

Calcolo Parallelo e Distribuito

a.a. 2021-2022

Prodotto Matrice-Vettore
approfondimenti
parte 1

Docente: Prof. L. Marcellino

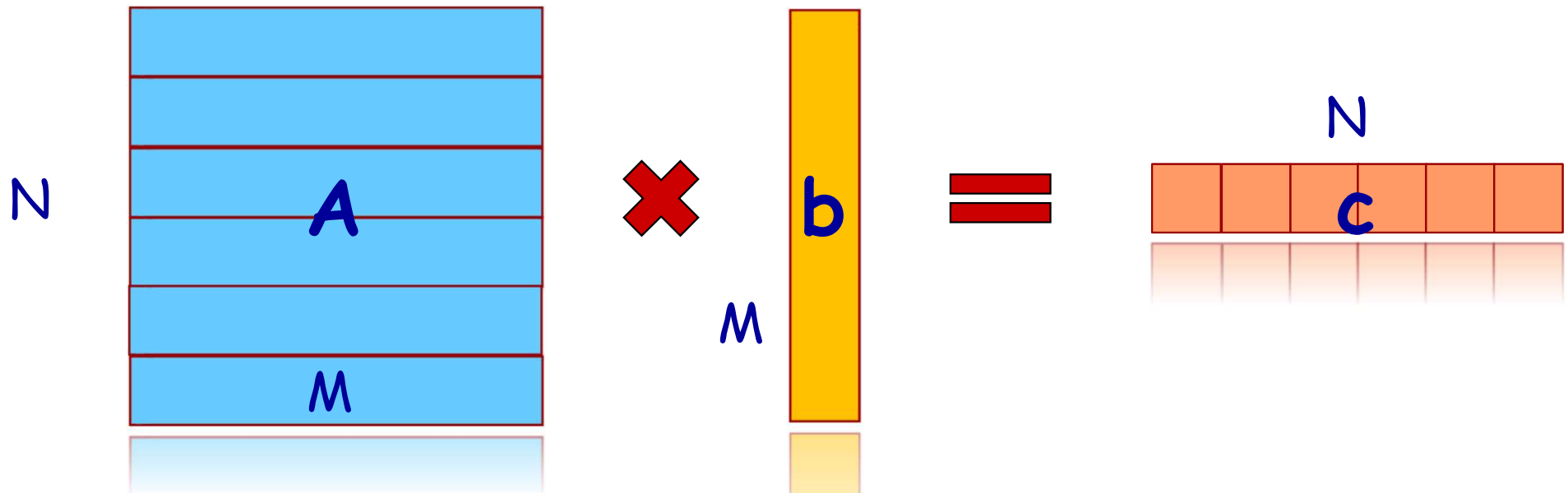
PROBLEMA: Prodotto Matrice-Vettore

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

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

Complessità computazionale dell'algoritmo sequenziale

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$



In sequenziale, **N prodotti scalari** di lunghezza **M** .

Per fare 1 prodotto scalare di lunghezza M , devo fare:

M molt + $(M-1)$ add

Complessità computazionale dell'algoritmo sequenziale

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

In sequenziale, N prodotti scalari di lunghezza M , cioè:

$$N[M \text{ molt} + (M-1) \text{ add}]$$

$$\text{molt} \sim \text{add}$$

$$T_1(N \times M) = N[2M-1] t_{\text{calc}}$$

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 :

