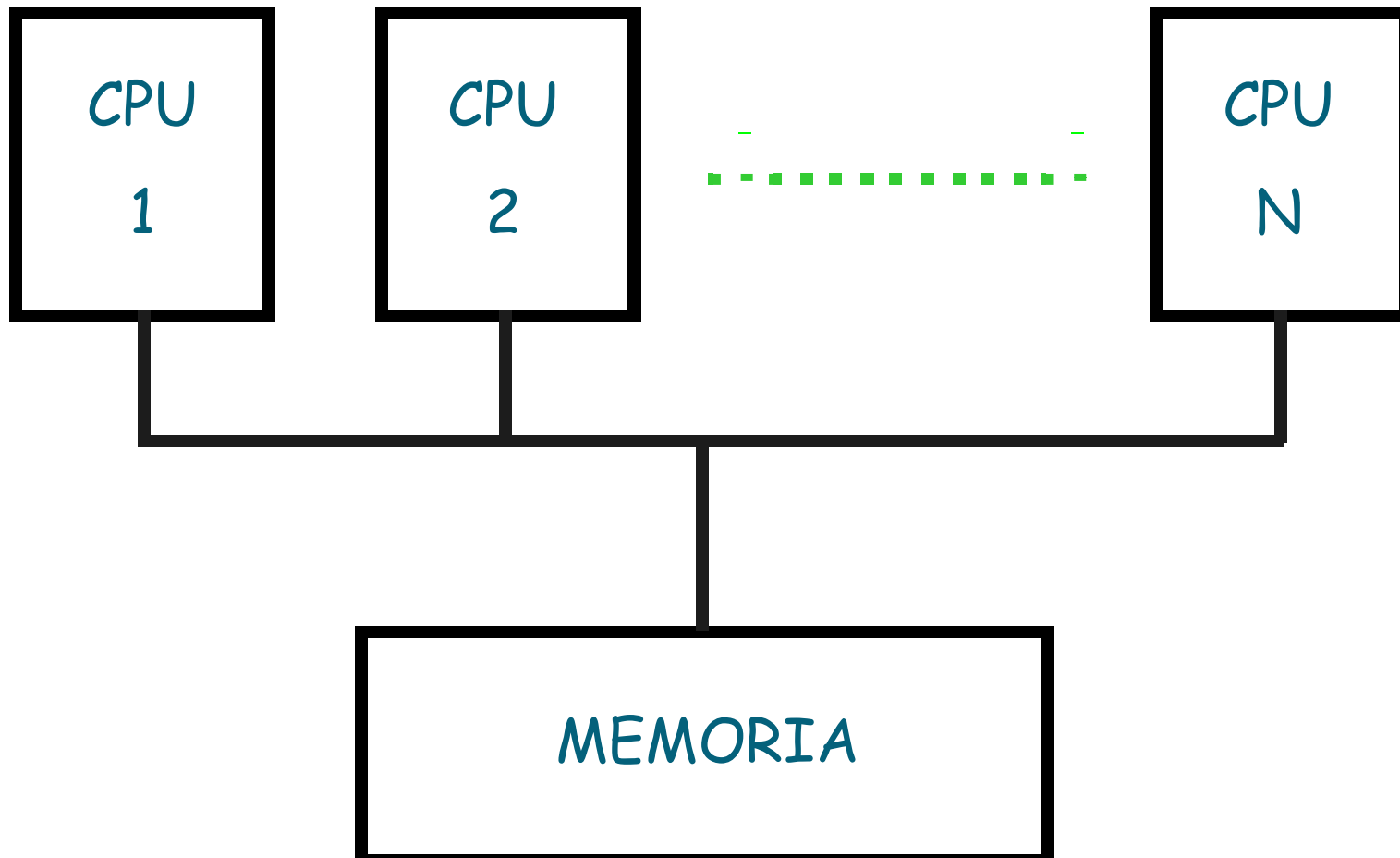
implementazione di ambiente multicore - openMP

