

# Calcolo Parallelo e Distribuito

---

Esempio uso del costrutto work-sharing: sections  
prova esame maggio

**Docente:** Prof. L. Marcellino

**Tutor:** Prof. P. De Luca

# 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.

# Il parallelismo delle architetture MIMD-SM

## metodi numerici paralleli

Modello Numerico  
 $M_h(P)$

Ripartire dal modello numerico (modello matematico discretizzato  $h=1,...,N$ ) e analizzare gli  $N$  passi in modo da distribuirli, eventualmente, a più unità processanti.

Più possibilità:

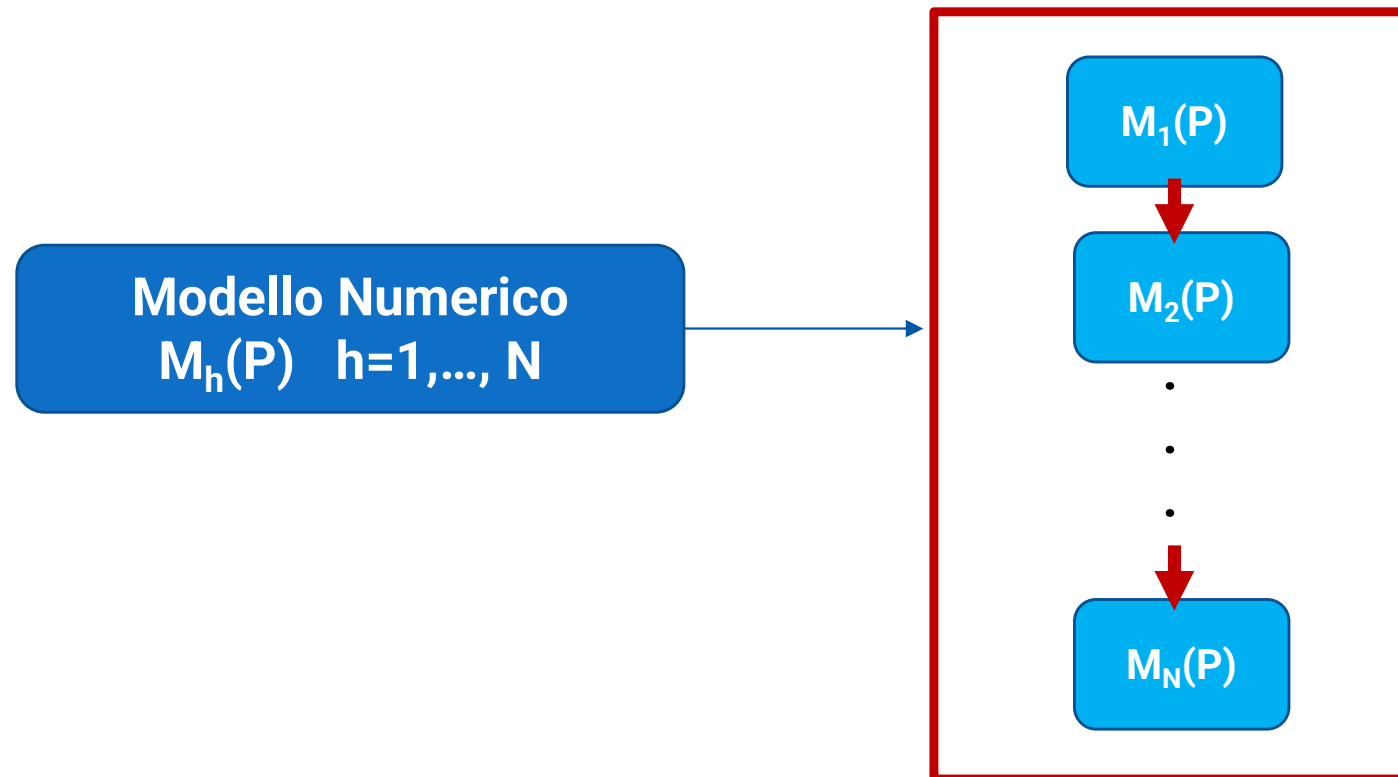
- ogni unità processante esegue un passo **differente**  
(**decomposizione funzionale**)
- tutte le unità processanti eseguono **la stessa** operazione su un sottoinsieme di dati  
(**decomposizione del dominio**)
- **combinazione delle due possibilità precedenti**

# Il parallelismo delle architetture MIMD-SM

## metodi numerici paralleli

Modello Numerico  
 $M_h(P)$

Ripartire dal modello numerico (modello matematico discretizzato  $h=1,\dots,N$ ) e analizzarlo per individuare **task indipendenti** che possano essere **processati separatamente e contemporaneamente**.