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

I STRATEGIA

Decomposizione 1
matrice A in
BLOCCHI di RIGHE

Calcolo **di speedup ed efficienza**
(def classica)

in ambiente **MIMD-DM**

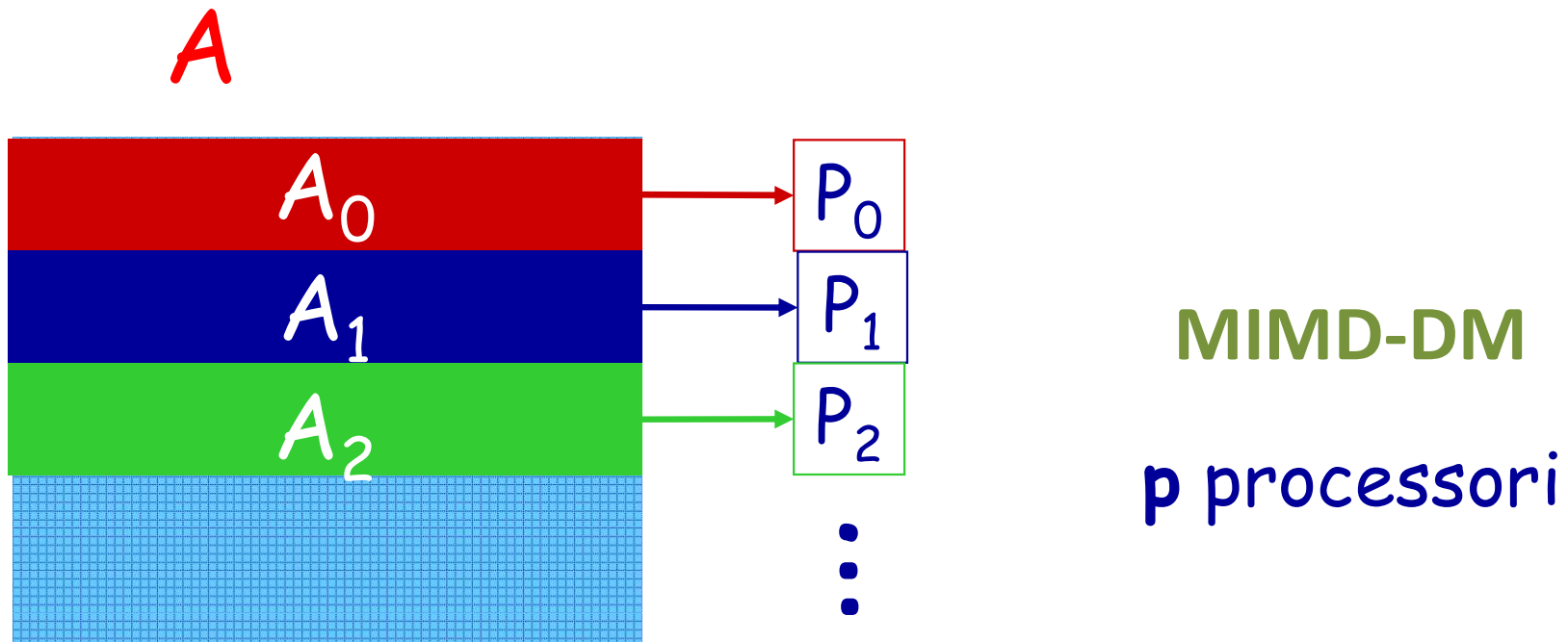
e

in ambiente **MIMD-SM**

I Strategia: speed-up/efficienza (**def classica**)

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

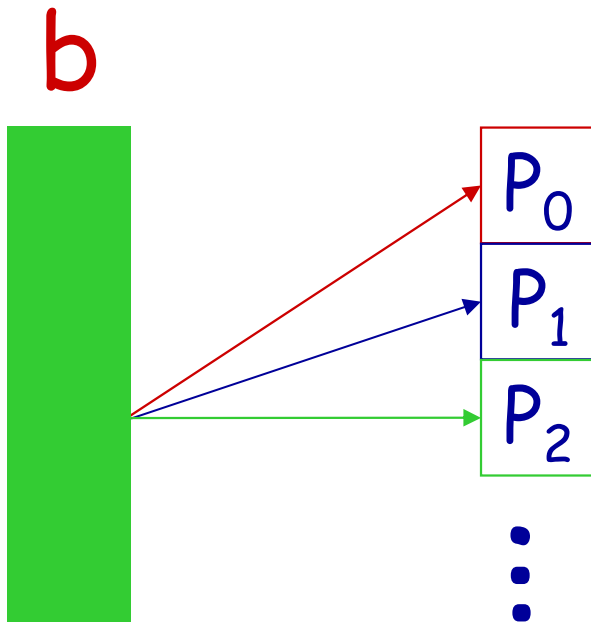
$$T_1(N \times M) = N[2M-1] t_{\text{calc}}$$



