MPI_COMM_WORLD

Un comunicatore e' un handler che rappresenta un gruppo di processiin grado di comunicare

- L'handler del comunicatore di default e' MPI_COMM_WORLD
- Ogni processo puo' utilizzare MPI_COMM_WORLD solo dopo la chiamata a MPI_Init
- Un comunicatore definisce un contesto di comunicazione
- All'interno di un comunicatore ogni processo ha un identificativo unico
- Il nome del comunicatore da utilizzare e' un argomento obbligatorio di tutte le funzioni di comunicazione
- E' possibile utilizzare nello stesso programma piu' di un comunicatore
- In generale, due processi possono comunicare solo se fanno parte dello stesso comunicatore
- In molti casi reali e' sufficiente utilizzare il solo comunicatore di default MPI_COMM_WORLD

Info dal comunicatore

Un processo puo' ottenere alcune informazioni utili dal comunicatore di cui fa parte

- Communicator size
  - Un processo puo' determinarela dimensione di un comunicatore di cui fa parte con una chiamata alla funzione MPI_Comm_size
  - La “size” di un comunicatore e' un intero
- Process rank
  - Un processo puo' determinare il proprio identificativo(rank) in un comunicatore con una chiamata a MPI_Comm_rank
  - I rank dei processi che fanno parte di un comunicatore sono numeri interi, consecutivi ed il piu' piccolo e' lo 0

Binding

```
int MPI_Comm_size(MPI_Comm comm, int *size)
```

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

In ambo i casi **comm** e' il comuincatore di cui si vole conoscere la “size” o del quale si vuole conoscere il rank del processo chiamante. (perche' le due funzioni necessitano puntatori per **rank** e **size** ?)

Provate adesso a realizzare compilare ed eseguire un programma che determini per ogni processo rank e size del comunicatore MPI_COMM_WORLD. (**primompi**)

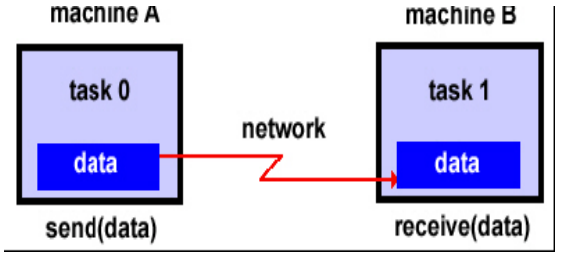
Processor Name

```
int MPI_Get_processor_name(char *name, int
*resultlen);
```

E' possibile cosi' determinare il nome del processore nel quale sta girando il processo . **resultlen** (< **MPI_MAX_PROCESSOR_NAME**) contiene il numero di caratteri in **name**,

Modificare il programma primompi aggiungendo anche la stampa a video del nome del “processore” (**primompiext**)

Comunicazione



Nel modello di programmazione message passing

- ogni processo accede direttamente solo alla propria area di memoria
- la cooperazione tra processi avviene attraverso operazioni esplicite di comunicazione
- l'operazione elementare di comunicazione e' la comunicazione "point-to-point"
- la comunicazione point-to-point vede coinvolti due processi:
  - Il processo sender invia un messaggio
  - Il processo receiver riceve il messaggio inviato
- I dati non sono trasferiti senza la partecipazione esplicita dei due processi

Messaggio

Cos'e' un messaggio

- In una generica operazione di send e receive un messaggio consiste in un certo blocco di dati da trasferire tra i processi

Un messaggio e' costituito da:

- Envelope contiene l'identificativo del messaggio, la sorgente e la destinazione dello stesso
- Body contiene i dati da inviare

L' envelope contiene:

- source: l'identificativo del processo che lo invia
- destination: l'identificativo del processo che lo deve ricevere
- communicator: l'identificativo del gruppo di processi cui appartengono sorgente e destinazione del messaggio
- tag: un identificativo che classifica il messaggio

Il body contiene:

