# "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

per usare processori su macchine diverse

```
#!/bin/bash
######################################
## The PBS directives ##
#PBS -q studenti
#PBS-N Hello
#PBS -o Hello.out
                                       #PBS -i oe
#PBS -e Hello.err ←
                                       per far confluire lo standard error nello standard output
#PBS -I nodes=8:ppn=8
                                                 8 nodi, 8 processori (8 processori per nodo)
  -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
```

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)

sort -u \$PBS\_NODEFILE > hostlist

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

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

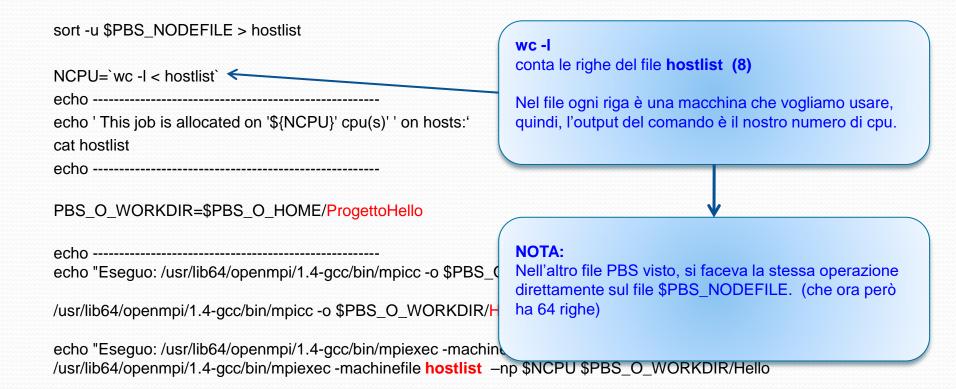
#### Sort

ordina le righe del file

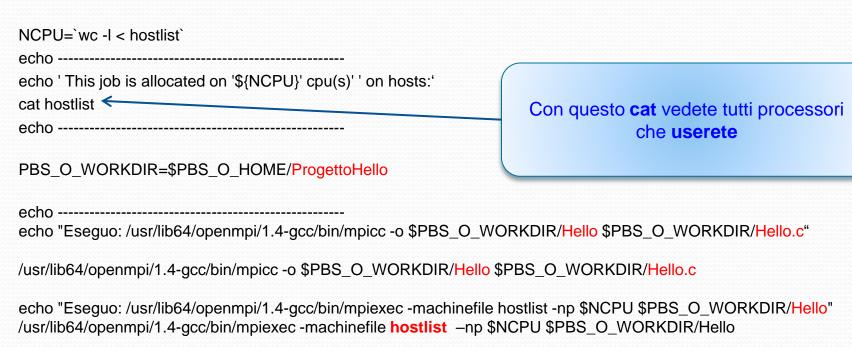
-u

prende solo una volta quelle uguali

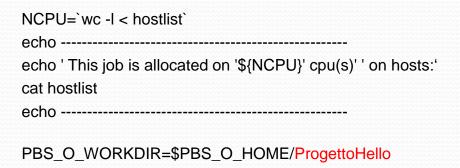
L'output viene scritto in **hostlist**, in cui ci saranno le diverse macchine ma una sola volta (8 macchine)



sort -u \$PBS\_NODEFILE > hostlist



sort -u \$PBS\_NODEFILE > hostlist



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

echo -----

echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o \$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 -machipefile hostlist -np \$NCPU \$PBS\_O\_WORKDIR/Hello" /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -np \$NCPU \$PBS\_O\_WORKDIR/Hello

#### 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++)</pre>
      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)

```
/*ora ogni processore SA il proprio tempo*/
time=t1-t0;
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