



Calcolo Parallelo e Distribuito

Laboratorio - libreria OpenMP:

algoritmo somma 1 strategia

Tempo di esecuzione del codice

Docente: Prof. L. Marcellino

Tutor: Prof. P. De Luca

Esempio: somma N numeri - algoritmo

Algoritmo
sequenziale
(1 thread)

n elementi da sommare a_i

```
begin
    ...
    sumtot :=  $a_0$ 
    for i = 1 to n-1 do
        sumtot := sumtot +  $a_i$ 
    endfor
    ...
end
```

Esempio: somma N numeri - algoritmo ($n=kp$)

$p = \# \text{ thread}$

Porzione
di codice
eseguita
da ogni
thread T_i

begin

sumtot := 0

forall T_i , $0 \leq i \leq p-1$ do

$s_i := 0$

$h_i := i * (n/p)$

for $j = h_i$ to $h_i + (n/p) - 1$ do

$s_i := s_i + a_j$

endfor

lock(sumtot)

sumtot := sumtot + s_i

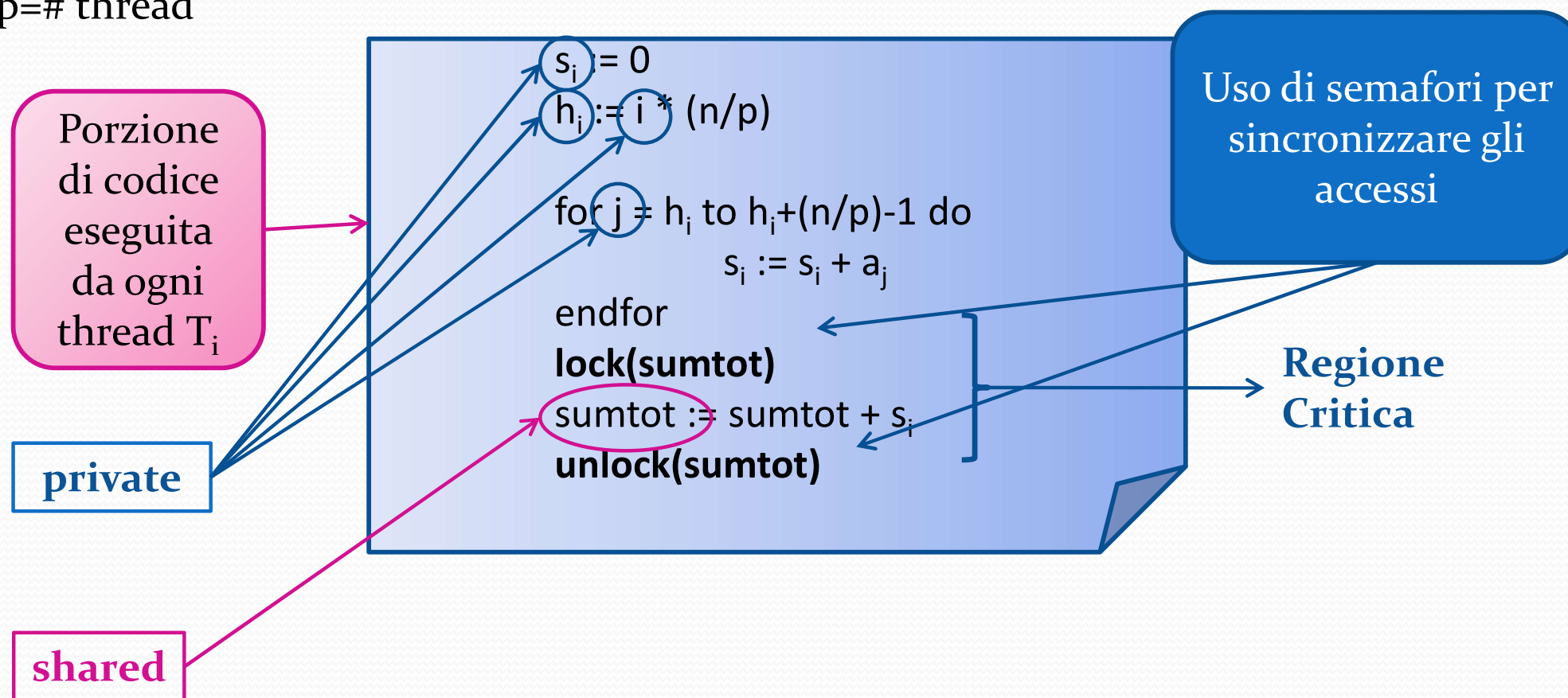
unlock(sumtot)