Docente: Prof. L. Marcellino

Tutor: Prof. P. De Luca

Schema Calcolatori

MIMD a memoria **condivisa** (shared-memory)

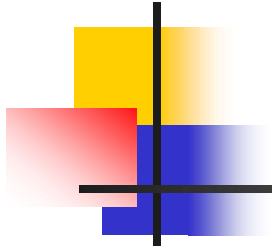


PROBLEMA: Prodotto Matrice-Vettore

Progettazione
di un algoritmo parallelo
per architettura MIMD

per il calcolo del prodotto
di una matrice A pr un vettore b :

Matrice A : N righe, M colonne, Vettore b : M elementi



OpenMP

ulteriori costrutti

algoritmo parallelo

matrice per vettore

1 strategia

Direttive

La direttiva

Si usano per creare un team e stabilire quali istruzioni devono essere eseguite in parallelo e come queste devono essere distribuite tra i thread del team creato

```
#pragma omp costrutto [clause], [clause] ... new-line
```

oltre al costrutto **parallel** prevede anche:

- **tre** tipi di costrutti detti **WorkSharing** perché si occupano della distribuzione del lavoro al team di thread: **for**, **sections**, **single**
- Anche all'uscita da un costrutto work-sharing è sottintesa una barriera di sincronizzazione, se non diversamente specificato dal programmatore.

Direttive

- Il costrutto *for* specifica che le iterazioni del ciclo contenuto debbano essere distribuite tra i thread del team (secondo un ordine che non specificherò in dettaglio, come ho fatto con la somma; ma userò le potenzialità di OpenMP!)

```
#pragma omp for [clause], [clause] ... new-line  
{  
}  
clause: private(list)  
firstprivate(list)  
lastprivate(list)  
reduction(operator: list)  
schedule(kind[, chunk_size])  
collapse(n)  
ordered  
nowait
```

schedule
solo per questo
costrutto

Direttive

- Il costrutto ***sections*** conterrà un insieme di costrutti *section* ognuno dei quali verrà eseguito da un thread del team



Le diverse
sezioni
devono
poter essere
eseguite in
ordine
arbitrario

```
#pragma omp sections [clause], [clause] ... new-line
{
  [#pragma omp section new-line]

  [#pragma omp section new-line]
  ...
}
clause: private(list)
firstprivate(list)
lastprivate(list)
reduction(operator: list)
nowait
```



Rischio di
sbilanciamento
del carico

Direttive

- Il costrutto **single** specifica che il blocco di istruzioni successivo verrà eseguito da un solo thread QUALSIASI del team

```
#pragma omp single [clause], [clause] ... new-line  
{  
}  
clause: private(list)  
firstprivate(list)  
copyprivate(list)  
nowait
```

Gli altri
thread
attendono
che questo
termini la sua
porzione di
codice

- Tutti i costrutti **WorkSharing** possono essere combinati con il costrutto **parallel**, e le clausole ammesse sono l'unione di quelle ammesse per ognuno.

Direttive ... ulteriori costrutti

- Il costrutto ***master*** specifica che il blocco di istruzioni successivo verrà eseguito dal solo master thread

```
#pragma omp master  
{  
}  
}
```

Non sono sottintese
barriere di
sincronizzazione né
all'ingresso né all'uscita
del costrutto!

Direttive

- Il costrutto ***barrier*** forza i thread ad attendere il completamento di tutte le istruzioni precedenti da parte di tutti gli altri

`#pragma omp barrier`

Al momento del barrier
(implicito o esplicito) si
crea una vista consistente
dei dati dei thread

