UNIVERSITÀ DI PARMA
DIPARTIMENTO DI SCIENZE MATEMATICHE, FISICHE ED INFORMATICHE
Corso di Laurea in Informatica

Programmazione a memoria distribuita con MPI

Programmazione parallela e HPC - a.a. 2023/2024

Roberto Alfieri

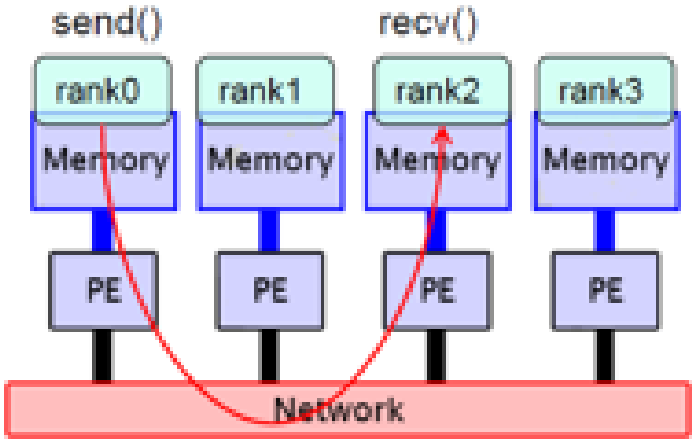
Programmazione Parallela e HPC: sommario

Message Passing

Nei sistemi a memoria distribuita ogni processo (**task**) ha la sua memoria locale con uno spazio di indirizzamento separato.

Per accedere a dati remoti, residenti nello spazio di indirizzamento di altri task, il programmatore deve esplicitamente gestire la comunicazione attraverso una libreria per lo scambio di messaggi (**message passing**).

La libreria maggiormente utilizzata per lo scambio di messaggi tra processi in un programma parallelo è MPI (**Message Passing Interface**).



Tutorial e link esterni: LLNL

MPI

MPI (Message Passing Interface) è la specifica emanata dall' MPI Forum per lo sviluppo di una libreria standard per lo scambio di messaggi.
L'MPI Forum, composto da vendor di sistemi HPC, ricercatori in informatica e sviluppatori software, ha emanato a partire dal 1994 diverse versioni dello standard.

La libreria MPI fornisce funzioni o metodi in accordo con i linguaggi supportati che sono C, C++ e Fortran.

MPI-1 versioni 1.0 (06/94) 1.1 (06/95) 1.3 (07/08)
circa 115 routine; ambiente run-time statico (i processi e la loro allocazione non e' modificabile)

MPI-2 versioni 2.0 (07/97) 2.1 (09/08) 2.2 (09/09)
ambiente dinamico, I/O scalabile su file, comunicazioni collettive tra 2 gruppi di processi. I linguaggi ufficialmente supportati sono C/C++ e Fortran

MPI-3 versioni 3.0 (09/12) 3.1 (06/15)
operazioni collettive non bloccanti

MPI-4 versioni 4.0 (06/21) 4.1 (11/23)

Implementazioni MPI

Gli standard sono supportati in diverse implementazioni, le principali sono:

OpenMPI
e' un progetto che combina tecnologie e risorse di diversi altri progetti (FT-MPI, LA-MPI, LAM/MPI, and PACX-MPI) con lo scopo di costruire la miglior libreria Message Passing Interface (MPI) disponibile.

MPICH
è una implementazione libera e portabile di MPI, sviluppata su iniziativa di Argonne National Laboratory (ANL). La prima implementazione di MPICH si chiama MPICH1 che implementa lo standard MPI-1.1. Attualmente l'ultima implementazione si chiama MPICH2 la quale implementa lo standard MPI-2.0

Intel MPI Library https://software.intel.com/en-us/intel-mpi-library
Implementazione commerciale sviluppata da Intel. Supporto per MPI-3 e MPI-4

Tutorial e link esterni: LLNL

Comunicazioni

Tipi di comunicazioni

- Comunicazioni punto-punto send/receive (sincrone, asincrone , bufferizzate)
- Operazioni collettive tra gruppi di nodi (communicators)
  - Comunicazioni (broadcast)
  - Trasferimento dati (gather/scatter)
  - Sincronizzazione (barrier)

Canali di comunicazione

Possono avvenire utilizzando diversi tipi di Network. I principali sono:

- TCP (disponibile su tutti i sistemi, ma lento)
- Shared Memory (sistemi multiprocessore)
- Intel Omnipath (high speed)
- Infiniband (high speed)

Formato dei messaggi

Nelle comunicazioni vengono inviati messaggi che sono composti di 2 parti:

Envelope
- source: rank of the sender
- destination: rank of the receiver
- tag: ID of the message (from 0 to MPI_TAG_UB)
- communicator: context of the communication

