Corso di Laurea in Informatica

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA

Corso
Calcolo Parallelo
Esercitazioni

Titolare del corso: prof. Luca Zanni (luca.zanni@unimore.it)

AA 2018/2019

MPI: concetti di base, intro al Laboratorio Zironi

Attività di laboratorio

Lab “Zironi”, primo piano dipartimento di Matematica.

iMac Linux (Ubuntu 16.04)
username: calcparX
password: _____

Collegarsi con ssh (usando username e passwd forniti) a aulad??.hpc.unimo.it
(con ??= 02 .. 13) .
Spostarsi nella dir /HOME/calcparX/CALCPAR1819.
Creare qui una dir nome.cognome.
Per scrivere un sorgente si può utilizzare vi da remoto (o altri editor disponibili, come gedit, nedit, emacs), oppure scrivere in locale e trasferire il file con scp.

Per compilare un sorgente seriale: icc –o eseguibile sorgente.c
Per lanciarlo: ./eseguibile
Useremo il compilatore intel

Attività di laboratorio

Per compilare un sorgente parallelo linkando automaticamente le librerie mpi:

mpiicc –o eseguibile sorgente.c

Per lanciare un eseguibile parallelo su 4 processori:

mpirun –np 4 eseguibile

ATTENZIONE
Ricordarsi di mantenere una copia di backup
(meglio ancora se più d'una)
del proprio lavoro in un luogo sicuro
(USB stick, PC di casa, directory di rete, …)

Installare libreria MPI sul vostro computer

Quasi tutte le distribuzioni linux hanno una implementazione pacchettizata del paragima MPI. Su Debian 8 installare la libreria è davvero facile:

apt-get install mpich

In seguito per questa implementazione (non Intel) basterà compilare con il comando:

mpicc –o eseguibile sorgente.c (!!! Non mpiicc !!!)

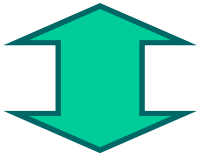
E per lanciare un eseguibile parallelo su 4 processori (come in lab):

mpirun –np 4 eseguibile

Nonostante la diversità dei compilatori il codice che impareremo a sviluppare è completamente portabile.
Potrete quindi sviluppare comodamente i vostri algoritmi a casa e testarli sulle macchine solo per ottenere **performance migliori**.

Lo standard MPI

Dalla seconda metà degli anni ’90 le macchine parallele hanno cominciato a diffondersi al di fuori dei loro ambienti tradizionali: in centri di ricerca e università più piccole, in aziende private, in enti pubblici.

CAUSA  EFFETTO

I costi sono sensibilmente diminuiti.

Inoltre l’aumento della velocità di interconnessione delle reti LAN ha fatto sì che diventasse ragionevole anche pensare a PC connessi in rete locale come a macchine per il calcolo parallelo.

Infine si sta cominciando a pensare di utilizzare macchine collegate in reti WAN come server di calcolo distribuito (grid).

L’hardware c’è, o almeno siamo sulla buona strada.

Lo standard MPI

La ricerca di algoritmi paralleli è attualmente un settore molto fertile.
Il parallelismo di un codice può nascere:

- dalla fisica (processi indipendenti),
- dalla matematica (set indipendenti di operazioni matematiche),
- dalla fantasia del programmatore.

Se hardware e algoritmi ci sono, perché i programmi paralleli sono poco diffusi ?

L'ostacolo principale alla diffusione di codici paralleli è (stata) la loro difficoltà di sviluppo. Il collo di bottiglia è il software: è spesso faticoso e laborioso (quindi costoso) sviluppare un codice parallelo e se poi questo non è portabile potrebbe non valerne la pena.

Lo standard MPI è nato da questa esigenza di portabilità e semplicità.

