W • SENSE
INTEGRATED CABLELESS SOLUTIONS

SAPIENZA
UNIVERSITÀ DI ROMA

Programmazione di sistemi multicore MPI #2

Fabrizio Gattuso

Punto a punto

La comunicazione punto a punto è la base di tutte le altre tipologie e si basa sulle funzioni:

MPI_Send(void* **data**, int **count**, MPI_Datatype datatype, int **destination**, int tag, MPI_Comm **communicator**)

MPI_Recv(void* **data**, int **count**, MPI_Datatype datatype, int **source**, int tag, MPI_Comm **communicator**, MPI_Status* **status**)

*Se la sorgente è impostata a **MPI_ANY_SOURCE** verranno ricevuti tutti i messaggi nel **communicator**.*

Punto a punto (2)

MPI datatype	C equivalent
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_LONG_LONG	long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	char

Esercizio MPI

Scrivere un programma che sfrutti i core del proprio laptop per:

1) Creare un **vettore di M interi di N dimensioni** (dove N è il vostro comm_size)
2) **Ogni core calcola la somma parziale** della dimensione n-esima corrispondente al numero di core
3) **Un core** si occupare di calcolare la **somma totale** e di stamparla

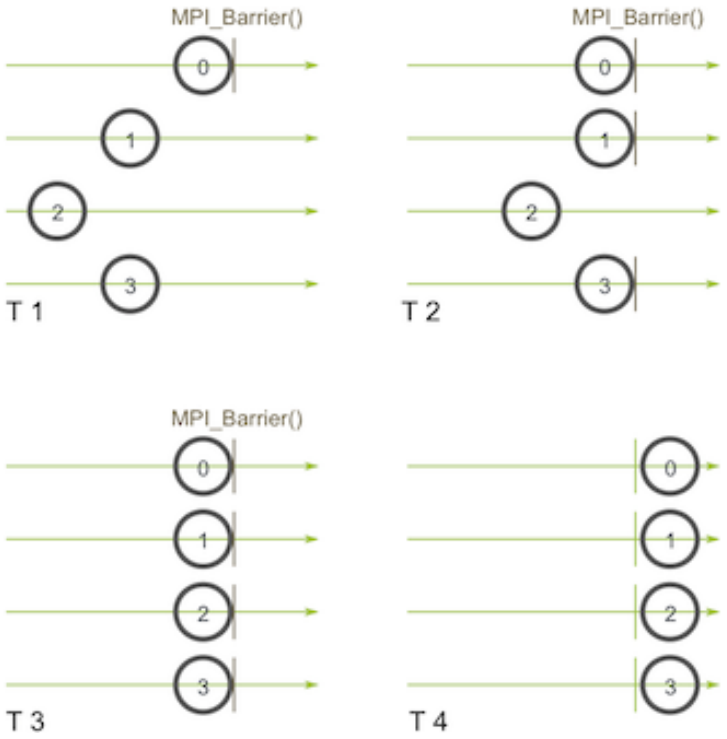
Comunicazione collettiva

Come già sapete comunicazione collettiva implica **sincronizzazione**.

In MPI esiste una primitiva che simula una "barriera" che i processi possono superare solamente tutti insieme.

MPI_Barrier(MPI_Comm **communicator**)

Comunicazione collettiva (2)

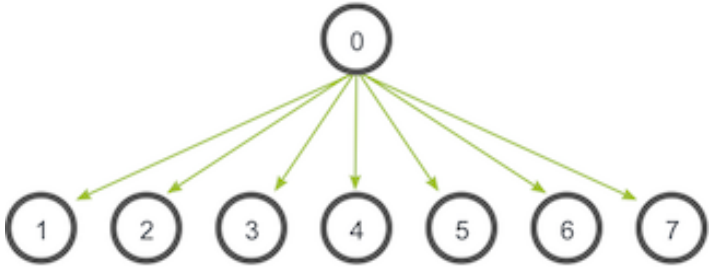


Una cosa importante da ricordare è che **TUTTI I PROCESSI DEVONO CHIAMARE LA BARRIER.**

Broadcast

Il concetto di broadcast è molto semplice:

un processo/core invia gli stessi dati a tutti i core rimanenti



MPI_Bcast(
void* **data**,
int **count**,
MPI_Datatype **datatype**,
int **root**,
MPI_Comm **communicator**)

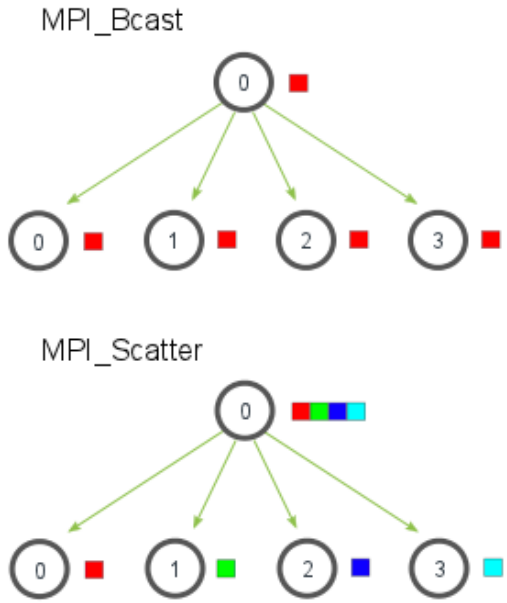
Broadcast (2)

