Lab 6b - MPI base

https://elly2023.smfi.unipr.it/mod/page/view.php?id=5298

OBIETTIVO
Studiare e analizzare mpirun e mpicc, i comandi principali per l'utilizzo del modello a memoria distribuita con MPI nei sistemi HPC.

Attività svolte

Come prima cosa ho creato la directory di lavoro "*mkdir -p ~/HPC2324/mpi/base/*" e copiato all'interno gli esercizi tramite "*cp /hpc/home/roberto.alfieri/SHARE/mpi/base/* .*"

La directory contiene alcuni esercizi dimostrativi dei principali schemi di comunicazione MPI:

- mpi1_helloworld.c
  Il programma stampa per ogni processo un messaggio di saluto che include il proprio ID (taskid) e il nome del processore (hostname); nel caso il processo è il master (ID 0) allora stampa anche il numero totale di processi.

```
[martina.genovese@ui01 base]$ mpirun mpi1_helloworld_intel
Hello from task 0 on ui01.hpc.unipr.it!
MASTER: Number of MPI tasks is: 4
Hello from task 2 on ui01.hpc.unipr.it!
Hello from task 3 on ui01.hpc.unipr.it!
Hello from task 1 on ui01.hpc.unipr.it!
```

- mpi2_send-recv.c
  Il programma crea una comunicazione punto-punto tra due processi: il processo con rank 0 invia un intero al processo con rank 1, il quale lo riceve e lo stampa.

```
[martina.genovese@ui01 base]$ mpirun mpi2_send-recv_intel
Process 1 receives 10 from process 0.
```

- mpi3_send_recv-array.c
  Il programma crea una comunicazione punto-punto tra due processi: il processo con rank 0 invia un array di float al processo con rank 1, il quale lo riceve e lo stampa.

```
[martina.genovese@ui01 base]$ mpirun mpi3_send_recv-array_intel
Process 1 receives the following array from process 0.
  0.00
  1.00
  2.00
  3.00
  4.00
  5.00
  6.00
  7.00
  8.00
  9.00
```

- mpi4_sendrecv.c
  Il programma crea una comunicazione circolare tra processi: ogni processo invia il proprio rank al processo successivo e riceve il rank del processo precedente. Ad esempio, se ci sono quattro processi con rank 0, 1, 2 e 4 il processo 0 invia 0 a processo 1 e riceve 2 (da processo 2), il processo 1 invia 1 a processo 2 e riceve 0 (da processo 0), e così via.

```
[martina.genovese@ui01 base]$ mpirun mpi4_sendrecv_intel
Process 0/4 receives 3 from process 3.
Process 1/4 receives 0 from process 0.
Process 2/4 receives 1 from process 1.
Process 3/4 receives 2 from process 2.
```

- mpi5_pingpong_time.c
  Il programma implementa un meccanismo di tipo "ping-pong" tra due processi. Il processo A invia un messaggio a B, che poi risponde, creando un ciclo continuo fino a un totale di 50 messaggi.
  Alla chiusura delle comunicazioni, il processo A calcola quanto tempo ha impiegato a completare

l'invio e la ricezione di messaggi e lo stampa, restituendo il tempo medio per ogni scambio di messaggio.

```
[martina.genovese@ui01 base]$ mpirun mpi5_pingpong_time_intel
Time for one message: 0.000022 seconds.
```

- mpi6_ring.c

  Il programma esegue una somma tra tutti i processi. Ogni processo invia un valore (il suo rank iniziale) al processo alla sua destra e riceve un valore dal processo alla sua sinistra. Attraverso size (numero totale di processi) iterazioni, ogni processo accumula i valori ricevuti e li somma.
  Al termine, ogni processo mostra il proprio ID e la somma totale.

```
[martina.genovese@ui01 base]$ mpirun mpi6_ring_intel
PE0:    Sum = 6
PE1:    Sum = 6
PE2:    Sum = 6
PE3:    Sum = 6
```

- mpi7_bcast.c

  Il programma utilizza MPI_Bcast per inviare un valore (buf) in broadcast da un processo root a tutti gli altri processi.

```
[martina.genovese@ui01 base]$ mpirun mpi7_bcast_intel
[1]: Before Bcast, buf is 32767
[2]: Before Bcast, buf is 32765
[3]: Before Bcast, buf is 32767
[0]: Before Bcast, buf is 777
[0]: After Bcast, buf is 777
[1]: After Bcast, buf is 777
[2]: After Bcast, buf is 777
[3]: After Bcast, buf is 777
```

- mpi8_scatter_gather.c

  Il programma distribuisce un vettore di interi tra più processi paralleli, questi lo modificano localmente e infine il vettore viene riassemblato e stampato.
  In particolare:
  - La funzione MPI_Scatter divide il Vector in pezzi uguali e distribuisce a ciascun processo LSIZE elementi. Ogni processo riceve una parte del Vector nel suo localVector
  - All'interno del ciclo "for (i=0; i < LSIZE ; i++) *(localVector+i)+=1;" ogni task incrementa gli interi ricevuti
  - La funzione MPI_Gather raccoglie i dati modificati dai vari processi e li mette di nuovo nel Vector globale, ma solo il processo 0 riceverà i valori finali