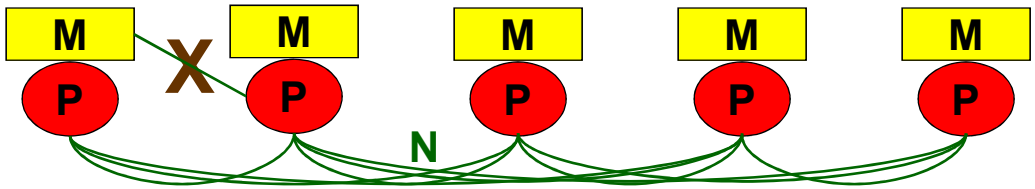
Si tratta di librerie oggi molto diffuse che si presentano al programmatore di un linguaggio ad alto livello (FORTRAN, FORTRAN90, C, C++) con un set di chiamate standard, indipendenti dalla specifica implementazione e dall'hardware su cui girano.

Lo standard MPI

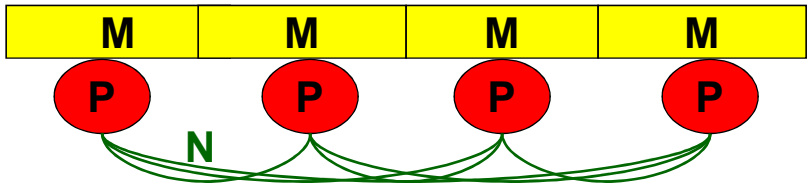
Il modello computazionale che realizzano è quello MESSAGE PASSING.
Altre possibilità sono quelli DATA PARALLEL (HPF, OpenMP)
e SHARED MEMORY (OpenMP, ShMem).

Attenzione a non confondere il modello computazionale (o di programmazione) con con quello architetturale che descrive la macchina (SIMD, MIMD UMA, ecc…).

Nel modello MESSAGE PASSING si suppone che ogni processo abbia una memoria locale (anche se fisicamente può non essere così) e che non possa accedere direttamente alla memoria degli altri processi.



Questo è vero anche se fisicamente la macchina è SMP



Lo standard MPI

I vantaggi del modello MESSAGE PASSING sono:

Universalità	E' naturale usarlo in ambienti eterogenei, in cui i PE sono diversi,
Facilità di debugging	Gli errori più comuni in un programma parallelo consistono nella sovrascrittura involontaria della memoria. Il modello MP costringe a esplicitare ogni processo di comunicazione da ambo le parti.
Performance	Agevola lo sfruttamento della struttura multi-livello della memoria evitando (o riducendo) problemi quali la coerenza di cache.

L'MPI Forum dal 1992 ha lavorato per standardizzare le librerie MPI.
La pagina di riferimento è http://www-unix.mcs.anl.gov/mpi/ (vedere sito del corso)

L'MPI è uno standard. Noi useremo una specifica implementazione: le librerie MPICH.
http://www-unix.mcs.anl.gov/mpi/mpich

MPI: i concetti di base

- Il codice sorgente che scriviamo è uno solo.
- L'eseguibile è comune a tutti i processi e sarà caricato da uno speciale loader su ogni PE.
- Ogni processo conosce (tramite una chiamata di libreria) il numero del PE su cui sta girando.
- Il programmatore prevede esplicitamente (nel codice ad es. con "IF") il branching in modo che ogni PE esegua la propria parte di codice.
- Ogni volta che un processo deve scambiare dati con un altro occorre introdurre istruzioni di *send* e *receive* (o di broadcast, riduzione, ecc…) esplicite in entrambi.
- E' possibile anche includere punti di sincronizzazione in modo che i processi si "aspettino" ad una data istruzione e ripartano insieme.

