Comunicazione

La comunicazione point-to-point e' la funzionalita' di comunicazione fondamentale disponibile in MPI

- Concettualmente la comunicazione point-to-point e' molto semplice:
  - Un processo invia un messaggio
  - Un altro processo lo riceve

- Nella pratica le cose sono meno semplici:
  - Il ritorno da una primitiva di comunicazione (ovvero la possibilita' di eseguire l'istruzione successiva) puo' avvenire quando
    - la comunicazione e' completata (**blocking send**)
    - la comunicazione e' avviata (**non-blocking send**)
  - Esistono diverse modalità di completamento di un'operazione di send

Modalita' di comunicazione

Ottica globale

- **SINCRONA:** il mittente sa se il messaggio e' arrivato o meno (ad esempio fax)
- **ASINCRONA:** Il mittente non sa se il messaggio e' arrivato o meno (ad esempio lettera)

Ottica locale

- **BLOCCANTE:** il controllo e' restituito al processo che ha invocato la primitiva di comunicazione solo quando la stessa e' stata **completata**
- **NON BLOCCANTE:** il controllo e' restituito al processo che ha invocato la primitiva di comunicazione quando la stessa e' stata eseguita. Il controllo sull'effettivo **completamento** della comunicazione deve essere fatto in seguito, nel frattempo il processo può eseguire altre operazioni

Completamento della comunicazione

Quando un comunicazione si puo' considerare completata ?

- Una comunicazione point-to-point e' considerata **completata localmente** quando le aree di memoria coinvolte nel trasferimento di dati possono essere riutilizzate in modo “sicuro”
  - dal punto di vista di chi spedisce dati la comunicazione e' completata se l'area di memoria puo' essere sovrascritta
  - dal punto di vista di chi riceve dati la comunicazione e' completata se le variabili ricevute possono essere utilizzate
- Dal punto di vista **globale** una comunicazione puo' essere considerata completata se e solo se tutti i processi coinvolti hanno completato le rispettive operazioni relative alla comunicazione. Parafrasando un'operazione di comunicazione e' completata globalmente se e solo se e' completata localmente su tutti i processi coinvolti

Classificare i tipi di comunicazione

Modalita' di blocking

SEND Non-Blocking: Ritorna “immediatamente”. Il buffer del messaggio non deve essere sovrascrittosubito dopo il ritorno al processo chiamante, ma si deve controllare che la SEND sia completata localmente.

SEND Blocking: Ritorna quando la SEND e' completata localmente. Il buffer del messaggio puo' essere sovrascritto subito dopo il ritorno al processo chiamante.

RECEIVE Non-Blocking: Ritorna “immediatamente”. Il buffer del messaggio non deve essere letto subito dopo il ritorno al processo chiamante, ma si deve controllare che la RECEIVE sia completata localmente.

RECEIVE Blocking: Ritorna quando la RECEIVE e' completata localmente. Il buffer del messaggio puo' essere letto subito dopo il ritorno.

Modalita' di comunicazione

La modalita' di una comunicazione punto-punto specifica quando un'operazione di SEND puo' iniziare a trasmettere e quando puo' ritenersi completata. MPI prevede 4 modalita':

- **Syncronous Send:** La SEND puo' iniziare (a trasmettere) anche se la RECEIVE corrispondente non e' iniziata. Tuttavia, la SEND e' completata solo quando si ha garanzia che il processo destinatario ha eseguito e completato la RECEIVE.
- **Buffered Send:** Simile alla modalita' sincrona per l'inizio della SEND. Tuttavia, il completamento e' sempre
- indipendente dall'esecuzione della RECEIVE. (Il messaggio pue' essere bufferizzato per garantire il funzionamento)
- **Standard Send:** La SEND puo' iniziare (a trasmettere) anche se la RECEIVE corrispondente non è iniziata. La semantica del completamento puo' essere sincrona o buffered.
- **Ready Send:** La SEND puo' iniziare (a trasmettere) assumendo che la corrispondente RECEIVE e' iniziata. Tuttavia, il completamento e' sempre indipendente dall'esecuzione della RECEIVE.
- **Receive:** e' completata quando il messaggio e' arrivato

Comunicazione

In MPI vi sono diverse combinazioni possibili tra SEND/RECEIVE sincrone e asincrone, bloccanti e non bloccantiOperazione di SEND:

SEND 8 tipi

MPI_[-,I][-,R,S,B]send
[-,I] = modalita' di blocking
[-,R,S,B] = modalita' di comunicazione

RECEIVE 2 tipi

MPI_[-,I]recv
[-,I] = modalita' di blocking

MPI_Ssend

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

- 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

Il processo mittente si blocca finche' il messaggio non e' stato ricevuto dal destinatario. E' la moidalita' di comunicazione piu' semplice ed affidabile, ma si possno avere notevoli perdite di tempo.