- buffer: i dati del messaggio
- datatype: il tipo di dati contenuti nel messaggio
- count: il numero di occorrenze di tipo datatype contenute nel messaggio

I principali MPI_Datatype

MPI_Datatype	C type
MPI_CHAR	signed char
MPI_DOUBLE	double
MPI_FLOAT	float
MPI_INT	signed int

Comunicazione punto a punto

- E’ la funzionalita' fondamentale disponibile nella libreria MPI
- Coinvolge sempre due processi: uno invia un messaggio e l’altro lo riceve
- Per mandare un messaggio il processo source effettua una chiamata ad una funzione MPI, in cui specifica il rank del processo destination nell’appropriato comunicatore (ad esempio MPI_COMM_WORLD)
- Anche il processo destination deve effettuare una chiamata ad una funzione per ricevere il messaggio

MPI_Send

Il processo sorgente effettua una chiamata ad una primitiva con la quale specifica in modo univoco envelope e body del messaggio da inviare:

- l'identita' della sorgente e' implicita (il processo che effettua l'operazione)
- Gli altri elementi che completano la struttura del messaggio
  - identità della destinazione
  - comunicatore da utilizzare
  - dati

  sono determinati esplicitamente dagli argomenti che il processo sorgente passa alla funzione di send

MPI_Send

```
int MPI_Send(void *buf, int count, MPI_Datatype dtype,
             int dest, int tag, MPI_Comm comm)
```

Tutti gli argomenti sono di input

- **buf** e' l'indirizzo iniziale del send buffer
- **count** e' di tipo int e contiene il numero di elementi del send buffer
- **dtype** e' di tipo MPI_Datatype e descrive il tipo di ogni elemento del send buffer
- **dest** e' di tipo int e contiene il rank del receiver all'interno del comunicatore comm
- **tag** e' di tipo int e contiene l'identificativo del
- messaggio
- **comm** e' di tipo MPI_Comm ed e' il comunicatore in cui avviene la send

MPI_Recv

Il processo destinazione chiama una primitiva, dai cui argomenti e' determinato “in maniera univoca” l’envelope del messaggio da ricevere

MPI confronta l’envelope del messaggio in ricezione con quelli dell’insieme dei messaggi ancora da ricevere (pending messages) e

- se il messaggio e' presente viene ricevuto
- altrimenti l’operazione non puo' essere completata fino a che tra i pending messages ce ne sia uno con l’envelope richiesto

Il processo di destinazione deve disporre di un’area di memoria sufficiente per salvare il body del messaggio

MPI_Recv

```
int MPI_Recv(void *buf, int count, MPI_Datatype dtype,
  int src, int tag, MPI_Comm comm, MPI_Status *status)
```

- [OUT] **buf** e' l'indirizzo iniziale del receive buffer
- [IN] **count** e' di tipo int e contiene il numero di elementi del receive buffer
- [IN] **dtype** e' di tipo MPI_Datatype e descrive il tipo di ogni elemento del receive buffer
- [IN] **src** e' di tipo int e contiene il rank del sender all'interno del comunicatore comm
- [IN] **tag** e' di tipo int e contiene l'identificativo del messaggio
- [IN] **comm** e' di tipo MPI_Comm ed e' il comunicatore in cui avviene la send
- [OUT] **status** e' di tipo MPI_Status e conterra' informazioni sul messaggio che e' stato ricevuto

Esempio di spedizione e ricezione dati (**sendrecv**)

Esercizio: 1 processo riempie e spedisce un vettore di float contenete numeri random

Esercizio ring: Scrivere un programma MPI in cui

- Il processo 0 legge da standard input un numero intero positivo A

  1. All'istante T1 il processo 0 invia A al processo 1
  2. All'istante T2 il processo 1 invia A al processo 2
  3. ….
  4. All'istante TN il processo N-1 invia A al processo 0

- Il processo 0 decrementa e stampa il valore di A se A è ancora positivo torna al punto 1, altrimenti termina l'esecuzione