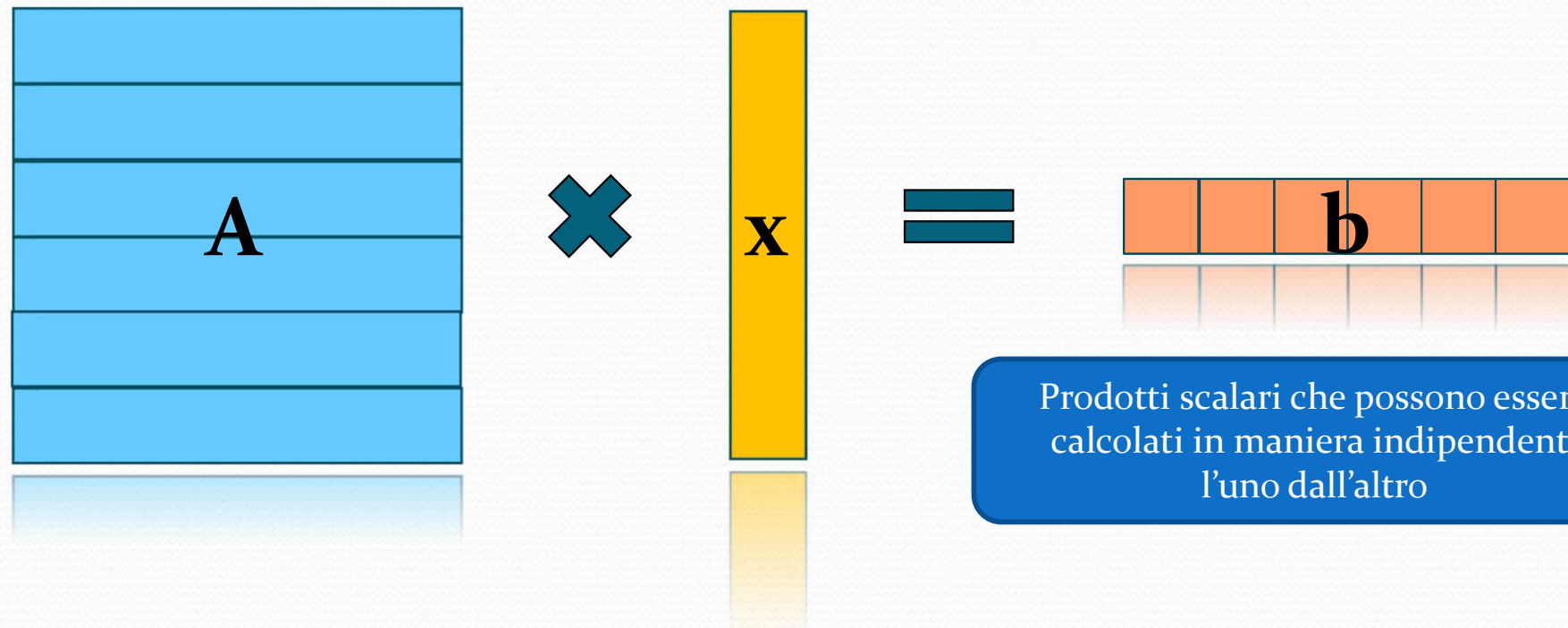
- ... ce ne sono tanti altri!!!

Algoritmo per il prodotto Matrice per vettore in ambiente MIMD_SM

Problema

Prodotto $Ax=b$, dove:

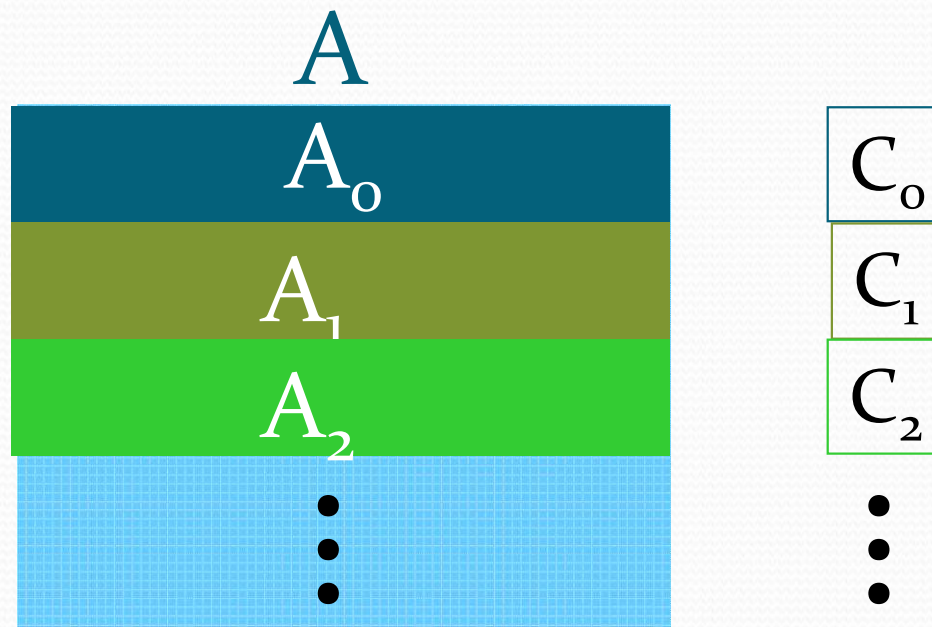
$$A \in \mathbb{R}^{n \times m}, x \in \mathbb{R}^m, b \in \mathbb{R}^n$$