Avete idee su come è possibile effettuare una comunicazione broadcast senza utilizzare la funzione **MPI_Bcast**?

```
for (i = 0; i < world_size; i++) {
        if (i != world_rank) {
            MPI_Send(data, count, datatype, i, 0, communicator);
        }
}
```

Scattering

La differenza principale rispetto al broadcast è che invece di inviare a tutti gli stessi dati, **invia a ciascun core una sola porzione del dato da inviare**



MPI_Scatter(
void* **send_data**,
int **send_count**,
MPI_Datatype **send_datatype**,
void* **recv_data**,
int **recv_count**,
MPI_Datatype **recv_datatype**,
int **root**,
MPI_Comm **communicator**)

Scattering (2)

Alcune precisazioni:

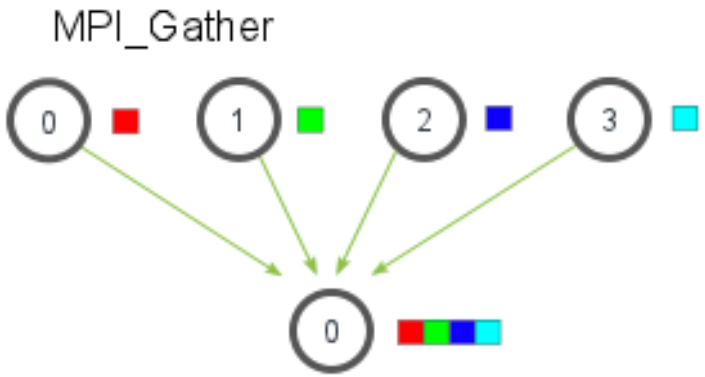
send_count indica il numero di elementi da inviare ad **OGNI** core

Questa funzione oltre a un buffer di invio ha anche un buffer di ricezione dove verranno salvati i dati parziali.

Gathering

Il gathering è l'operazione inversa rispetto allo scattering. Ovvero serve a raggruppare diversi dati in un unico core.

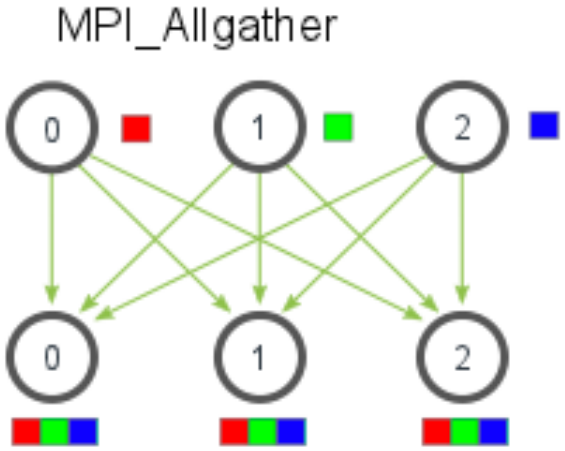
Di norma solamente il nodo root ha i campi receive impostati a valori diversi da NULL.



MPI_Gather(
void* **send_data**,
int **send_count**,
MPI_Datatype **send_datatype**,
void* **recv_data**,
int **recv_count**,
MPI_Datatype **recv_datatype**,
int **root**,
MPI_Comm **communicator**)

All gather

È una versione particolare della funzione Gather. Invece di inviare un dato a un singolo core lo invia a tutti i core. Sostanzialmente **è una MPI_Gather seguita da un MPI_Bcast**.

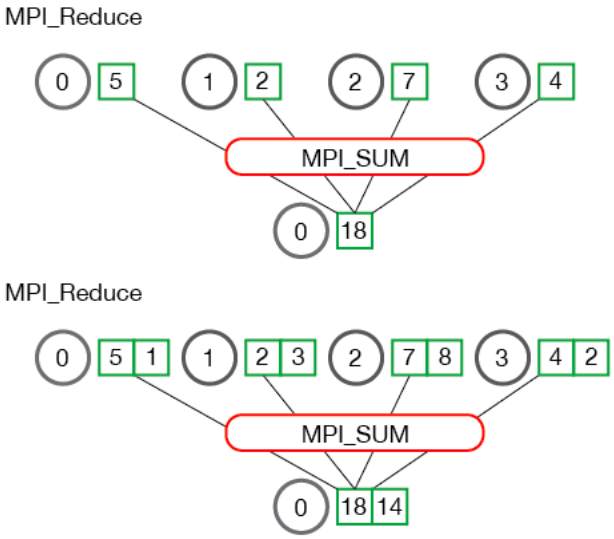


MPI_Allgather(
void* **send_data**,
int **send_count**,
MPI_Datatype **send_datatype**,
void* **recv_data**,
int **recv_count**,
MPI_Datatype **recv_datatype**,
MPI_Comm **communicator**)

Reduce

Reduce deriva dalla programmazione funzionale. È la trasformazione di un insieme di elementi in un singolo elemento.