end forall

end

Esempio: somma N numeri - algoritmo ($n=kp$)

$p = \# \text{ thread}$



OpenMp

Esempio: somma N numeri - algoritmo ($n=kp$)

Algoritmo parallelo

```
si := 0
hi := i * (n/p)
for j = hi to hi+(n/p)-1 do
    si := si + aj
endfor
lock(sumtot)
sumtot := sumtot + si
unlock(sumtot)
```

Utilizzando **OpenMP**:

- l'utilizzo di variabili private d'appoggio
 - la divisione del lavoro tra i thread
 - la collezione del risultato in una variabile shared in modo sincronizzato
- possono essere gestiti molto semplicemente!**

Algoritmo sequenziale

```
...
sumtot = 0;
#pragma omp ...(a,sumtot)
for(i=0;i<n;i++)
{
    sumtot=sumtot+a[i];
}
...
```


Introduzione ad OpenMp

Facile
trasformare
un codice
sequenziale
in parallelo

Open specifications for Multi Processing

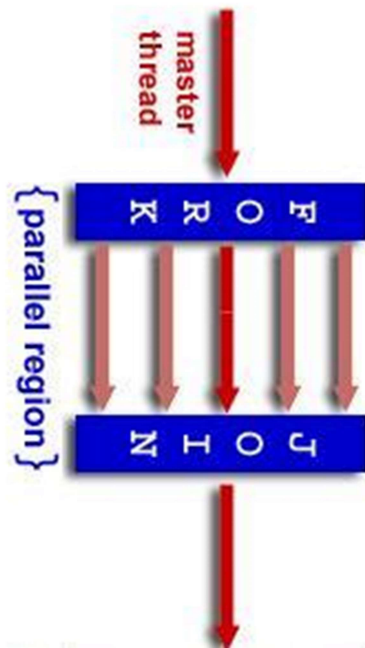
- Prevede Application Program Interface (API) per gestire il parallelismo shared-memory multi-threaded
- Consente un approccio ad alto livello, user-friendly
- Portabile: Fortran e C/C++, Unix/Linux e Windows

Algoritmo parallelo con OpenMP

```
...  
sumtot = 0;  
#pragma omp ...(a,sumtot)  
for(i=0;i<n;i++)  
{  
    sumtot=sumtot+a[i];  
}  
...
```

Introduzione ad OpenMp

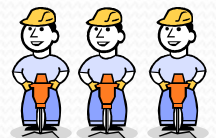
- Il modello d'esecuzione parallela è quello ***fork-join***



```
...  
sumtot = 0
```

```
#pragma omp ...(a,sumtot)  
for(i=0;i<n;i++)  
{  
    sumtot=sumtot+a[i];  
}
```

```
printf("\nLa somma totale è %d: ", sumtot);  
...
```



Esempio: Somma N numeri - codice

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

int main()
{
    int i,N,t,nloc;
    float sumtot,sum,*a;
    sumtot=0;

    printf("Inserire N\n");
    scanf("%d",&N);

    a=(float *)calloc(N,sizeof(float));

    printf("Inserire i numeri da sommare\n");
    for (i=0;i<N;i++)
    {
        scanf("%f",&a[i]);
    }
```

...

Esempio: Somma N numeri - codice

implementa la 1 strategia

...