I STRATEGIA: In generale

I passo: decomposizione del problema

I p core si **occuperanno** di
BLOCCHI di RIGHE della matrice A

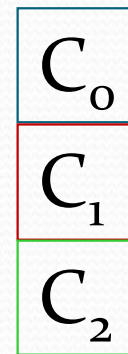
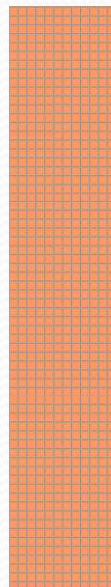


I STRATEGIA: In generale

I passo: decomposizione del problema

I p core si **occuperanno** di
TUTTO il vettore x

x



•
•
•

Esempio: Prodotto Mat-Vet

Programma chiamante

```
main()  
begin
```

*dichiarazione delle variabili
allocazione della memoria
assegnazione di valori alle dimensioni della matrice (**n,m**)
assegnazione dei valori alla matrice **A** ed al vettore **x**
inizializzazione del vettore risultato **b***

```
b:=matxvet(m,n,x,A)
```

```
for i:=0 to n-1  
  stampa b[i]  
end for
```

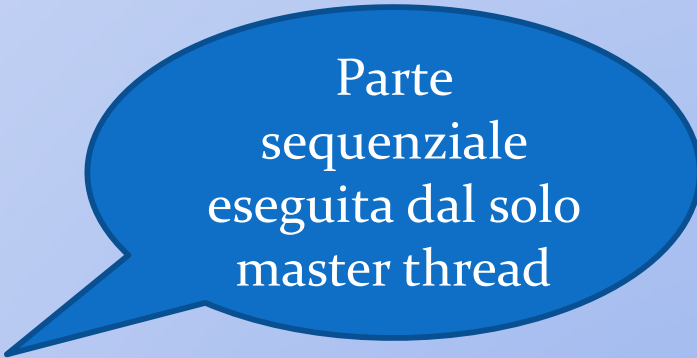
```
end
```

Funzione che realizza il
prodotto della matrice **A** per il
vettore **x**

Esempio: Prodotto Mat-Vet

```
#include <omp.h>
#include <math.h>
#include <stdlib.h>
...

double * matxvet(int m, int n, double *x, double **A){
    int i,j;
    double *b;
    ...
    /*allocazione memoria per b*/
    ...
}
```



Parte
sequenziale
eseguita dal solo
master thread

Esempio: Prodotto Mat-Vet

```
#include <omp.h>
#include <math.h>
#include <stdio.h>
...
double b, x, double **A){
    double b;
    ...
    /*allocazione memoria per b*/
    ...
    #pragma omp parallel for default(none) shared(m,n,A,x,b) private(i,j)
    for (i=0; i<n; i++){
        for (j=0; j<m; j++)
            b[i] += A[i][j]*x[j];
    }
}
```