I Strategia: speed-up/efficienza (**def classica**)

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

$$T_1(N \times M) = N[2M-1] t_{\text{calc}}$$



MIMD-DM

p processori

I Strategia: speed-up/efficienza (**def classica**)

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

MIMD-DM

p processori:

$$\dim[A_i] = (N/p) \times M; \quad \dim[b] = M$$

Tutti contemporaneamente, N/p prodotti scalari di lunghezza M ,
cioè:

$$T_p(N \times M) = N/p [2M-1] \dagger_{\text{calc}}$$

I Strategia: speed-up/efficienza (**def classica**)

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

MIMD-DM

p processori

$$\begin{aligned} S_p(N \times M) &= T_1(N \times M) / T_p(N \times M) = \\ &= N[2M-1] / (N/p [2M-1]) = p \quad N[2M-1] / (N[2M-1]) = p \end{aligned}$$

$$\begin{aligned} Oh &= p T_p(N \times M) - T_1(N \times M) = \\ &= p(N/p [2M-1]) \dagger_{\text{calc}} - N[2M-1] \dagger_{\text{calc}} = 0 \end{aligned}$$

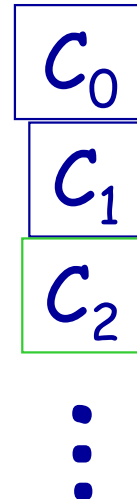
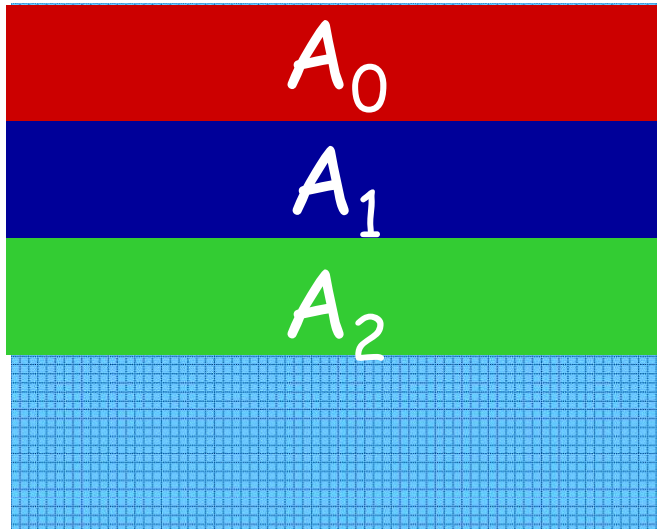
$$E_p(N \times M) = S_p(N \times M) / p = p/p = 1$$

Se si sceglie di non unire il risultato finale!

I Strategia: speed-up/efficienza (**def classica**)

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

$$T_1(N \times M) = N[2M-1] t_{\text{calc}}$$



MIMD-SM

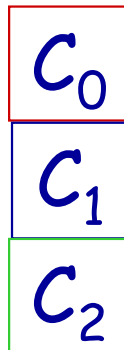
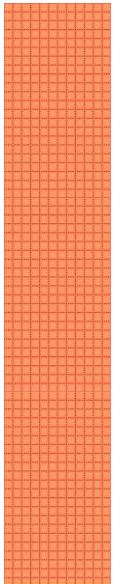
p core

I Strategia: speed-up/efficienza (**def classica**)

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

$$T_1(N \times M) = N[2M-1] t_{\text{calc}}$$

b



⋮

MIMD-SM

p core

I Strategia: speed-up/efficienza (**def classica**)

