



Laboratorio - libreria OpenMP: algoritmo somma tra vettori

Docente: Prof. L. Marcellino

Tutor: Prof. P. De Luca

## Esempio: Somma tra vettori - algoritmo

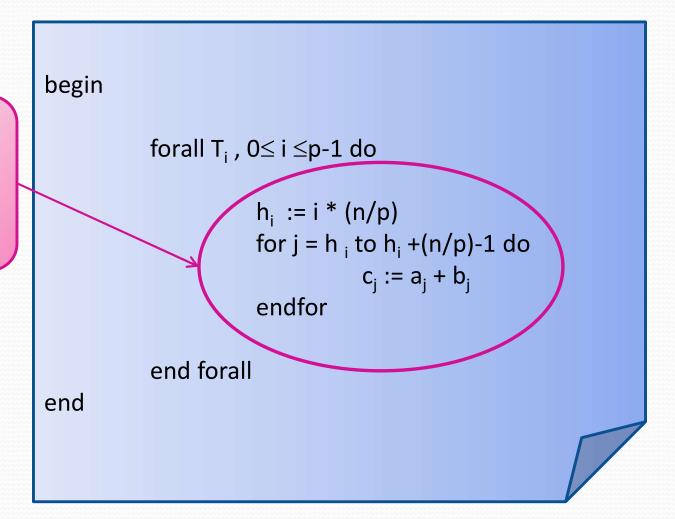
Algoritmo sequenziale (1 thread)

due vettori a, b di n elementi da sommare

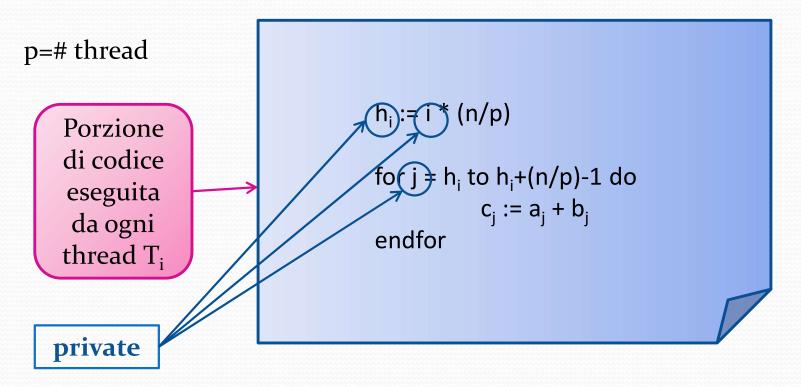
## **Esempio:** Somma tra vettori – algoritmo N=k\*p

p=# thread

Porzione di codice eseguita da ogni thread T<sub>i</sub>



## **Esempio:** Somma tra vettori – algoritmo N=k\*p



## OpenMP

### Esempio: somma tra vettori – algoritmo N=k\*p

### Algoritmo parallelo

```
h_i := i * (n/p)
for j = h_i to h_i+(n/p)-1 do
c_j := a_j + b_j
endfor
```

### Utilizzando **OpenMP**:

- •l'utilizzo di variabili private d'appoggio
- •la divisione del lavoro tra i thread

possono essere gestiti molto semplicemente!

```
Algoritmo parallelo con OpenMP

...

#pragma omp ...(a, b, c)

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

{
            c[i] = a[i] + b[i];
}
...
```

## OpenMP

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

```
#pragma omp ... (a, b, c)
for(i=0;i<n;i++)
    c[i] = a[i] + b[i];
...
```

Facile trasformare un codice sequenziale in parallelo

## Esempio: somma tra vettori – codice

vediamo nel dettaglio il codice

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main()
       int i,N,t,nloc, indice;
       float *a, *b, *c;
       printf("Inserire N\n");
       scanf("%d",&N);
       // allocazione
       a=(float *)calloc(N,sizeof(float));
       b=(float *)calloc(N,sizeof(float));
       c=(float *)calloc(N,sizeof(float));
```

## Esempio: somma tra vettori – codice

vediamo nel dettaglio il codice

```
// lettura
printf("Inserire gli elementi del vettore a\n");
for (i=0;i<N;i++)
  scanf("%f",&a[i]);
printf("Inserire gli elementi del vettore b\n");
for (i=0;i<N;i++)
  scanf("%f",&b[i]);
```

## Esempio: somma tra vettori – codice

vediamo nel dettaglio il codice

```
#pragma omp parallel private(nloc,i, indice) shared(a, b, c)
 { // 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);
      for(i=0;i<nloc;i++)
                 indice = i+nloc*omp get thread num();
                 c[indice] = a[indice] + b[indice];
 } //fine direttiva
                                    Compilate ed eseguite dando in input
// stampa finale
                                    un N multiplo del numero di t...
 for (i=0;i<N;i++)
   printf("%f",c[i]);
```

## 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 calcolare il resto di questa divisione:

```
r=N%t
```

 Se il resto non è zero, tutti i core con identificativo strettamente minore del resto devono occuparsi di calcolare un elemento in più del vettore soluzione c:

```
id=omp_get_thread_num();
if (id < r)
{
      nloc++; step=0;
}
else
      step=r;</pre>
```

 La variabile step consente ad ogni core di sapere di quali elementi, dei vettori a e b, 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, dei vettori a e b, si deve occupare:

```
for(i=o;i<nloc;i++)
{
  indice = i+nloc*omp_get_thread_num() +step;
  c[indice] = a[indice] + b[indice];
}</pre>
```

**VEDIAMO COME VIENE MODIFICANDOLO!** 

Provate da soli...

### aggiungere variali nella fase di dichiarazione

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main()
       int i,N,t,nloc, indice, r, id, step;
       float *a, *b, *c;
       printf("Inserire N\n");
       scanf("%d",&N);
       // allocazione
       a=(float *)calloc(N,sizeof(float));
       b=(float *)calloc(N,sizeof(float));
       c=(float *)calloc(N,sizeof(float));
```

### la lettura resta uguale

```
// lettura
printf("Inserire gli elementi del vettore a\n");
for (i=0;i<N;i++)
  scanf("%f",&a[i]);
printf("Inserire gli elementi del vettore b\n");
for (i=0;i<N;i++)
  scanf("%f",&b[i]);
```

#### modifica 1 nella direttiva

```
#pragma omp parallel private(nloc,i,id,step) shared(a, b, c, 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 2 nella direttiva

```
// 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);
```

#### modifica 3 nella direttiva

```
// piccola modifica dell'operazione
 for(i=0;i<nloc;i++)</pre>
   indice = i+nloc*omp_get_thread_num() +step;
   c[indice] = a[indice] + b[indice];
} // fine della direttiva
```

la stampa, chiaramente, resta la stessa

```
// stampa finale
 for (i=0;i<N;i++)
   printf("%f\n",c[i]);
} // fine codice
```