```
#pragma omp parallel private(sum,nloc,i) shared(sumtot)
```

```
{ // se piu di un'istruzione
```

```
    t=omp_get_num_threads();
```

```
    nloc=N/t;
```

```
    printf("sono %d, di %d: numeri %d\n",omp_get_thread_num(),t,nloc);
```

```
    sum=0;
```

```
    for(i=0;i<nloc;i++)
```

```
    {
```

```
        sum=sum+a[i+nloc*omp_get_thread_num()];
```

```
    }
```

```
    sumtot+=sum;
```

```
} //fine direttiva
```

```
    printf("somma totale: %f\n",sumtot);
```

```
}
```

Esempio: Somma

implementa la 1 strategia

```
...  
  
#pragma omp parallel private(sum,nloc,i) sh  
  
{ // se piu di un'istruzione  
  
    t=omp_get_num_threads();  
    nloc=N/t;  
    printf("sono %d, di %d: numeri %d\n",omp_get_thread_num(),nloc,i);  
  
    sum=0;  
    for(i=0;i<nloc;i++)  
    {  
        sum=sum+a[i+omp_get_thread_num()];  
    }  
    sumtot+=sum;  
  
} //fine direttiva  
  
printf("somma totale: %f\n",sumtot);  
  
}
```

Se non diversamente specificato, nel momento in cui all'interno di una direttiva di tipo parallel si fa accesso ad un variabile condivisa, utilizzando valori conservati in variabili private, l'accesso avviene in maniera sequenziale da parte dei threads: proprio per conservare la consistenza dei dati!

Libreria parallela alto livello!

Compile ed eseguite dando in input un N multiplo del numero di t...

Accorgimenti

Che succede, invece se in input viene dato N non divisibile per t???

E' necessario effettuare una modifica al programma:

- Nella direttiva, subito dopo il calcolo di N/t , è necessario verificare quanto viene il resto della divisione:

$r=N\%t$

- Se il resto non è zero, tutti i processori con id strettamente minore del resto devono occuparsi di sommare un elemento in più:

$id=omp_get_thread_num();$

$if (id < r)$

{

$nloc++; step=0;$

}

else

$step=r;$

- La variabile step consente ad ogni core di sapere di quali elementi si deve occupare

Accorgimenti

Che succede, invece se in input viene dato N non divisibile per t???

- La variabile step consente ad ogni core di sapere di quali elementi si deve occupare:

```
sum=0;
for(i=0;i<nloc;i++)
{
    sum=sum+a[i+nloc*omp_get_thread_num()+step];
}
```

- Tutto il resto è uguale: **VEDIAMO COME VIENE MODIFICANDOLO!**

Modifica... **sommaSM_v2**

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main()
{

    int i,N,t,nloc, r,id,step;
    float sumtot,sum,*a;
    sumtot=0;

    printf("Inserire N\n");
    scanf("%d",&N);

    a=(float *)calloc(N,sizeof(float));
    printf("Inserire i numeri da sommare\n");
    for (i=0;i<N;i++)
    {
        scanf("%f",&a[i]);
    }
    ...
```

Modifica... **sommaSM_v2**

...

```
#pragma omp parallel private(sum,nloc,i,id,step) shared(sumtot,r)  
{ // inizio direttiva, con le nuove variabili
```

```
t=omp_get_num_threads(); nloc=N/t;
```

```
r=N%t;
```

```
id=omp_get_thread_num();
```

```
// stampa di prova per vedere se tutto procede bene
```

```
printf("sono %d, di %d: numeri %d, r=%d\n",id,t,nloc,r);
```

...