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

MIMD-SM

p core:

$$\dim[A_i] = (N/p) \times M; \quad \dim[b] = M$$

Tutti contemporaneamente, **N/p prodotti scalari di lunghezza M** ,
cioè:

$$T_p(N \times M) = N/p [2M-1] \mathbf{\dagger}_{\text{calc}}$$

I Strategia: speed-up/efficienza (**def classica**)

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

MIMD-SM

p core

$$\begin{aligned} S_p(N \times M) &= T_1(N \times M) / T_p(N \times M) = \\ &= N[2M-1] / (N/p [2M-1]) = p \quad N[2M-1] / (N[2M-1]) = p \end{aligned}$$

$$\begin{aligned} Oh &= p T_p(N \times M) - T_1(N \times M) = \\ &= p(N/p [2M-1]) \dagger_{calc} - N[2M-1] \dagger_{calc} = 0 \end{aligned}$$

$$E_p(N \times M) = S_p(N \times M) / p = p/p = 1$$

Osservazione:

Come nel caso dell'algoritmo della nomma di N numeri le spedizioni relative alla distribuzione dei dati iniziali (ambiente MIMD-DM)

NON sono contemplate nei conti di speedup, overhead ed efficienza!

Ma quando prendo i tempi del SW si!

I Strategia: **isoefficienza**

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

p core/processors

$O_h = 0 \rightarrow I(p_0, p_1, n_0) = 0/0$ forma indeterminata

Per convenzione l'isoefficienza è posta uguale ad infinito, ovvero posso usare qualunque costante moltiplicativa per calcolare n_1 e quindi controllare la scalabilità dell'algoritmo.

Calcolo **di speedup ed efficienza**
(def Ware Amdahl-generalizzata)

in ambiente **MIMD-DM**

e

in ambiente **MIMD-SM**

I Strategia: speed-up/efficienza (**W-A**)

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

In sequenziale:

$$T_1(N \times M) = N[2M-1] \text{ operazioni}$$

Per calcolare lo speedup con la legge di W-A, la prima domanda che mi devo fare è se per questa strategia di parallelizzazione posso esattamente distinguere la parte parallela
(nella fase di calcolo locale lavorano tutti i processori)
e la parte sequenziale
~~(la collezione dei risultati avviene in maniera sequenziale)~~

SI

$$S_p = \frac{1}{\alpha + (1 - \alpha)/p}$$

I Strategia: speed-up/efficienza (**W-A**)

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

In sequenziale:

$$T_1(N \times M) = N[2M-1] \text{ operazioni}$$

In parallelo:

1 fase (tutta parallela)

Calcolo prodotti parziali

$N/p [2M-1]$ operazioni

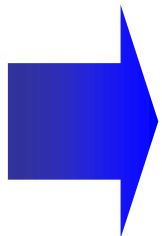


contemporaneamente

fatto da p processori/core

$$p \frac{N}{p} [2M-1]$$

delle $N[2M-1]$ operazioni



$$1 - \alpha = p \frac{N / p [2M - 1]}{N [2M - 1]} = 1 \Rightarrow \frac{1 - \alpha}{p} = \frac{1}{p}$$

I Strategia: speed-up/efficienza (**W-A**)

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

In sequenziale:

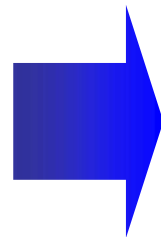
$$T_1(N \times M) = N[2M-1] \text{ operazioni}$$

In parallelo:

... e basta

$$\alpha = 0$$

$$\frac{1 - \alpha}{p} = \frac{1}{p}$$



$$S(p) = \frac{1}{\frac{1}{p}} = p$$



Message Passing Interface

MPI

topologie virtuali

Decomposizione 1

matrice A in

BLOCCHI di RIGHE


```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char *argv[])
{
    int menum, nproc;
    int colonne, righe; //dimensioni della griglia 1D
    int coord[2];
    MPI_Status status;
    MPI_Comm grigliar;
    ...
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &menum);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);
```

