Cenni sulla costruzione

Algoritmo per la Somma di n numeri

Problema

Scrivere un algoritmo parallelo per il calcolo della somma di n numeri

Scrivere un algoritmo parallelo per il calcolo della somma di n numeri

- Descriviamo 4 fasi fondamentali separatamente, costruendo una bozza per il codice implementativo:
  - Lettura e distribuzione dei dati
  - Calcolo del sottoproblema
  - Eventuali comunicazioni per il calcolo del risultato finale
  - Stampa del risultato

Dichiarazioni

```
/* SCOPO: Calcola la somma di un insieme di interi
    Consideriamo:
                    numero di processi >=1
                    numero di elementi >= numero di processi  */

#include <stdio.h>
#include “mpi.h”
main(int argc, char *argv[]){
    int menum, nproc,… ;
    int n, nloc, tag, i,…;
    int *x, *xloc;
    MPI_Status status;
…
```

Lettura e distribuzione dei dati (1)

(prima versione)

```
MPI_Init(&argv, &argc);
MPI_Comm_rank(MPI_COMM_WORLD, &menum);
MPI_Comm_size(MPI_COMM_WORLD, &nproc);
if (menum==0){
        “Lettura dei dati di input: n e x”
        for(i=1;i<nproc;i++){
                tag=10+i
                MPI_Send(&n, 1, MPI_INT, i, tag, MPI_COMM_WORLD);
        }
}else{
        tag=10+menum
        MPI_Recv(&n, 1, MPI_INT, 0, tag, MPI_COMM_WORLD,&status);
```

Lettura e distribuzione dei dati (1)

(seconda versione)

```
MPI_Init(&argv, &argc);
MPI_Comm_rank(MPI_COMM_WORLD, &menum);
MPI_Comm_size(MPI_COMM_WORLD, &nproc);
if (menum==0){
        “Lettura dei dati di input: n e x”
}
MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);
```

Lettura e distribuzione dei dati (1)

```
MPI_Init(&argv, &argc);
MPI_Comm_rank(MPI_COMM_WORLD, &menum);
MPI_Comm_size(MPI_COMM_WORLD, &nproc);
if (menum==0){
        “Lettura dei dati di input: n e x”
}
MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);
nloc=n/nproc
rest=n%nproc
if (menum<rest) nloc=nloc+1
“allocazione di xloc”
if (menum==0){
        xloc=x
```

Lettura e distribuzione dei dati (2)

```
        tmp=nloc
        start=0
        for (i=1;i<nproc;i++){
                start=start+tmp
                tag=22+i;
                if(i==rest) tmp=tmp-1
                MPI_Send(&x[start],tmp,MPI_INT,i,tag,MPI_COMM_WORLD);
        }
}/*endif*/
else{
        tag=22+menum
        MPI_Recv(xloc,nloc,MPI_INT,0,tag, MPI_COMM_WORLD,&status);
}
```

Lettura e distribuzione dei dati (NOTE)

- Attenzione all'allocazione di **x** ed **xloc**: in Po non sono necessarie entrambi!! Si può decidere di usare solo uno dei due vettori (il maggiore risparmio di memoria si ha usando solo **xloc**, ma bisogna spezzare in più parti le fasi di lettura-invio)
- La variabile **tmp** serve a gestire la situazione in cui i primi **rest-1** processori hanno un elemento in più degli altri
- La variabile **start,** ad ogni passo di invio di elementi, contiene il numero di elementi già spediti e quindi l'indice da cui ripartire per il prossimo invio.
- Non è l'unico modo di scrivere questo algoritmo: si possono scrivere i cicli in modo diverso senza necessariamente cambiare il numero di operazioni, e si può certamente scriverne una versione ottimizzata.
- Alla fine di questa fase si può procedere con il calcolo locale delle somme parziali.

Calcolo Locale

```
…
…
…
/*tutti i processori*/
sum=0
for(i=0;i<nloc;i++)
        sum=sum+xloc[i]
…
…
…
```

Calcolo della somma totale

(I strategia di comunicazione)

```
…
…
…
if (menum==0){
        for(i=1;i<nproc;i++){
                tag=80+i
                MPI_Recv(&sum_parz,1,MPI_int,i,tag,MPI_COMM_WORLD,&status);
                sum=sum+sum_parz
        }
}else{
        tag=menum+80
        MPI_Send(&sum,1,MPI_INT,0,tag, MPI_COMM_WORLD);
}
```

Calcolo della somma totale

(II strategia di comunicazione)

```
…
for(i=0;i<log2nproc;i++){ /*passi di comunicazione*/
        if ((menum%2^i)==0){   /*chi partecipa alla comunicazione*/
                if ((menum%2^(i+1))==0){ /*chi riceve*/
                        "Ricevi da menum+2^i"
                }else{
                        "Spedisci a menum-2^i"
                }
        }
}
…
```

Calcolo della somma totale

(III strategia di comunicazione)

```
…
for(i=0;i<log2nproc;i++){ /*passi di comunicazione*/
 /*tutti partecipano ad ogni passo*/
        if ((menum%2^(i+1))<2^i){   /*decidiamo solo a chi si invia e da chi si riceve*/
                "Ricevi da menum+2^i"
                "Spedisci a menum+2^i"
        }else{
                "Ricevi da menum-2^i"
                "Spedisci a menum-2^i"

        }
}
…
…
```

Stampa del risultato
(I versione)

```
…
…
…
/*se ci basta che la stampi solo un processore (P0)*/
if (menum==0)
        printf("\nSomma totale=%d\n", sum);
…
…
…
```

Stampa del risultato
(II versione)

```
…
…
…
/*se vogliamo che la stampino tutti i processori*/

printf("\nSono il processo %d: Somma totale=%d\n", menum,sum);
…
…
…
```