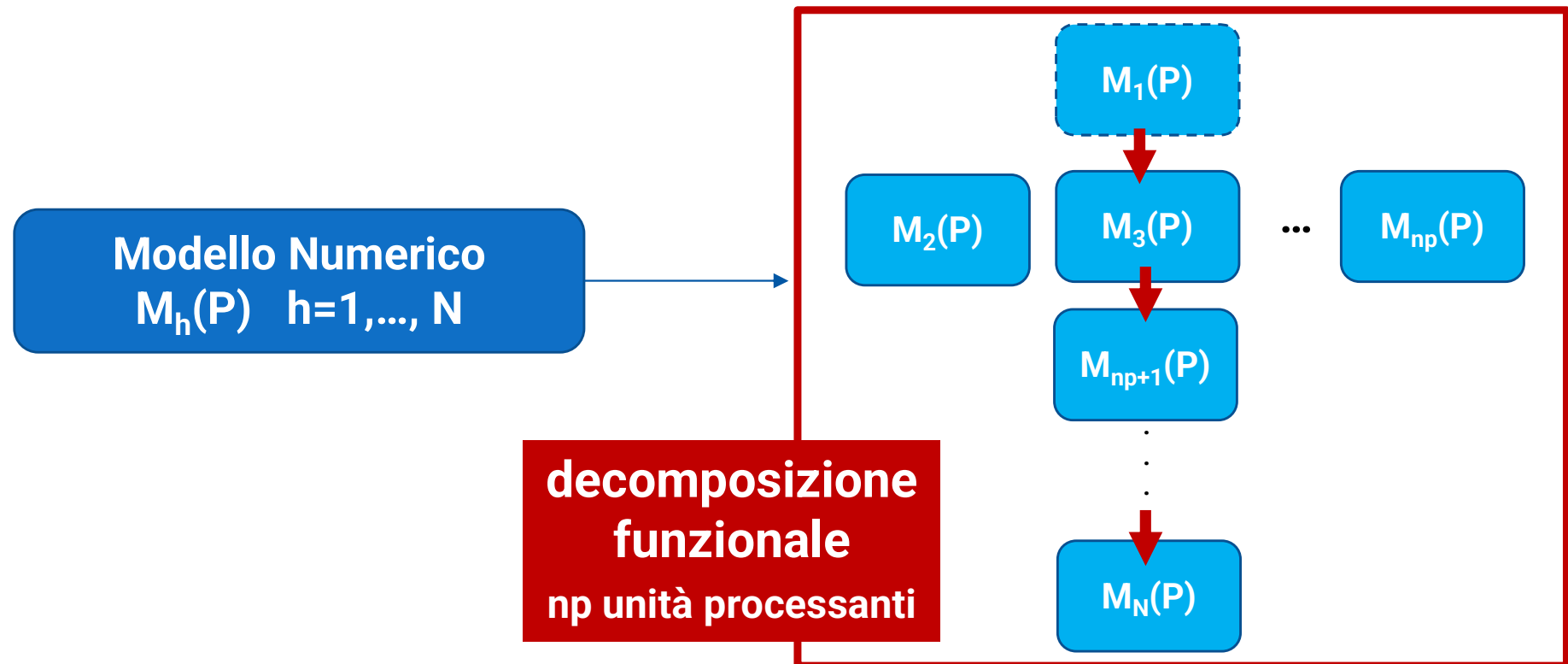


# Il parallelismo delle architetture MIMD-SM

## metodi numerici paralleli

Modello Numerico  
 $M_h(P)$

Ripartire dal modello numerico (modello matematico discretizzato  $h=1,...,N$ ) e analizzarlo per individuare **task indipendenti che possano essere processati separatamente e contemporaneamente**.





# Il parallelismo delle architetture MIMD-SM

## metodi numerici paralleli – decomposizione funzionale

Ripartire dal modello numerico (modello matematico discretizzato  $h=1,...,N$ ) e analizzarlo per individuare **task indipendenti che possano essere processati separatamente e contemporaneamente**.

Decomposizione funzionale = **eseguire elaborazioni differenti e indipendenti contemporaneamente**

Il modello è decomposto in base al lavoro che deve essere svolto.

**Caratteristica fondamentale:**

applicabile a un modello caratterizzato da più nuclei computazionali

# 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 parallel 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

# Come assegnare lavori differenti a differenti thread

```
#pragma omp parallel sections num_threads(3)
{
    #pragma omp section
    {
        printf( Hello world ONE! )
    }
    #pragma omp section
    {
        printf( Hello world TWO! )
    }
    #pragma omp section
    {
        printf( Hello world THREE! )
    }
}
```

# Attenzione

Tre thread che fanno contemporaneamente la stessa cosa.

```
#pragma omp parallel num_threads(3)
{
    printf( Hello world ! )
}
```

```
#pragma omp parallel sections num_threads(3)
{
    #pragma omp section
    {
        printf( Hello world ONE! )
    }
    #pragma omp section
    {
        printf( Hello world TWO! )
    }
    #pragma omp section
    {
        printf( Hello world THREE! )
    }
}
```



Implementatelo aggiungendo la stampa da parte di ogni thread del proprio identificativo

```
#pragma omp parallel sections num_threads(3)
{
    #pragma omp section
    {
        printf( Hello world ONE! )
    }
    #pragma omp section
    {
        printf( Hello world TWO! )
    }
    #pragma omp section
    {
        printf( Hello world THREE! )
    }
}
```

`omp_get_thread_num( )`

Che succede se cambio il numero di thread in questo codice?

```
#pragma omp parallel sections num_threads(4)
{
    #pragma omp section
    {
        printf( Hello world ONE! )
    }
    #pragma omp section
    {
        printf( Hello world TWO! )
    }

    #pragma omp section
    {
        printf( Hello world THREE! )
    }
}
```

Un quarto thread è allocato, ma non fa niente!!!  
CHI? Facciamo un po' di prove...