Griglia no-periodica-no reorder (3,1):

P_0

P_1

P_2

...

```
// lettura dimensioni, controlli, allocazione
```

```
// lettura matrice e vettore
```

```
/*CREAZIONE DELLA GRIGLIA DI PROCESSORI*/
```

```
crea_griglia(&grigliar,menum,nproc,righe,colonne) ;
```

...

```
void crea_griglia (MPI_Comm *grigliar, int menum, int nproc, int  
    riga, int col)  
{  
    int dim, *ndim, reorder, *period, vc[2];  
    dim = 2;  
    ndim = (int*) calloc (dim, sizeof(int));  
    ndim[0] = 3; // o un qualunque numero > 1  
    ndim[1] = 1;  
    period = (int*) calloc (dim, sizeof(int));  
    period[0] = period [1] = 0;  
    // la periodicità non è necessaria  
    reorder = 0; //NO  
  
    // creazione griglia  
    MPI_Cart_create(MPI_COMM_WORLD, dim, ndim, period, reorder, grigliar);  
  
    return;  
}
```



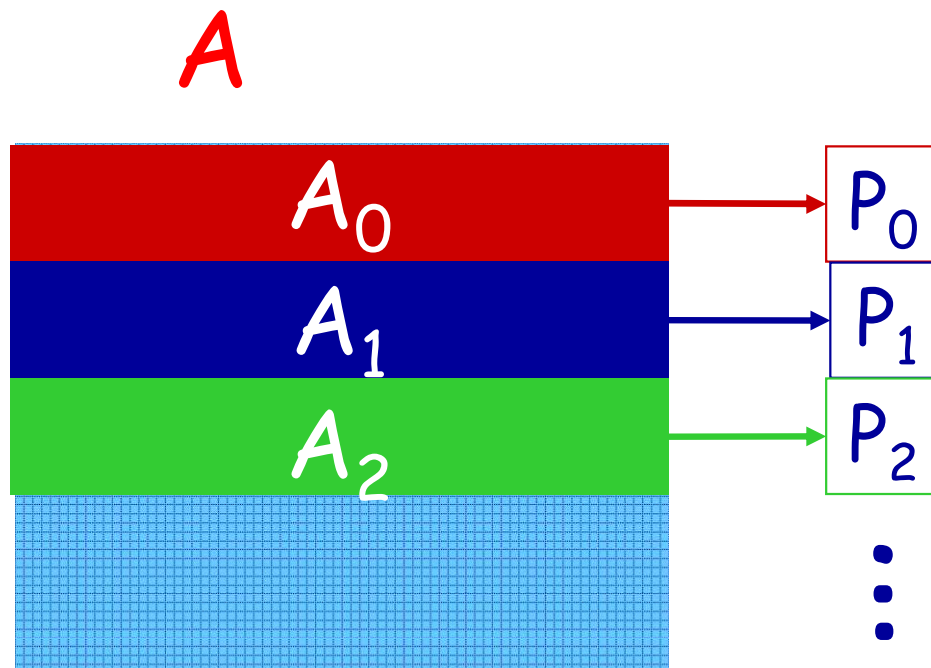
```
...  
// lettura dimensioni, controlli, allocazione  
// lettura matrice e vettore  
  
/*CREAZIONE DELLA GRIGLIA DI PROCESSORI*/  
crea_griglia(&grigliar,menum,nproc,righe,colonne);  
  
/*DISTRIBUZIONE DATI*/  
//invio dimensioni matrice con Bcast su grigliar  
//calcolo dimensioni locali [att non divisibilità]  
// invio blocchi matrice
```

I Strategia:

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

$$N_{\text{loc}} = N/p$$

$$\dim[A_i] = (N/p) \times M$$



p processori

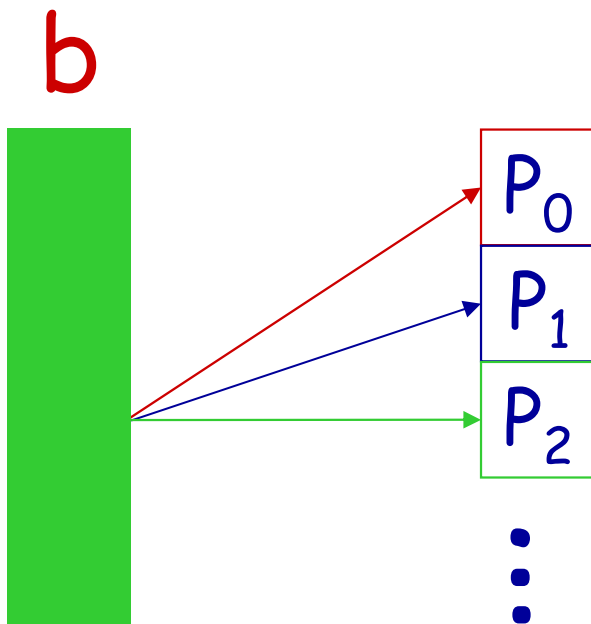

```
...  
// lettura dimensioni, controlli, allocazione  
// lettura matrice e vettore  
  
/*CREAZIONE DELLA GRIGLIA DI PROCESSORI*/  
crea_griglia(&grigliar,menum,nproc,righe,colonne,coord);  
  
/*DISTRIBUZIONE DATI*/  
//invio dimensioni matrice con Bcast su grigliar  
//calcolo dimensioni locali [att non divisibilità]  
// invio blocchi matrice  
// Bcast del vettore su grigliar
```

I Strategia:

MIMD-DM

$$A \in \mathbb{R}^{N \times M}, \quad b \in \mathbb{R}^M, c \in \mathbb{R}^N$$

$$\dim[b] = M$$



p processori

Possibilità varie

- Attenzione all'allocazione della matrice locale in ogni processore **Aloc**: in P0 ci deve stare anche **A**
- Vedremo insieme nelle prossime lezioni come trattare la non divisibilità delle righe di **A** per il numero dei processori
- Ottimizzazione accessi in memoria:
 - La matrice può essere LINEARIZZATA -

...
// lettura dimensioni, controlli, allocazione
// lettura matrice e vettore

/*CREAZIONE DELLA GRIGLIA DI PROCESSORI*/
crea_griglia(&**grigliar**,menum,nproc,righe,colonne);

/*DISTRIBUZIONE DATI*/
//invio dimensioni matrice con Bcast su grigliar
//calcolo dimensioni locali **[att non divisibilità]**
//invio blocchi matrice

/*CALCOLO LOCALE*/

//riadattare il codice sequenziale alle nuove dimensioni
matXvet_local(&grigliar,menum,nproc,righe_loc,colonne,
A_loc, b);

In particolare ...

Algoritmo sequenziale

```
for  $i=0, n-1$  do
```

```
     $y_i = 0$ 
```

```
    for  $j=0, n-1$  do
```

```
         $y_i = y_i + a_{ij} x_j$ 
```

```
    endfor
```

```
endfor
```

N_loc, M, a_loc, b

L' i -esimo elemento di y
è il prodotto scalare della
 i -esima riga di A per il vettore x


```
...  
// lettura dimensioni, controlli, allocazione  
// lettura matrice e vettore  
  
/*CREAZIONE DELLA GRIGLIA DI PROCESSORI*/  
crea_griglia(&grigliar,menum,nproc,righe,colonne);  
  
/*DISTRIBUZIONE DATI*/  
//invio dimensioni matrice con Bcast su grigliar  
//calcolo dimensioni locali [att non divisibilità]  
//invio blocchi matrice  
  
/*CALCOLO LOCALE*/  
//riadattare il codice sequenziale alle nuove dimensioni  
matXvet_local(&grigliar,menum,nproc,righe_loc,colonne,  
A_loc, b);  
...  
...
```

...