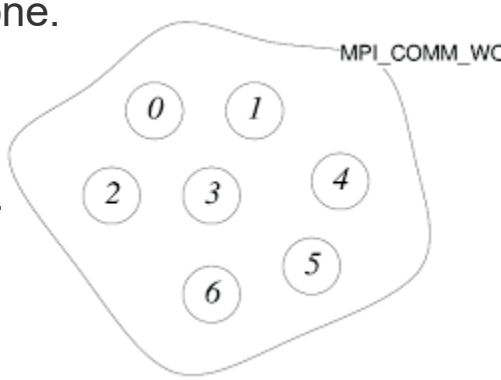
Body
- type: MPI datatype
- length: number of elements
- buffer: array of elements

Tipi di dati nei messaggi

tipo MPI	Tipo C	Byte
MPI_CHAR	signed char	1
MPI_SHORT	signed short int	2
MPI_INT	signed int	4
MPI_LONG	signed long int	4
MPI_UNSIGNED_CHAR	unsigned char	1
MPI_UNSIGNED_SHORT	unsigned short	1
MPI_UNSIGNED	unsigned int	4
MPI_UNSIGNED_LONG	unsigned long int	4
MPI_FLOAT	float	4
MPI_DOUBLE	double	8
MPI_LONG_DOUBLE	long double	12
MPI_BYTE	8 binary digit	1
MPI_PACKED	packed with MPI_Pack() unpacked with MPI_Unpack()	

MPI communicator

Un Communicator è l'insieme di processi (task) coinvolti nella comunicazione.
Esiste un Communicator di default, denominato **MPI_COMM_WORLD**
che include tutti i task disponibili (quelli richiesti all'avvio).
Ogni task ha un identificativo numerico **Rank** assegnato automaticamente.
Il processo che lancia il Job e attiva gli altri ha rank 0 (detto master o root).



La libreria MPI fornisce funzioni per le gestione del communicator:

```
MPI_Init (&argc,&argv)  //Inizializza l'ambiente MPI
MPI_Comm_size (MPI_COMM_WORLD,&size) // scrive in size il numero di task del communicator
MPI_Comm_rank (MPI_COMM_WORLD,&rank) // scrive in rank il Rank del task chiamante
MPI_Get_processor_name(&name,&namelength) //scrive nome dell'host del task chiamante
MPI_Finalize()          // Chiude l'ambiente MPI
```

Misura del tempo

I timer standard non sono adeguati a causa della risoluzione insufficiente e della mancanza di portabilità.

MPI_Wtime() ritorna il tempo corrente misurato a partire da un tempo prefissato

Comunicazioni punto-punto

Le primitive di base per la comunicazione punto-punto sono MPI_send() e MPI_recv(), bloccanti:

MPI_Recv (&buf,count,datatype,source,tag,comm,&status)
RECEIVE: Si sblocca quando il dato atteso è disponibile

MPI_Send (&buf,count,datatype,dest,tag,comm)
SEND: Si sblocca quando il buffer di spedizione è stato svuotato

where:
- buf is the pointer to the message to be sent (application buffer)
- count is the number of elements in the message
- datatype specifies the type of the elements in the message
-- Source is the rank of the sender.
Il ricevente può indicare la wildcard MPI_ANY_SOURCE
-- Dest is the rank of the receiver
-- tag is a non-negative integer whose purpose is left to the user.
Il ricevente può indicare la wildcard MPI_ANY_TAG
.. comm is the communicator
-- status contains info about the envelope of the message to be received

esempio: send/recv di numero intero

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char **argv)
{
  MPI_Status status;
  int rank, size;
  int data_int;    /* data to communicate */
  MPI_Init(&argc, &argv);   /* initialize MPI */
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  if (rank == 0) {
       data_int = 10;
       MPI_Send(&data_int, 1, MPI_INT, 1, 666, MPI_COMM_WORLD);
    } else if (rank == 1) {
       MPI_Recv(&data_int, 1, MPI_INT, 0, 666, MPI_COMM_WORLD, &status);
       printf("Process 1 receives %d from process 0.\n", data_int);
    }
  MPI_Finalize();
  return 0;
}
```

Esempio: send/recv di array di float

```
#include <stdio.h>
#include "mpi.h"
#define MSIZE 10

int main(int argc, char **argv)
{
  MPI_Status status;
  int rank, size;
  int i, j;
  float a[MSIZE]; /*data to communicate */
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if (rank == 0) {
     for (i = 0; i<MSIZE; i++)
        a[i] = (float) i;
    MPI_Send(a, MSIZE,MPI_FLOAT, 1, 666, MPI_COMM_WORLD);
   } else if (rank == 1) {
     MPI_Recv(a, MSIZE, MPI_FLOAT, 0, 666, MPI_COMM_WORLD, &status);
     printf("Process 1 receives the following array from process 0.\n");
     for (i = 0; i<MSIZE; i++)
          printf("%6.2f\n", a[i]);
  }
  MPI_Finalize();
  return 0;
}
```

MPI_Sendrecv

Questa funzione bloccante esegue una send e una receive. Si blocca fino a quando il buffer del mittente è libero per il riuso e il buffer applicativo del ricevente contiene il messaggio ricevuto.