E se ne metto meno?

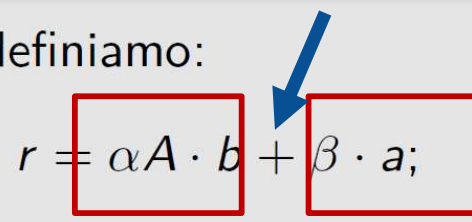
```
#pragma omp parallel sections num_threads(2)
{
    #pragma omp section
    {
        printf( Hello world ONE! )
    }
    #pragma omp section
    {
        printf( Hello world TWO! )
    }

    #pragma omp section
    {
        printf( Hello world THREE! )
    }
}
```

Un thread fa il doppio lavoro

Problema:

Dati  $A \in \mathbb{R}^{m \times n}$ ,  $b \in \mathbb{R}^n$ ,  $a \in \mathbb{R}^m$  definiamo:

$$r = \alpha A \cdot b + \beta \cdot a;$$


dove  $\alpha, \beta \in \mathbb{N}$ .

L'output è definito:

$$\prod_{i=0}^{m-1} r_i$$

- Implementare un programma in linguaggio C/C++ per l'ambiente parallelo MIMD-SM con l'ausilio della libreria OpenMP per risolvere il problema proposto.


Le due operazioni sono completamente indipendenti!



# Prova maggio

**Implementare un programma parallelo per l'ambiente multicore con  $np$  unità processanti che impieghi la libreria OpenMP. Il programma deve essere organizzato come segue:**

- **il core master deve generare una matrice  $B$  di dimensione  $N \times N$  e due vettori  $a$ ,  $b$  di lunghezza  $N$**
- **i core devono collaborare per costruire, in parallelo, una nuova matrice  $A$  ottenuta sommando alla diagonale principale della matrice  $B$  il vettore  $b$**
- **quindi, i core devono collaborare per calcolare in parallelo il prodotto tra la nuova matrice  $A$  ed il vettore  $a$ , distribuendo il lavoro per colonne**
- **infine, il core master stampa il risultato finale ed il tempo d'esecuzione**



Le due operazioni da fare sono sequenziali, quindi no SECTIONS:  
Non si può fare parallelismo funzionale

Attenzione alle metriche di valutazione!

- Il software deve compilare!
- È sempre necessario che il vostro software abbia la possibilità di leggere come input la dimensione del problema e il numero di core con cui deve lavorare.
- Il run del software deve essere completo (deve terminare correttamente, rispondendo correttamente al quesito)
- Lo sviluppo dei due quesiti deve essere svolto correttamente in parallelo!
- La stampa dei tempi deve correttamente variare al variare della dimensione e del numero di core utilizzati.



## Il primo quesito

### Algoritmo full parallel

```
#pragma omp parallel for shared(B, b) private(i)  
  for (i = 0; i < N; i++) {  
    B[i][i] += b[i];  
  }
```

## Il secondo quesito

Algoritmo con riduzione (distribuzione per colonne)

**v1: la più semplice ma la meno performante**

```
#pragma omp parallel for shared(B, a, result) private(i, j)
for (j = 0; j < N; j++) {
    for (i = 0; i < N; i++) {
        #pragma omp atomic
        result[i] += B[i][j] * a[j];
    }
}
```



## Il secondo quesito

Algoritmo con riduzione (distribuzione per colonne)

**v2: la prima strategia per la collezione dei vettori**

```
#pragma omp parallel for shared(B, a, result) private(i, j)
  for (j = 0; j < N; j++) {
    double temp[N] = {0}; // risultato parziale per ogni colonna
    for (i = 0; i < N; i++) {
      temp[i] = B[i][j] * a[j];
    }
    #pragma omp critical
    {
      for (i = 0; i < N; i++) {
        result[i] += temp[i];
      }
    }
  }
}
```

## Il secondo quesito

Algoritmo con riduzione (distribuzione per colonne)

**v3: la seconda strategia per la collezione dei vettori**

```
#pragma omp parallel for shared(B, a) private(i, j) reduction(+:result[:N])  
  for (j = 0; j < N; j++) {  
    for (i = 0; i < N; i++) {  
      result[i] += B[i][j] * a[j];  
    }  
  }
```

result[:N] indica che la riduzione deve essere effettuata su un array result di lunghezza N

riduzione su array

**disponibile a partire da OpenMP 4.5**