Piccola guida a MPI-Fortran

Davide Giacca -:- *dave@davethewave.it*

0.1 Chiamate di base

Inizio e chiusura comunicazioni

```
integer :: ierr
call MPI_INIT(ierr)
call MPI_FINALIZE(ierr)
```

Comunicatore di default

```
MPI_COMM_WORLD
```

Conoscere etichetta del processo e grandezza del comunicatore

```
integer :: id_processo, numero_processi, ierr
call MPI_COMM_RANK(MPI_COMM_WORLD, id_processo,ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, numero_processi,ierr)
```

0.2 Tipi di dato

Tipo di dato Fortran	*Tipo di dato MPI*
integer	MPI_INTEGER
real	MPI_REAL
double precision	MPI_DOUBLE_PRECISION
complex	MPI_COMPLEX
double complex	MPI_DOUBLE_COMPLEX
logical	MPI_LOGICAL
character(1)	MPI_CHARACTER

0.3 Chiamate punto-punto bloccanti

```
integer, dimension(MPI_STATUS_SIZE) :: status
call MPI_SEND(variabile_da_mandare, lunghezza, tipo_dato,
              destinazione,tag, comunicatore, error)
call MPI_RECV(variabile_in_cui_ricevere, lunghezza, tipo_dato,
              sorgente, comunicatore, status, error)
```

0.4 Chiamate punto-punto non bloccanti

```
integer, dimension(MPI_STATUS_SIZE) :: status
integer request
call MPI_ISEND(variabile_da_mandare, lunghezza, tipo_dato,
              destinazione,tag, comunicatore, request, ierr)

call MPI_IRECV(variabile_in_cui_ricevere, lunghezza, tipo_dato,
              sorgente, comunicatore, request ,ierr)
...
call MPI_WAIT(request, status)
```

0.5 Chiamate uno-molti, molti-uno, molti-molti

Sincronizza tutti i processi... solo per debug

```
call MPI_BARRIER(comunicatore)
```

Manda a tutti una variabile

```
call MPI_BCAST(variabile,lunghezza,tipo,sorgente,comunicatore,ierr)
```

Manda pezzi di un vettore a tutti

```
call MPI_SCATTER(variabile_da_mandare, contatore_invio, tipo_dato_invio,
     variabile_di_ricezione, contatore_di_ricezione,
     tipo_di_ricezione, sorgente, comunicatore)
```

Raccoglie i pezzi di un vettore da tutti

```
call MPI_GATHER(variabile_da_mandare, contatore_invio,
         tipo_dato_invio, variabile_ricezione, contatore_ricezione,
         tipo_dato_ricezione, sorgente, comunicatore)
```

All to All

```
P_0 [a_0,a_1,a_2,a_3]      P_0 [a_0,b_0,c_0,d_0]
P_1 [b_0,b_1,b_2,b_3]   => P_1 [a_1,b_1,c_1,d_1]
P_2 [c_0,c_1,c_2,c_3]      P_2 [a_2,b_2,c_2,d_2]
P_3 [d_0,d_1,d_2,d_3]      P_3 [a_3,b_3,c_3,d_3]
```

```
call MPI_ALLTOALL(variabile_da_mandare, contatore_invio,
         tipo_dato_invio, variabile_ricezione, contatore_ricezione,
         tipo_dato_ricezione, comunicatore)
```

0.6 Tempo di calcolo

```
integer :: t0,t1,count_rate,count_max
real::secondi0,secondi1,tempototale
call SYSTEM_CLOCK(t0,count_rate,count_max)
//qui il corpo programma di prima
call SYSTEM_CLOCK(t1,count_rate,count_max)
secondi1=REAL(t1)/count_rate tempototale=secondi1-secondi0
write(*,*) "tempo di esecuzione",tempototale
```

0.7 Compilare e lanciare un programma in ambiente LAM-MPI

Compilare

```
 mpif77 -o nome_eseguibile sorgente
```

File lamboot.mio

```
# numero_ip cpu=numerocpu
mucca cpu=2
pollo
capra
```

Creare la LAM

```
 lamboot lamboot.mio
```

Lanciare l'eseguibile

```
 mpirun C nome_eseguibile
```