MPI_Bsend

- Una buffered send e' completata immediatamente, non appena il processo ha copiato il messaggio su un opportuno buffer di trasmissione
- Il programmatore non puo' assumere la presenza di un buffer di sistema allocato per eseguire l'operazione, ma deve effettuare un'operazione di
  - BUFFER_ATTACH per definire un'area di memoria di dimensioni opportune come buffer per il trasferimento di messaggi
  - BUFFER_DETACH per rilasciare le aree di memoria di buffer utilizzate
- Pro
  ritorno immediato dalla primitiva di comunicazione
- Contro
  E' necessaria la gestione esplicita del buffer
  Implica un'operazione di copia in memoria dei dati da trasmettere

MPI_Bsend

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

- 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

Gestione dei buffer

```
MPI_Buffer_attach(void *buf, int size)
```

Consente al processo sender di “allocare” il buffer per una successiva chiamata di MPI_Bsend

- .buf e' l’indirizzo iniziale del buffer da “allocare”
- .size e' un int e contiene la dimensione in byte del buffer da “allocare” (size + MPI_BSEND_OVERHEAD).

```
MPI_Buffer_detach(void *buf, int *size)
```

Consente di “deallocare” il buffer creato con MPI_Buffer_attach

- .buf e' l’indirizzo iniziale del buffer da allocare
- .size e' un int e contiene la dimensione in byte del buffer

MPI_Send

- Una standard send e' completata quando il processo sender puo' riutilizzare l'area di memoria contenente i dati oggetto del trasferimento
- Il programmatore non puo' assumere che l'operazione sara' completata prima che:
  - il processo destinazione inizi la ricezione
  - il processo destinazione termini la ricezione
- Una standard send puo' essere implementata come
  - una syncronous send
  - una buffered send, utilizzando pero' un buffer di sistema
- Nel caso in cui il messaggio da inviare non entri completamente nel buffer di sistema, la standard send si comporta come una syncronous send

MPI_Recv

Standard RECEIVE blocking

- Il processo destinazione usa un solo tipo di RECEIVE blocking, indipendentemente dalla particolare funzionalita' di SEND utilizzata nel processo sorgente
- Gli argomenti di tipo envelope sono necessari per richiedere ad MPI la ricezione del giusto messaggio
- I dati ricevuti sono salvati in memoria a partire dalla locazione argomento della funzione di receive
- Al completamento dell'operazione i dati ricevuti posso essere utilizzati

Esercizio (18_time): provate a fare il send di una matrice usando in sequenza MPI_Ssend, MPI_Send, MPI_Bsend

Deadlock

- Si parla di deadlock quando 2 (o piu') processi sono bloccati ed ognuno e' in attesa dell'altro per riprendere l'esecuzione
- Utilizzare operazioni di comunicazione di tipo blocking senza un corretto protocollo di comunicazione puo' portare a situazioni di deadlock

- Esempio:

```
if (myrank = 0)
  Standard SEND A to Process 1
  RECEIVE B from Process 1
else if (myrank = 1)
  Standard SEND B to Process 0
  RECEIVE A from Process 0
endif
```

- in quali condizioni genera un deadlock?

Deadlock

- L'esempio precedente genera un deadlock se la standard send e' implementata come una syncronous send
- In questo caso semplice il deadlock si evita facendo attenzione all'ordine delle chiamate:

```
if (myrank = 0)
  Standard SEND A to Process 1
  RECEIVE B from Process 1
else if (myrank = 1)
  RECEIVE A from Process 0
  Standard SEND B to Process 0
endif
```

MPI_Sendrecv

- La funzionalita' MPI Send-Receive e' utile quando un processo deve sia inviare che ricevere dati
- In tali casi una MPI Send-Receive consente di utilizzare un unico costrutto, evitando situazioni di deadlock
- MPI Send-Receive e' un'estensione delle comunicazioni point-to-point
- La MPI Send-Receive e' completamente compatibile con le altre funzionalita' di comunicazione point-to-point, ovvero puo' essere utilizzata per:
  - Inviare un messaggio che sara' ricevuto tramite una funzione di receive
  - Ricevere un messaggio inviato tramite una funzione di send
- E' utile per implementare un pattern di comunicazione di tipo shift, utilizzando funzioni blocking

MPI_Sendrecv

```
int MPI_Sendrecv(void *sbuf,int scount, MPI_Datatype
s_dtype, int dest, int stag, void *dbuf, int dcount,
MPI_Datatype d_type, int src, int dtag,MPI_Comm comm,
MPI_Status *status)
```

- [IN] sbuf e' l'indirizzo iniziale del send buffer
- [IN] scount contiene la size del send buffer
- [IN] s_dtype descrive il tipo di ogni elemento del send buffer
- [IN] dest e' il rank del receiver all'interno del comunicatore comm
- [IN] stag e' l'identificativo del messaggio da inviare
- [OUT] dbuf e' l'indirizzo iniziale del receive buffer
- [IN] dcount contiene la size del receive buffer
- [IN] d_dtype descrive il tipo di ogni elemento del receive buffer
- [IN] src e' il rank del receiver all'interno del comunicatore comm
- [IN] dtag e' l'identificativo del messaggio da ricevere
- [IN] comm e' il comunicatore in cui avviene la sendrecv
- [OUT] status conterra' informazioni sul messaggio

Esempio (18_sendrecv): semplice esempio di sendrecv

Esercizio (19_ring): ogni processo spedisce il proprio rank a destra e riceve quello del vicino, usando MPI_Sendrecv. I processi avranno una topologia ad anello come in figura. Usare MPI_sendrecv