// nessuna collezione dei risultati (se non esplicitamente richiesta)

I STRATEGIA: Esempio $n=6$, $p=2$

Il processore 0 può calcolare
SOLO le prime tre componenti
del vettore y

P_0

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}$$

Il processore 1 può calcolare
SOLO le altre tre componenti
del vettore y

$$\begin{bmatrix} a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{50} & a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} y_3 \\ y_4 \\ y_5 \end{bmatrix}$$

...

```
/* nessuna collezione dei risultati (se non esplicitamente richiesta)*/  
// organizzazione della stampa  
    //sequenziale ciclo for sui processori di grigliar
```

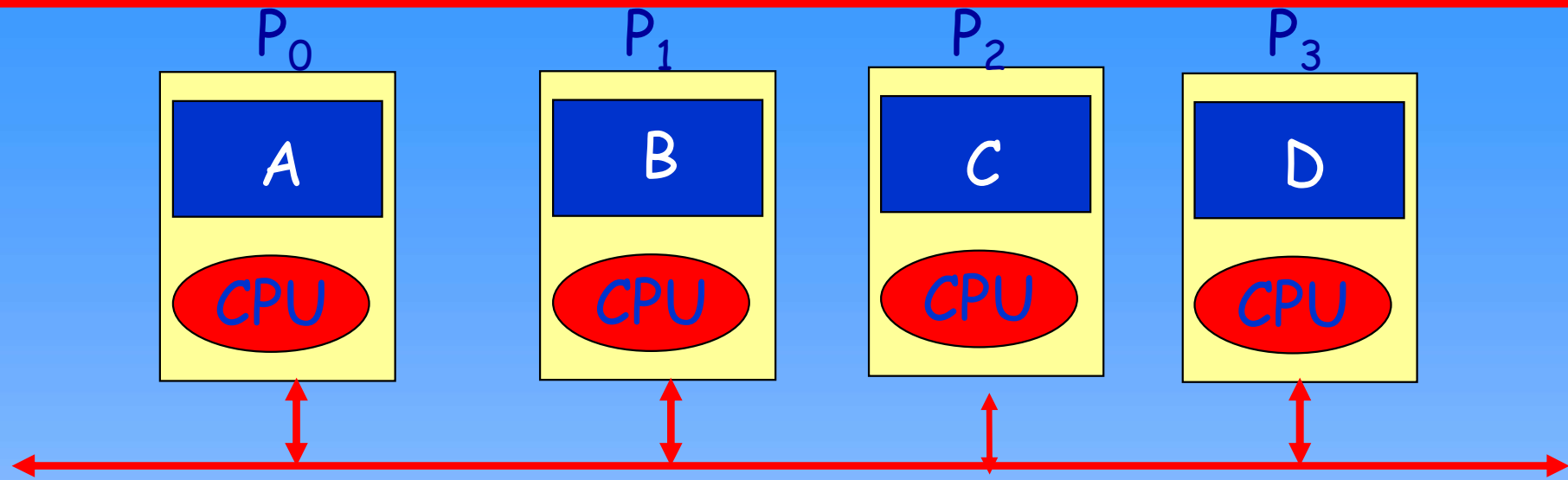
...