Programmeremo quindi usando un modello **SPMD** (Single-Program Multiple Data) su una architettura **MIMD**

MPI: i concetti di base

Prima di usare una chiamata (subroutine o funzione) ad una libreria MPI occorre inizializzare le librerie con

MPI_Init

Quando non servono più (o alla fine del programma) occorre dichiarare il termine del loro uso con

MPI_Finalize

Ecco un programma banale che non usa le MPI ma le inizializza e termina..

```
#include <stdio.h>
#include "mpi.h"
int main( int argc, char *argv[] )
{
    MPI_Init( &argc, &argv );
    printf( "Hello world \n" );
    MPI_Finalize();
    return 0;
}
```

MPI: i concetti di base

```
#include <stdio.h>
#include "mpi.h"
int main( int argc, char *argv[] )
{
    MPI_Init( &argc, &argv );
    printf( "Hello world \n" );
    MPI_Finalize();
    return 0;
}
```

- occorre includere *mpi.h*
- le chiamate sono funzioni
- l'error-code è il valore della funzione
- gli arg di MPI_Init sono gli indirizzi degli arg del main
- vari argomenti sono tipi specifici (MPI_Comm, MPI_Datatype,...)

I nomi delle chiamate e gli argomenti sono gli stessi (a parte poche eccezioni).

MPI: i concetti di base

Il modulo (.mod) o l'header (.h) MPI contengono la dichiarazione e definizione di varie costanti indispensabili per l'uso delle MPI. La lista completa è parte dello standard

http://www.mpi-forum.org/docs/mpi-11-html/node169.html#Node169

Ad esempio le costanti che indicano gli error-code sono:

MPI_SUCCESS	MPI_ERR_REQUEST	MPI_ERR_UNKNOWN
MPI_ERR_BUFFER	MPI_ERR_ROOT	MPI_ERR_TRUNCATE
MPI_ERR_COUNT	MPI_ERR_GROUP	MPI_ERR_OTHER
MPI_ERR_TYPE	MPI_ERR_OP	MPI_ERR_INTERN
MPI_ERR_TAG	MPI_ERR_TOPOLOGY	MPI_PENDING
MPI_ERR_COMM	MPI_ERR_DIMS	MPI_ERR_IN_STATUS
MPI_ERR_RANK	MPI_ERR_ARG	MPI_ERR_LASTCODE

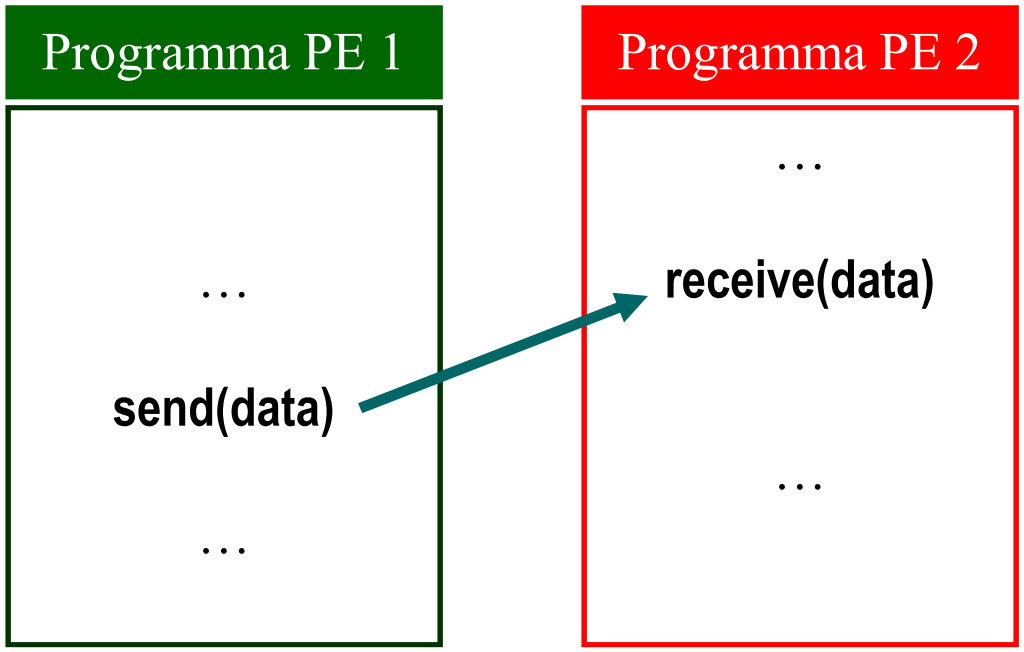
Se in valore della funzione è

$\neq$ MPI_SUCESS

si è verificato un errore nella routine di libreria (e il programma di default abortisce).

