Corso di High Performance Computing

Esercitazione MPI del 6/11/2018

Moreno Marzolla

Ultimo aggiornamento: 2018-11-02

Per svolgere l'esercitazione è possibile collegarsi al server `isi-raptor03.csr.unibo.it` tramite ssh, usando come *username* il proprio indirizzo mail istituzionale completo, e come password la propria password istituzionale (cioè quella usata per accedere alla casella di posta o ad AlmaEsami). Sulla macchina è installato il compilatore `gcc` e alcuni editor di testo per console: `vim`, `pico`, `joe`, `ne` e `emacs`. Per chi non è pratico suggerisco `pico`, che è semplice da usare e richiede poche risorse. Chi ha un portatile con Linux può lavorare localmente, dopo aver installato il compilatore.

Per scaricare l'archivio con i sorgenti di questa esercitazione è possibile usare i comandi:

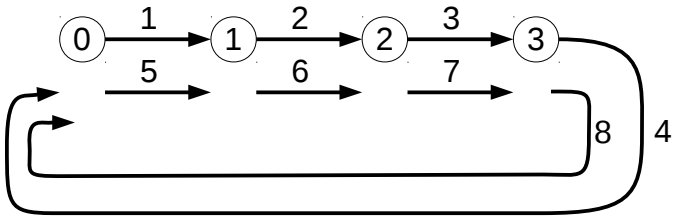
```
wget http://www.moreno.marzolla.name/teaching/HPC/ex1-mpi.zip
unzip ex1-mpi.zip
cd ex1-mpi/
```

1. Comunicazione ad anello

Realizzare un programma MPI chiamato `mpi-ring.c` che effettua una comunicazione ad anello tra i processi. Più in dettaglio, detto P il numero di processi MPI (da specificare con il comando `mpirun`; si deve avere $P > 1$) si richiede di realizzare le funzionalità seguenti:

- Il programma riceve sulla riga di comando un valore intero K, che rappresenta il numero di "giri" dell'anello che devono essere effettuati ($K \geq 1$). Ricordo che tutti i processi MPI hanno accesso ai valori passati sulla riga di comando, quindi tutti possono conoscere il valore di K senza bisogno di comunicazioni esplicite.
- Il processo 0 (il master) invia al processo 1 un intero, il cui valore iniziale è 1.
- Ciascun processo p (incluso il master) rimane in attesa di ricevere un valore v dal processo $p - 1$; una volta ricevuto, il processo p invia il valore $(v + 1)$ al processo $p + 1$. Poiché la comunicazione si considera ad anello, il predecessore del processo 0 è $(P - 1)$, mentre il successore del processo $(P - 1)$ è il processo 0.
- Il master stampa il valore ricevuto dopo la K-esima iterazione e la computazione termina; dati il numero P di processi e il valore di K, quale valore deve stampare il master?

Ad esempio, se $K = 2$ e ci sono $P = 4$ processi, la comunicazione da realizzare deve essere la seguente (i cerchi sono i processi MPI; le frecce rappresentano messaggi il cui contenuto è il numero indicato sopra). I messaggi percorrono $K = 2$ "giri" dell'anello composto da tutti i processi; al termine dell'esecuzione il processo 0 riceve il valore 8.



2. Broadcast tramite comunicazioni punto-punto

Scopo di questo esercizio è di implementare la funzione

```
void my_Bcast(int *v)
```

La funzione deve svolgere operazioni diverse a seconda che venga eseguita dal master (processo 0) o da un altro processo. Per fare ciò ogni processo determina il proprio rango p e il numero P di processi MPI attivi.

Il processo 0:

- invia il valore `*v` ai processi $(2p + 1)$ e $(2p + 2)$ (purché i destinatari esistano, cioè il loro rango sia $< P$)

Ogni altro processo $p > 0$:

- riceve un valore dal processo $(p – 1)/2$ e lo memorizza nella locazione di memoria `v`;
- invia il valore ricevuto ai processi $(2p + 1)$ e $(2p + 2)$ (purché i destinatari esistano).

Ad esempio, nel caso $P = 14$ si otterrebbe lo schema di comunicazione illustrato nella Figura 1; le frecce indicano comunicazioni punto-punto; i numeri indicano il rango dei processi coinvolti. Si noti che il procedimento descritto in precedenza funziona correttamente qualunque sia il numero P di processi.

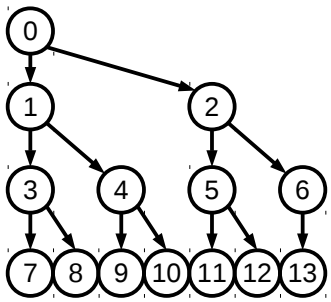


Figura 1: Schema d comunicazione con P = 14 processi

Vedremo più avanti che MPI mette a disposizione una funzione per realizzare una comunicazione di tipo *broadcast*, che va sempre preferita a soluzioni "fatte in casa" come quella descritta sopra. Scopo di questa esercitazione è di vedere come sia possibile realizzare una comunicazione broadcast usando esclusivamente i costrutti MPI che abbiamo visto fino a questo momento, ossia `MPI_Send()` e `MPI_Recv()`.

Il file `mpi-my-bcast.c` contiene lo scheletro della funzione `my_Bcast()` cui manca il corpo. Completare il corpo della funzione `my_Bcast()` usando `MPI_Send()` e `MPI_Recv()`.

