“Prendere” il tempo di esecuzione in ambiente a memoria distribuita

Prendere i tempi di esecuzione

- Rispondiamo con queste note a due domande:
  - Come facciamo a sapere che i processi stanno lavorando effettivamente con memoria distribuita?
  - Quali istruzioni si devono usare nel codice per avere il tempo giusto in output?

Eseguire in un ambiente a memoria distribuita

- In questa fase, vogliamo che i nostri processi vengano eseguiti su nodi diversi, perché vogliamo lavorare in un ambiente omogeneo a memoria distribuita.
- Ricordando come è fatto il nostro cluster (ogni nodo ha 8 processori, i quali in parte condividono la memoria), dobbiamo quindi assicurarci che tali processi vengano eseguiti su nodi diversi
- Vediamo un "trucco" per realizzare questa condizione, con una rapida **modifica allo script pbs** già visto a lezione
- Approfondiremo successivamente gli altri modi di fare la stessa cosa

Modifiche al job-script

per usare processori su macchine diverse

```
#!/bin/bash
############################
##   The PBS directives   ##
############################
#PBS –q studenti
#PBS -N Hello
#PBS -o Hello.out
#PBS -e Hello.err

#PBS -l nodes=8:ppn=8

##############################################
#   -q  coda su cui va eseguito il job  #
#    -l   numero di nodi richiesti  #
#    -N  nome job(stesso del file pbs)   #
#    -o, -e nome files contenente l'output  e gli errori  #
##############################################
#                                #
#          stampa di  qualche informazione sul job     #
#                                #
##############################################
```

#PBS -j oe
per far confluire lo standard error nello standard output

8 nodi, 8 processori (8 processori per nodo)

Modifiche al job-script

```
echo PBS: qsub is running on $PBS_O_HOST
echo PBS: originating queue is $PBS_O_QUEUE
echo PBS: executing queue is $PBS_QUEUE
echo PBS: working directory is $PBS_O_WORKDIR
echo PBS: execution mode is $PBS_ENVIRONMENT
echo PBS: job identifier is $PBS_JOBID
echo PBS: job name is $PBS_JOBNAME
echo PBS: node file is $PBS_NODEFILE
echo PBS: number of nodes is $NNODES
echo PBS: current home directory is $PBS_O_HOME
echo PBS: PATH = $PBS_O_PATH

echo ------------------------------------------------------
echo 'Job reserved node(s): ‘
cat $PBS_NODEFILE
echo ------------------------------------------------------
```

Con questo **cat** vedete tutti processori che vi siete riservati

In questo file ci sono tutti i nodi riservati
(ogni nodo compare tante volte quanti sono i suoi processori, cioè ci sono 8x8=64 macchine)

Modifiche al job-script

```
sort -u $PBS_NODEFILE > hostlist

NCPU=`wc -l < hostlist`
echo ----------------------------------------------------
echo ' This job is allocated on '${NCPU}' cpu(s)' ' on hosts:‘
cat hostlist
echo ----------------------------------------------------

PBS_O_WORKDIR=$PBS_O_HOME/ProgettoHello

echo ----------------------------------------------------
echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/Hello $PBS_O_WORKDIR/Hello.c“

/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/Hello $PBS_O_WORKDIR/Hello.c

echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -np $NCPU $PBS_O_WORKDIR/Hello"
/usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist  –np $NCPU $PBS_O_WORKDIR/Hello
```

Sort
ordina le righe del file
-u
prende solo una volta quelle uguali

L’output viene scritto in **hostlist**, in cui ci saranno le diverse macchine ma una sola volta (8 macchine)

Modifiche al job-script

```
sort -u $PBS_NODEFILE > hostlist

NCPU=`wc -l < hostlist`
echo ----------------------------------------------------
echo ' This job is allocated on '${NCPU}' cpu(s)' ' on hosts:‘
cat hostlist
echo ----------------------------------------------------

PBS_O_WORKDIR=$PBS_O_HOME/ProgettoHello

echo ----------------------------------------------------
echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O

/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/H

echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machine
/usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist  –np $NCPU $PBS_O_WORKDIR/Hello
```

wc -l
conta le righe del file **hostlist (8)**

Nel file ogni riga è una macchina che vogliamo usare, quindi, l’output del comando è il nostro numero di cpu.

NOTA:
Nell’altro file PBS visto, si faceva la stessa operazione direttamente sul file $PBS_NODEFILE. (che ora però ha 64 righe)

Modifiche al job-script

```
sort -u $PBS_NODEFILE > hostlist

NCPU=`wc -l < hostlist`
echo ----------------------------------------------------
echo ' This job is allocated on '${NCPU}' cpu(s)' ' on hosts:‘
cat hostlist
echo ----------------------------------------------------

PBS_O_WORKDIR=$PBS_O_HOME/ProgettoHello

echo ----------------------------------------------------
echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/Hello $PBS_O_WORKDIR/Hello.c“

/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/Hello $PBS_O_WORKDIR/Hello.c

echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -np $NCPU $PBS_O_WORKDIR/Hello"
/usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist  –np $NCPU $PBS_O_WORKDIR/Hello
```

Con questo **cat** vedete tutti processori che **userete**

Modifiche al job-script

```
sort -u $PBS_NODEFILE > hostlist

NCPU=`wc -l < hostlist`
echo ----------------------------------------------------
echo ' This job is allocated on '${NCPU}' cpu(s)' ' on hosts:‘
cat hostlist
echo ----------------------------------------------------

PBS_O_WORKDIR=$PBS_O_HOME/ProgettoHello

echo ----------------------------------------------------
echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/Hello $PBS_O_WORKDIR/Hello.c“

/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/Hello $PBS_O_WORKDIR/Hello.c

echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -np $NCPU $PBS_O_WORKDIR/Hello"
/usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist  –np $NCPU $PBS_O_WORKDIR/Hello
```

RICORDATE di dare il giusto file a mpiexec!!!!

ATTENZIONE!!!!!

Utilizzate questo secondo tipo di script

SOLO

quando sarete certi che il vostro codice funziona e realizza sempre la somma che volete

Istruzioni da aggiungere al programma

per ottenere il tempo di esecuzione in output

MPI fornisce una funzione semplice:

double MPI_Wtime()

che restituisce un tempo in secondi (double)

- L'utilizzo di questa funzione all'interno del vostro codice è piuttosto semplice…
- Vediamo un esempio attraverso l'algoritmo per la somma di n numeri, già visto a lezione

Dichiarazioni

```
#include <stdio.h>
#include “mpi.h”
main(int argc, char *argv[]){
    int menum, nproc,… ;
    int n, nloc, tag, i,…;
    int *x, *xloc;
    double t0, t1, time;        /*servono a tutti i processi*/
    double timetot;             /*serve solo a P0*/
    MPI_Status status;
…
```

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);
}
```

Calcolo Locale

…

…

…

…

MPI_Barrier(MPI_COMM_WORLD);

t0=MPI_Wtime();

/*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);
}
t1=MPI_Wtime();
```

Stampa del risultato
(I versione)

```
…
 time=t1-t0;        /*ora ogni processore SA il proprio tempo*/
 printf("Sono %d: Tempo impiegato: %e secondi\n",menum,time);
…
MPI_Reduce(&time,&timetot,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);
 /*ora P0 sa anche il MASSIMO tra tutti i tempi dei processi*/
…
/*se ci basta che la stampi solo un processore (P0)*/
if (menum==0){
        printf("\nSomma totale=%d\n", sum);
        printf("Tempo totale impiegato: %e secondi\n",timetot);
…
}
```

Fine Note