MPI: i concetti di base

Per scambiare dati occorre quindi la partecipazioni di entrambi i processi.



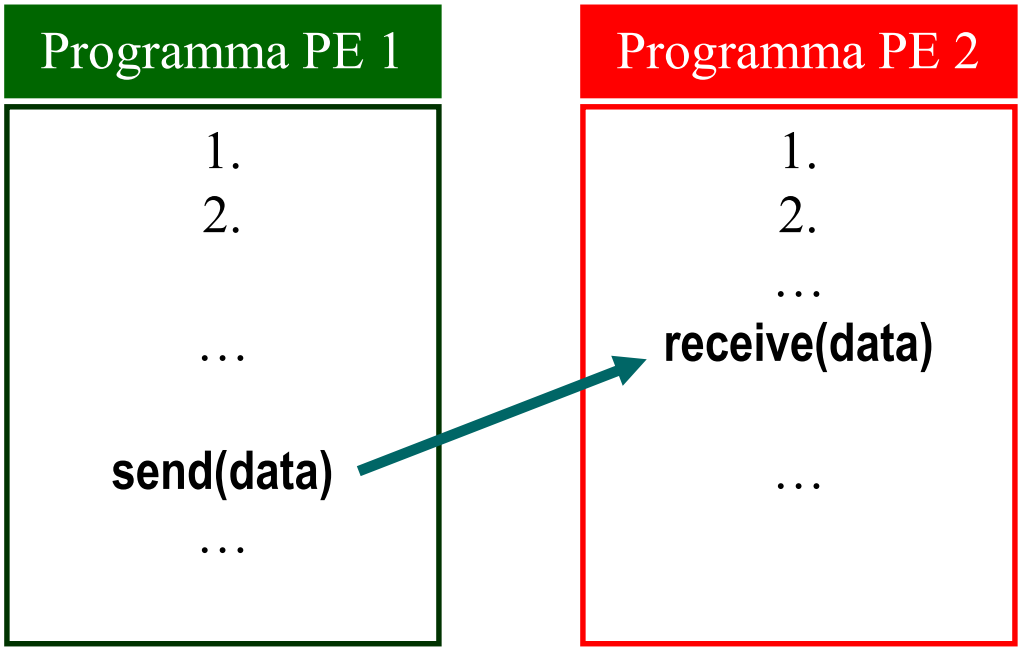
La possibilità di scambiare dati senza la partecipazione esplicita di entrambi (PUT e GET) è parte delle specifiche MPI-2 che noi non trattiamo.

In generale però occorreranno più informazioni alle chiamate *put* e *get*, almeno:

- il PE di destinazione per il *send* e quello di origine per il *get*;
- una etichetta (tag) per identificare il messaggio;
- una descrizione dei dati in transito.

MPI: i concetti di base

Ma prima di cominciare a scambiarsi i dati i programmi devono conoscere l'environment



1. Quanti processi ci sono ?
2. Chi sono io ?

1. MPI_Comm_size
2. MPI_Comm_rank

1. int MPI_Comm_size (MPI_Comm comm, int *numpe)
2. int MPI_Comm_rank (MPI_Comm comm, int *mynumpe)

MPI: i concetti di base

Possiamo dare un senso all'uso delle MPI nei nostri due programmi helloworld.

```
#include <stdio.h>
#include "mpi.h"
int main( int argc, char *argv[] )
{
  int rank, size;
  MPI_Init( &argc, &argv );
  MPI_Comm_size( MPI_COMM_WORLD, &size );
  MPI_Comm_rank( MPI_COMM_WORLD, &rank );
  printf( "Hello world from process %d of %d\n", rank, size );
  MPI_Finalize();
  return 0;
}
```

Se come argomento che indica il "communicator" (qui il primo argomento) indichiamo

MPI_COMM_WORLD

vogliamo fare un'operazione che tiene conto di TUTTI i processi. Vedremo altri casi.