[1, 2, 3, 4, 5] -> sum([1, 2, 3, 4, 5]) -> 15



MPI_Reduce(
void* **send_data**,
void* **recv_data**,
int **count**,
MPI_Datatype **datatype**,
MPI_Op **op**,
int **root**,
MPI_Comm **communicator**)

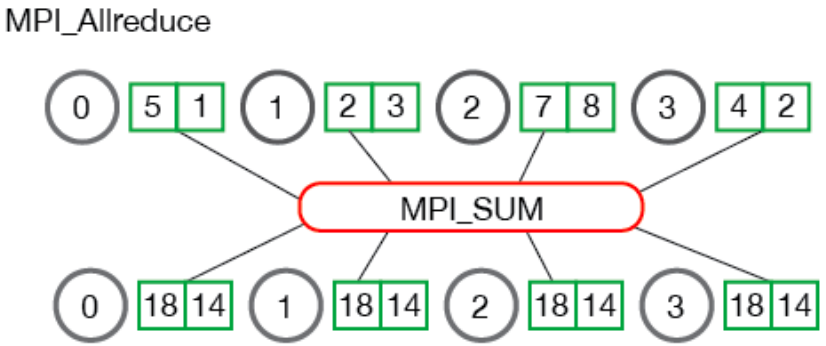
Reduce (2)

Macro	Operazione
MPI_MAX	Calcola il massimo
MPI_MIN	Calcola il minimo
MPI_SUM	Esegue la somma
MPI_PROD	Esegue il prodotto
MPI_LAND	AND logico
MPI_LOR	OR logico
MPI_BAND	Bitwise AND dei bits degli elementi
MPI_BOR	Bitwise OR dei bits degli elementi
MPI_MAXLOC	Calcola il massimo e ritorna il rank del processo
MPI_MINLOC	Calcola il minimo e ritorna il rank del processo

È possibile definire una propria operazione custom

All Reduce

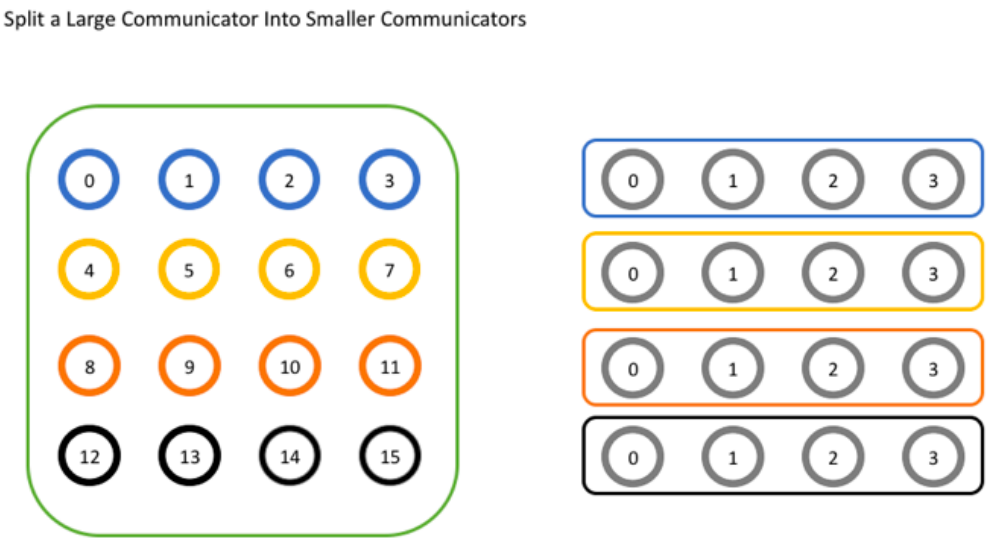
Dopo aver svolto l'operazione di reduce invia il risultato in broadcast.



MPI_Allreduce(
void* **send_data**,
void* **recv_data**,
int **count**,
MPI_Datatype **datatype**,
MPI_Op **op**,
MPI_Comm **communicator**)

Gruppi

In tutti gli esempi visti abbiamo utilizzato come **COMMUNICATOR** la costante **MPI_COMM_WORLD**.



Gruppi (2)

MPI_Comm_split(
MPI_Comm **comm**,
int **color**,
int **key**,
MPI_Comm* **newcomm**)

MPI_Comm row_comm;
MPI_Comm_split(MPI_COMM_WORLD, world_rank / 4, world_rank, &row_comm);