Modifica... **sommaSM_v2**

```
...  
  
// suddivisione del lavoro tra i thread  
if (id < r)  
{  
    nloc++;  
    step=0;  
}  
  
else  
    step=r;  
  
// Un'altra stampa di prova per sicurezza  
printf("sono %d, di %d: numeri %d, r=%d, passo=%d\n",id,t,nloc,r,step);  
  
// piccola modifica dell'operazione  
sum=0;  
for(i=0;i<nloc;i++)  
{  
    sum=sum+a[i+nloc*omp_get_thread_num()+step];  
}  
  
...
```

Modifica...

sommaSM_v2

```
...  
  
// ultima stampa di prova  
  
    printf("sono %d, di %d: numeri %d, r=%d, la mia sum=%f\n",id,t,nloc,r,sum);  
  
    sumtot+=sum;  
} //fine direttiva  
  
printf("somma totale: %f\n",sumtot);  
  
}
```




OpenMp


prendere i tempi
&
fare i grafici

Tempi di un algoritmo parallelo

- Per prendere i tempi in maniera accurata si utilizza una funzione apposita di openMP all'interno del codice.

Calcolo del tempo
trascorso

```
...  
#include <time.h>  
...  
  
double * sumtot(int N){  
    double result;  
    ...  
    double t0,t1;  
    double tempototale;  
    ...  
    t0=omp_get_wtime();  
  
    #pragma omp parallel ...  
    ...  
    // fine del pragma  
    t1=omp_get_wtime();  
    tempototale= t1 - t0;  
    return result;  
}
```

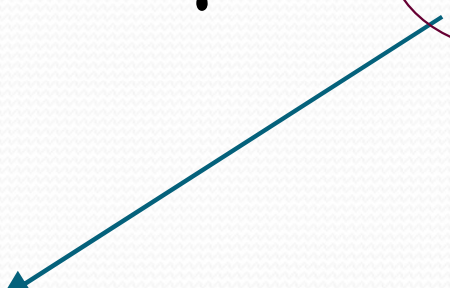


A che mi serve prendere i tempi
dell'algoritmo parallelo
da me implementato
(ovvero del codice)

?

Da dove siamo partiti...

una rappresentazione semplificata del
tempo richiesto per l'esecuzione
di un software è:

$$\tau = \mu \cdot k \cdot T(N)$$


$$\mu = t_{\text{calc}}$$

$T(N)$ = complessità computazionale dell'algoritmo

Dipendenza dall'algoritmo

In parallelo...

$$\tau_p = \mu k \cdot T_p(N)$$

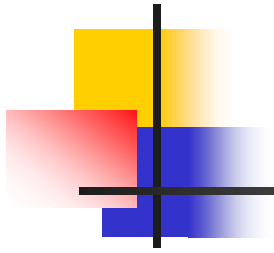
$T_p(N)$ = complessità computazionale
dell'algoritmo parallelo

Dipendenza dall'algoritmo
parallelo

Analisi della strategia di parallelizzazione relativa all'algoritmo parallelo



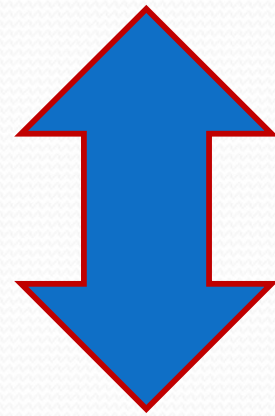
Vale la pena implementare in un software
la strategia di parallelizzazione pensata
per l'algoritmo?



Analisi dell'algoritmo parallelo

$$T_p(N)$$

Analisi delle performance del software



Conferma di quanto ho previsto
teoricamente
in uno specifico ambiente parallelo



Prestazioni di un software parallelo:

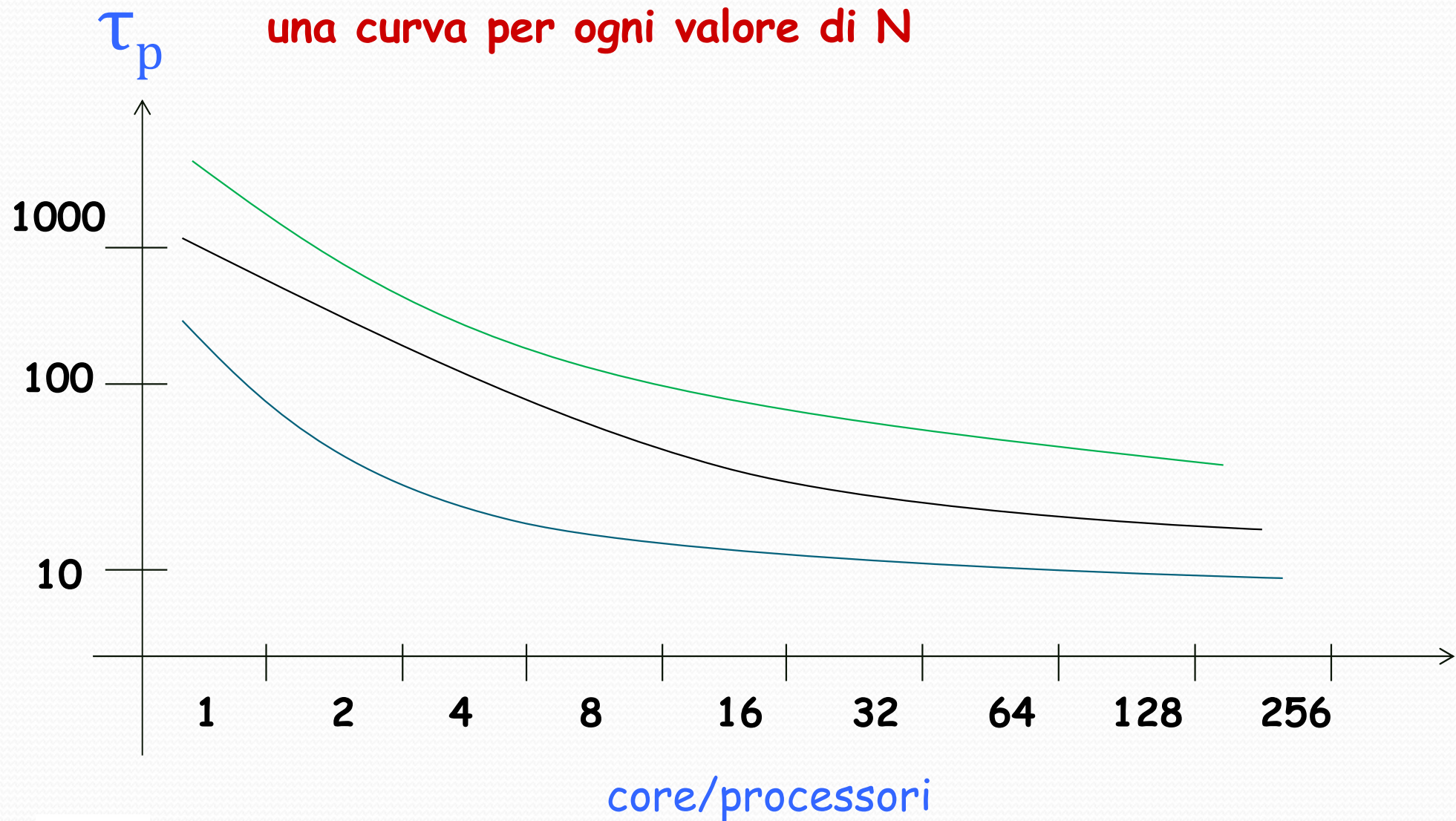
$$\tau_p = \boxed{\mu k} \cdot T_p(N)$$

Misurare i tempi
d'esecuzione effettivi

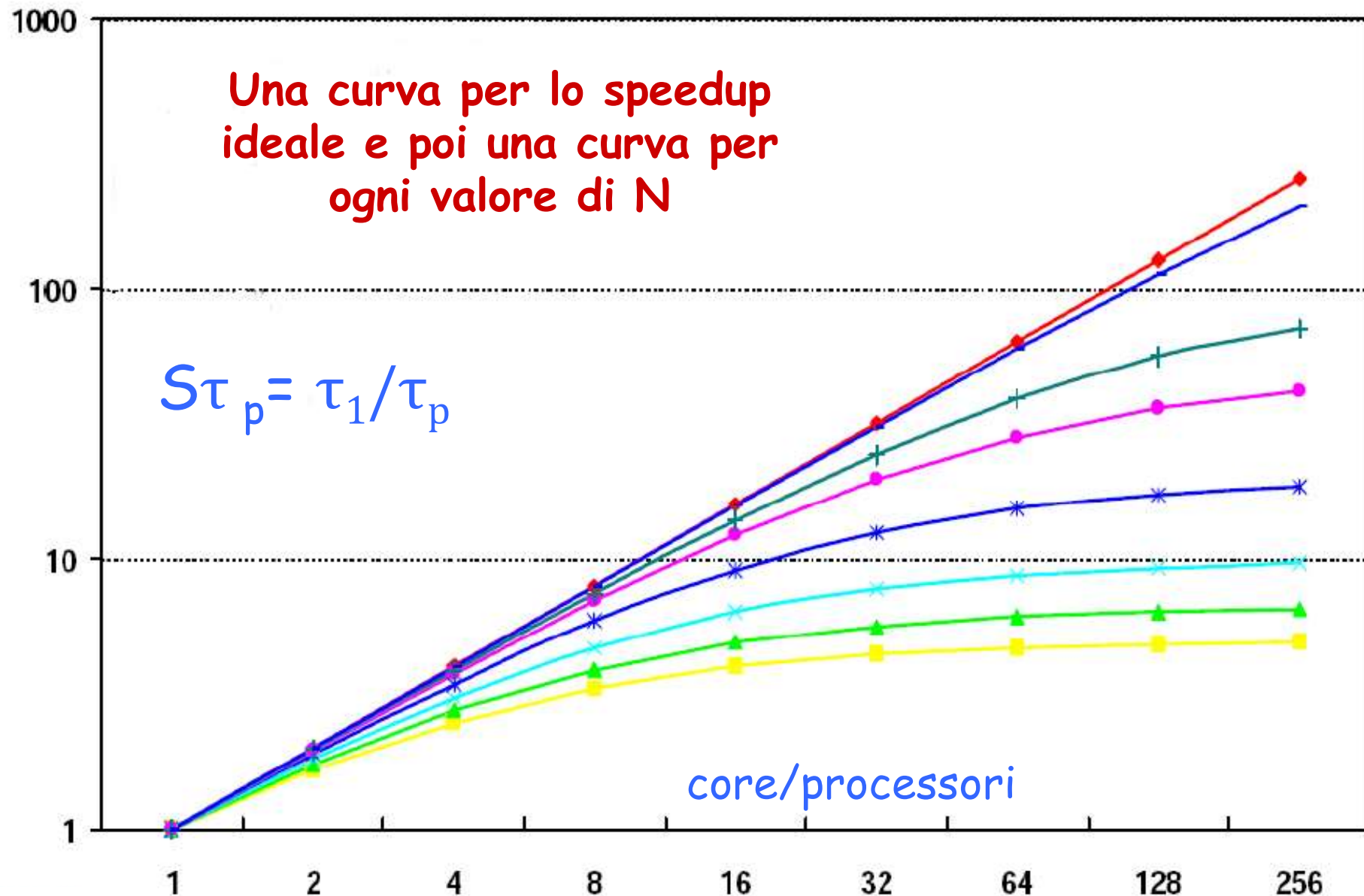


realizzare grafici

I grafici dei tempi del software