MPI: i concetti di base

Adesso che sappiamo inizializzare/terminare le MPI e identificare i processi, facciamoli dialogare.

Abbiamo detto che in teoria serve almeno:

- il PE di destinazione per il *send* e quello di origine per il *get*;
- una etichetta (tag) per identificare il messaggio;
- una descrizione dei dati in transito.

MPI_Send(buffaddr, count, datatype, destinationpe, tag, comm)

MPI_Recv(buffaddr, maxcount, datatype, sourcepe, tag, comm, status)

Queste due istruzioni sono **BLOCKING**. Ovvero l'esecuzione continua solo quando l'operazione indicata (di send o receive, non entrambe) è andata a buon fine.

MPI: i concetti di base

MPI_Send(buffaddr, count, datatype, destinationpe, tag, comm)

argomenti:

- buffaddr puntatore all'indirizzo iniziale del buffer da spedire/ricevere. Di solito sarà il nome della variabile o array da mandare o ricevere. (Attenzione: in FORTRAN gli argomenti sono passati sempre per indirizzo, non per contenuto).
- count numero di elementi di tipo datatype che costituiscono il buffer. Se vogliamo ad esempio mandare una singola variabile, sarà =1; se vogliamo mandare un array sarà = dimensione array.
- datatype indica il tipo di dato che stiamo trasmettendo/ricevendo. Può essere un tipo predefinito (MPI_INT, MPI_DOUBLE_PRECISION), una struttura di tipi o un tipo definito da noi.
- destinationpe indica il numero del processo a cui il buffer viene inviato.
- tag permette di mettere un etichetta all'invio in modo da identificarlo quando lo si riceve.
- comm indica il communicator (context+group) nel cui ambito viene effettuata l'operazione.

MPI: i concetti di base

MPI_Recv(buffaddr, maxcount, datatype, sourcepe, tag, comm, status)

argomenti:

- buffaddr puntatore all'indirizzo iniziale del buffer da spedire/ricevere. Di solito sarà il nome della variabile o array da mandare o ricevere.
- maxcount numero di elementi di tipo datatype che costituiscono il buffer di ricezione, ovvero massimo numero di elementi che posso ricevere.
- datatype indica il tipo di dato che stiamo trasmettendo/ricevendo.
- sourcepe indica il numero del processo da cui ricevere il contenuto del buffer.
- tag permette di mettere un etichetta alla ricezione in modo da "accoppiarlo" con l'invio corrispondente.
- comm indica il communicator (context+group) nel cui ambito viene effettuata l'operazione.
- status contiene info sull'effettiva lunghezza del messaggio ricevuto, sull'identità effettiva del mittente e sul tag effettivo (utili in receive generici). In FORTRAN è un array di MPI_STATUS_SIZE interi.

MPI: i concetti di base

- MPI_INIT
- MPI_FINALIZE
- MPI_COMM_SIZE
- MPI_COMM_RANK
- MPI_SEND
- MPI_RECV

Con le sei routine introdotte è già possibile scrivere qualunque programma parallelo. Sono però indispensabili altri tipi di chiamate per incrementare efficienza, leggibilità, flessibilità di un codice parallelo.

Consideriamo adesso alcuni semplici esempi!

Sorgenti: **hello_world_mpi.c, Hello_world_mpi2.c, ping_pong_mpi.c**