/* collezione in P0 dei risultati

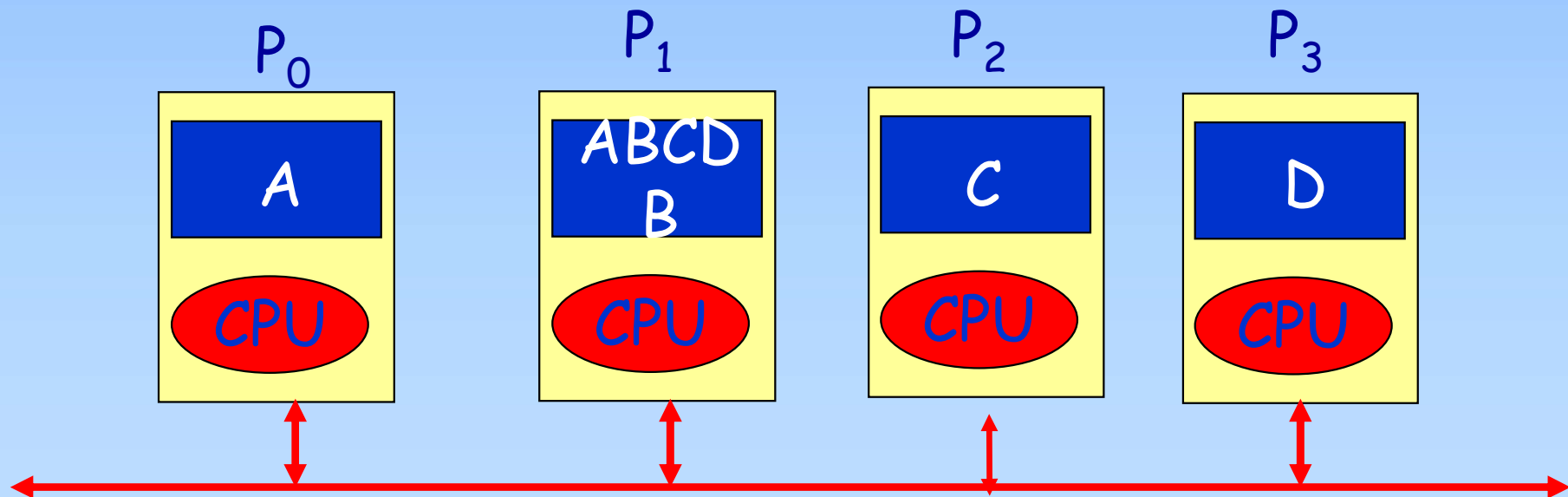
// organizzazione della stampa

//Gathering sui processori

Spedizione collettiva di tipo: *data gather*



Tutti i processori spediscono i propri dati ad un processore assegnato (es al processore P_1)



Il gather in MPI: in dettaglio...

```
MPI_Gather(void *send_buff, int send_count,  
           MPI_Datatype sendtype,  
           void *recv_buff, int recv_count,  
           MPI_Datatype recv_type,  
           int root, MPI_Comm comm);
```

***send_buff** indirizzo del dato da spedire

send_count numero dei dati da spedire

sendtype tipo dei dati da spedire

***recv_buff** indirizzo del dato in cui **root** riceve

recv_count numero dei dati che root riceve

recv_type tipo dei dati che root riceve


root identificativo del processore che riceve da tutti

comm identificativo del communicator

Il gather in MPI

```
MPI_Gather(void *send_buff, int send_count,  
          MPI_Datatype datatype,  
          void *recv_buff, int recv_count,  
          MPI_Datatype recv_type,  
          int root, MPI_Comm comm);
```

- Gli argomenti **recv_** sono significativi solo per il processore **root**
- L'argomento **recv_count** è il numero di dati da ricevere da ogni processore, non è il numero totale dei dati da ricevere.



I progetti possono essere svolti in gruppo per un massimo di 3 studenti.

- Scelta e comunicazione del progetto
- Approvazione del docente (che sono io!)

email: livia.marcellino@uniparthenope.it
no teams

Consegna progetto prima di sostenere l'esame orale:

La consegna del progetto dovrà avvenire entro la settimana **prima** del data indicata dal docente per l'orale

- Entro metà maggio fisso tutte le date per tutti i mesi dell'anno solare 2022
- Sarà necessario inviare il codice sviluppato e una **breve relazione** a corredo
- **Materiale Aggiuntivo:** **documentazione software parallelo**

Progetti

Alla lista dei 6 già assegnati:

- 7. Implementazione dell'algoritmo parallelo (np processori) per il calcolo del prodotto tra una matrice A di dimensione $N \times M$ ed un vettore b di dimensione M (I strategia), in ambiente MPI-Docker.