3. Calcolo di Pi greco

Il file `mpi-pi.c` contiene una implementazione seriale di un algoritmo di tipo Monte Carlo per il calcolo del valore approssimato di π (pi greco). L'algoritmo è già stato descritto nella prima esercitazione OpenMP, e viene richiamato nuovamente per comodità (si faccia riferimento alla Figura 2).

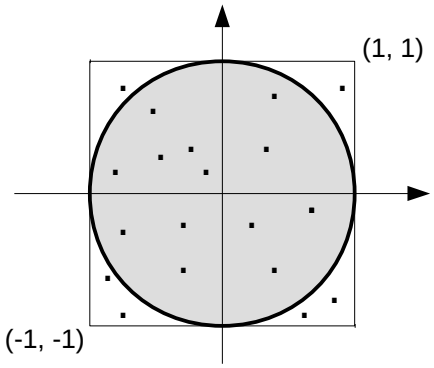


Figura 2: Calcolo del valore di π con metodo Monte Carlo

- Vengono generati *N* punti casuali all'interno del quadrato di vertici opposti (-1, -1) e (1, 1);
- Si definisce *x* il numero di punti che cadono all'interno del cerchio di raggio unitario e centro nell'origine degli assi;
- Il rapporto *x* / *N* approssima il rapporto tra l'area del quadrato (che vale 4) e l'area del cerchio inscritto in esso. Poiché sappiamo che l'area di tale cerchio è π, possiamo stimare π come (4*x* / *N*)

Modificare il file `mpi-pi.c` per parallelizzare il calcolo del valore di π. Sono possibili diverse strategie; si consiglia di usare quella seguente che ha il vantaggio di essere abbastanza semplice da realizzare (*P* rappresenta il numero di processi MPI attivi):

1. Ciascun processo ottiene il valore di *N* dalla riga di comando; si può inizialmente assumere che *N* sia multiplo di *P*, e successivamente rilassare questo requisito per fare funzionare il programma con qualsiasi valore di *N*.
2. Ciascun processo *p*, incluso il master, genera *N*/*P* punti casuali e tiene traccia del numero x_p di punti all'interno del cerchio;
3. Ciascun processo *p* diverso dal master invia al master il proprio valore x_p
4. Il master riceve i valori x_p (per ogni $p = 1, \ldots P - 1$; il valore x_0 già ce l'ha) e calcola la loro somma *x* stampando il valore approssimato di π come (4*x* / *N*).

Realizzare il passo 3 mediante operazioni send/receive punto-punto. Questo approccio non è efficiente: vedremo nelle prossime lezioni come il passo 4 possa (debba!) essere realizzato mediante una primitiva di comunicazione collettiva che realizza l'operazione di riduzione.

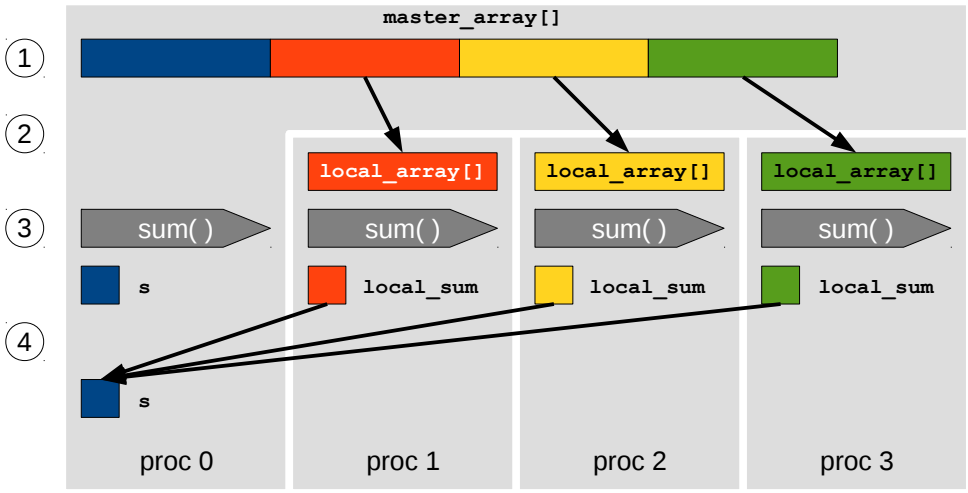
4. Somma degli elementi di un array

Il file `mpi-sum.c` contiene una implementazione essenzialmente seriale di un programma MPI che calcola la somma degli elementi di un array; nella versione fornita, il processo 0 esegue tutte le computazioni. Modificare il programma in modo da parallelizzare la somma, seguendo le seguenti indicazioni (si faccia riferimento alla figura)

1. Il processo master (quello con *rank* 0) crea e inizializza l'array `master_array[]` di cui calcolare la somma.
2. Detto *P* il numero di processi MPI, il master suddivide logicamente l'array `master_array[]` in *P* blocchi di dimensioni uniformi, e invia $P - 1$ blocchi (tutti tranne il primo) agli altri processi utilizzando la funzione `MPI_Send()`; i processi diversi dal master allocano un array locale di dimensioni appropriate e ricevono il proprio blocco usando

`MPI_Recv()`

3. Ciascun processo calcola la somma parziale della propria porzione di array; il master opera sul primo blocco dell'array globale.
4. Ciascun processo diverso dal master invia la propria somma locale al master, usando `MPI_Send()`; il master riceve le somme locali usando `MPI_Recv()` e le accumula nella somma globale `s`.



Vedremo nelle prossime lezioni come i passi 2) e 4) possano essere effettuati in modo più efficiente usando le operazioni di comunicazione collettiva messe a disposizione da MPI.