```
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)
```

La prima metà della lista di argomenti è relativa a send, la seconda metà è relativa a receive.

Esempio: MPI_Sendrecv() circolare

I rank, disposti circolarmente, inviano il proprio numero di rank al rank successivo.

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv)
{
 MPI_Status status;
 int rank, size;
 int prev_rank, next_rank;
int data_send, data_recv; /* data to communicate */

MPI_Init(&argc, &argv); /* initialize MPI */
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
 prev_rank = (rank-1+size) % size;
 next_rank = (rank+1)      % size;
 data_send = rank;

 MPI_Sendrecv(&data_send, 1, MPI_INT, next_rank, 666,
                    &data_recv,  1, MPI_INT, prev_rank, 666,   MPI_COMM_WORLD, &status);

  printf("Process %d/%d receives %d from process %d.\n", rank, size, data_recv,
status.MPI_SOURCE);
 MPI_Finalize();
 return 0;
}
```

Altre Comunicazioni punto-punto bloccanti

MPI_Ssend (&buf,count,datatype,dest,tag,comm)
SYNCRONOUS SEND: si sblocca quando il destinatario ha iniziato a ricevere.

MPI_Bsend (&buf,count,datatype,dest,tag,comm)
BUFFERED SEND: Ritorna quando i dati sono copiati in un buffer specifico

MPI_Rsend (&buf,count,datatype,dest,tag,comm)
READY SEND: Da usare quando siamo certi che il destinatario è in ascolto

Comunicazioni punto-punto non bloccanti

MPI_Irecv(&buf,count,datatype,source,tag,comm,&request)
#I-RECEIVE: Ritorna immediatamente.

MPI_Isend (&buf,count,datatype,dest,tag,comm,&request)
I-SEND: Ritorna immediatamente senza attende che i dati vengano copiati nel buffer.

#request è un handle per verificare lo status della richiesta (vedi MPI_Wait e MPI_Test)
#La «i» sta per immediato.

MPI_Test (&request,&flag,&status)
#TEST: verifica lo stato di una request send/receive non bloccante. flag=1 -> operazione completato, flag=0 -> non completata

MPI_Wait (&request,&status)
#WAIT: come MPI_Test, ma si blocca finchè non termina la request

Le comunicazioni non bloccanti consentono di sovrapporre comunicazione e computazione, limitando i tempi di idle dei task.

Comunicazioni collettive

MPI fornisce primitive che coinvolgono tutti i task di un communicator e possono essere all-to-one, one-to-all e all-to-all:

MPI_Bcast (&buffer,count,datatype,root,comm)
#send di un messaggio da un processo a tutti gli altri (root spedisce, gli altri ricevono):

MPI_Scatter (&sendbuf,sendcnt,sendtype,&recvbuf, recvcnt,recvtype,root,comm)
#Un array su un processo root viene distribuito inviando un elemento ad ogni altro processo

MPI_Gather (&sendbuf,sendcnt,sendtype,&recvbuf,recvcount,recvtype,root,comm)
Da un array distribuito viene costruito un array sul processo root.

MPI_Allgather (&sendbuf,sendcount,sendtype,&recvbuf,recvcount,recvtype,comm)
Da un array distribuito viene costruito un array replicato su tutti i processi

MPI_Reduce (&sendbuf,&recvbuf,count,datatype,op,root,comm)
I dati da un array distribuito vengono elaborati con l'operazione 'op'; il risultato
è scritto nel processo root.

MPI_Allreduce (&sendbuf,&recvbuf,count,datatype,op,comm)
Il risultato della riduzione è replicato su tutti i processi (MPI_Reduce + MPI_Bcast)

Principali operazioni di reduce

OP	function	C-type
MPI_MAX	maximum	integer, float
MPI_MIN	minimun	integer, float
MPI_SUM	sum	integer, float
MPI_PROD	product	integer, float
MPI_LAND	logical AND	integer
MPI_BAND	bitwise AND	integer, MPI_BYTE
MPI_LOR	logical OR	integer
MPI_BOR	bitwise OR	integer, MPI_BYTE

Barriere

Le comunicazioni collettive devono essere eseguite da tutti i task del communicator; un task non può proseguire se tutti gli altri non hanno completato la comunicazione (barriera implicita).

E' comunque possibile creare esplicitamente una barriera che deve essere eseguita da tutti i task:
MPI_Barrier (MPI_Comm comm)

Comunicazioni one-side

In tutti gli esempi visti, la comunicazione coinvolgeva sempre due (o più entità). Infatti un'operazione di Send, doveva sempre avere una corrispondente operazione di Receive (discorso analogo per le comunicazioni collettive). Si imponeva in sostanza una sorta di sincronizzazione: i due processi dovevano invocare due funzioni complementari.

Con la funzionalità di One-sided Communication, invece, la comunicazione coinvolge unicamente un'entità. Vengono messe a disposizione infatti delle chiamate che consentono di leggere e scrivere nella memoria di un altro processo. Naturalmente, affinché ciò sia possibile, tale processo dovrà aver condiviso inizialmente una quantità di memoria.