Creazione di un
team di thread

i-mo prodotto
scalare

Esempio: Prodotto Mat-Vet

```
#include <omp.h>
#include <math.h>
#include <stdlib.h>
...
```

```
double * matxvet(int m, int n, double *x, double *a,
    int i,j;
    double *b;
    ...
    /*allocazione memoria per b*/
    ...
    #pragma omp parallel for default(none) shared(m,n,A,x,b) private(i,j)
    for (i=0; i<n; i++){
        for (j=0; j<m; j++)
            b[i] += A[i][j]*x[j];
    }
}
```

Distribuzione
delle iterazioni
del for tra i
thread

Con la clausola
schedule si può
scegliere la
distribuzione

i-mo prodotto
scalare

Esempio: Prodotto Mat-Vet

```
#include <omp.h>
#include <math.h>
#include <stdlib.h>
...
double * matxvet(
    int i,j;
    double *b;
    ...
    /*allocazione memoria per b*/
    ...
    #pragma omp parallel for default(none) shared(m,n,A,x,b) private(i,j)
    for (i=0; i<n; i++){
        for (j=0; j<m; j++)
            b[i] += A[i][j]*x[j];
    }
}
```

Sarà il
programmatore a
stabilire cosa è
condiviso e cosa
non lo è

i-mo prodotto
scalare

Esempio: Prodotto Mat-Vet

```
#include <omp.h>
#include <math.h>
#include <stdlib.h>
...
```

```
double * matxvet(int m, int n, double *x, double **A){
    int i,j;
    double *b;
    ...
    /*allocazione memoria per b*/
    ...
    #pragma omp parallel for default(none) shared(m,n,A,x,b) private(i,j)
    for (i=0; i<n; i++){
        for (j=0; j<m; j++)
            b[i] += A[i][j]*x[j];
    }
}
```

Variabili
condivise tra
i thread

Tutti devono
poterle
leggere

Se fossero private
m, n, A e x
verrebbero
de-inizializzate

i-mo prodotto
scalare

Esempio: Prodotto Mat-Vet

```
#include <omp.h>
#include <math.h>
#include <stdlib.h>
...
```

```
double * matxvet(int m, int n, double *x, double **A){
    int i,j;
    double *b;
    ...
    /*allocazione memoria per b*/
    ...
    #pragma omp parallel for default(none) shared(m,n,A,x,b) private(i,j)
    for (i=0; i<n; i++){
        for (j=0; j<m; j++)
            b[i] += A[i][j]*x[j];
    }
}
```

Variabili
condivise tra
i thread

Tutti devono
poterle
leggere

Se fosse privata b,
non sarebbe
accessibile fuori
dalla regione
parallela

i-mo prodotto
scalare

Esempio: Prodotto Mat-Vet

```
#include <omp.h>
#include <math.h>
#include <stdlib.h>
...
```

```
double *
```

**Comportamento
imprevedibile**

```
...
/*allocazione memoria per b*/
```

```
...
#pragma omp parallel for default(none) shared(m,n,A,x,b) private(i,j)
```

```
for (i=0; i<n; i++){
```

```
    for (j=0; j<m; j++){
```

```
        b[i] += A[i][j]*x[j];
```

```
    }
```

```
}
```

Se i e j fossero
condivise, un
thread potrebbe
modificare il
contatore per un
altro

Variabili
private per
ogni thread

i-mo prodotto
scalare

Esempio: Prodotto Mat-Vet

```
#include <omp.h>
#include <math.h>
#include <stdlib.h>
...

double * matxvet(int m, int n, double *x, double **A){
    int i,j;
    double *b;
    ...
    /*allocazione memoria per b*/
    ...
    #pragma omp parallel for default(none) shared(m,n,A,x,b) private(i,j)
    for (i=0; i<n; i++){
        for (j=0; j<m; j++)
            b[i] += A[i][j]*x[j];
    }
    /*-- Fine del for parallelo--*/
    return b;
}
```

Non ci vogliono
le parentesi per
le istruzioni della
direttiva

Parte
sequenziale
eseguita dal solo
master thread

Proviamo ad implementare la seconda strategia

attenzione alla collezione dei risultati locali