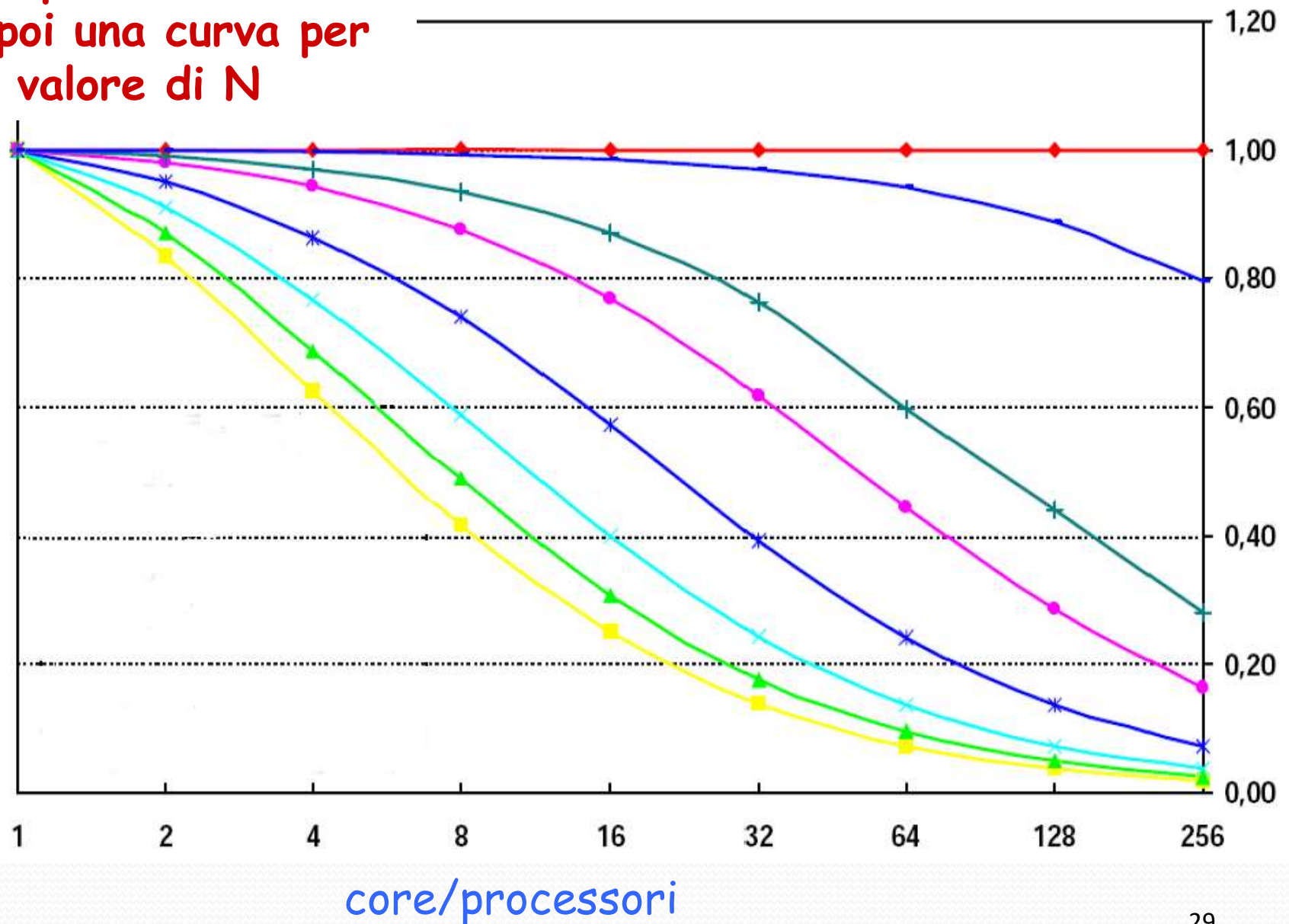
I grafici di speedup del software



I grafici di efficienza del software

Una curva per l'efficienza ideale e poi una curva per ogni valore di N

$$E_{\tau_p} = S\tau_p/p$$



Somma N numeri II strategia

- Per implementare la II strategia della somma di N numeri, si può utilizzare la clausola:

`reduction(operator: list)`

**vediamo insieme
la prossima lezione di laboratorio**