  Al termine il processo 0 stampa il contenuto aggiornato del Vector.

```
[martina.genovese@ui01 base]$ mpirun mpi8_scatter_gather_intel
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
 36 37 38 39 40
```

- mpi9_reduction.c

  Nel programma tutti gli N task inviano il proprio rank con MPI_Send() al task 0.
  Il task 0 itera N volte MPI_recv() per ricevere tutti messaggi, sommarli e stampare il risultato.

```
[martina.genovese@ui01 base]$ mpirun mpi9_reduction_intel
mpi_size: 4, reduction: 6
```

- mpi_DataMining.c

  Nel programma tutti i processi definiscono lo stesso intero x (ad esempio x=7) e generano ogni secondo un intero random tra 0 e 100.
  Chi trova il numero x invia un messaggio agli altri con il numero trovato e termina.
  Tutti gli altri processi ricevono e stampano il valore ricevuto e l'ID del processo mittente, quindi terminano.

```
[martina.genovese@ui01 base]$ mpirun mpi_DataMining_intel
Task 1/4 MPI_Irecv started
Task 1/4 - x:7  r:55
Task 2/4 MPI_Irecv started
Task 2/4 - x:7  r:94
Task 3/4 MPI_Irecv started
Task 3/4 - x:7  r:33
Task 0/4 MPI_Irecv started
Task 0/4 - x:7  r:17
Task 2/4 - x:7  r:57
Task 3/4 - x:7  r:49
Task 0/4 - x:7  r:74
Task 1/4 - x:7  r:66
Task 2/4 - x:7  r:3
Task 3/4 - x:7  r:0
Task 0/4 - x:7  r:9
Task 1/4 - x:7  r:6
Task 1/4 - x:7  r:48
Task 2/4 - x:7  r:9
Task 3/4 - x:7  r:70
Task 0/4 - x:7  r:87
Task 1/4 - x:7  r:80
Task 2/4 - x:7  r:3
Task 3/4 - x:7  r:25
Task 0/4 - x:7  r:57
Task 1/4 - x:7  r:90
Task 2/4 - x:7  r:1
Task 3/4 - x:7  r:13
Task 0/4 - x:7  r:78
Task 2/4 - x:7  r:20
Task 3/4 - x:7  r:45
Task 1/4 - x:7  r:94
Task 0/4 - x:7  r:69
Task 2/4 - x:7  r:18
Task 1/4 - x:7  r:77
Task 3/4 - x:7  r:59
Task 0/4 - x:7  r:36
Task 1/4 - x:7  r:98
Task 2/4 - x:7  r:9
Task 3/4 - x:7  r:20
Task 0/4 - x:7  r:87
Task 1/4 - x:7  r:45
```

```
Task 2/4 - x:7  r:17
Task 3/4 - x:7  r:88
Task 0/4 - x:7  r:74
Task 1/4 - x:7  r:9
Task 2/4 - x:7  r:74
Task 3/4 - x:7  r:38
Task 0/4 - x:7  r:44
Task 2/4 - x:7  r:52
Task 3/4 - x:7  r:10
Task 1/4 - x:7  r:93
Task 0/4 - x:7  r:35
Task 2/4 - x:7  r:35
Task 1/4 - x:7  r:54
Task 3/4 - x:7  r:16
Task 0/4 - x:7  r:73
Task 2/4 - x:7  r:0
Task 1/4 - x:7  r:63
Task 3/4 - x:7  r:37
Task 0/4 - x:7  r:26
Task 2/4 - x:7  r:70
Task 1/4 - x:7  r:54
Task 3/4 - x:7  r:85
Task 0/4 - x:7  r:39
Task 1/4 - x:7  r:55
Task 2/4 - x:7  r:34
Task 3/4 - x:7  r:12
Task 0/4 - x:7  r:77
Task 2/4 - x:7  r:83
Task 1/4 - x:7  r:33
Task 3/4 - x:7  r:32
Task 0/4 - x:7  r:84
Task 2/4 - x:7  r:62
Task 1/4 - x:7  r:60
Task 3/4 - x:7  r:65
Task 0/4 - x:7  r:57
Task 2/4 - x:7  r:22
Task 1/4 - x:7  r:97
Task 3/4 - x:7  r:48
Task 0/4 - x:7  r:71
Task 1/4 - x:7  r:59
Task 2/4 - x:7  r:10
Task 3/4 - x:7  r:61
```

```
Task 0/4 - x:7  r:8
Task 1/4 - x:7  r:11
Task 2/4 - x:7  r:76
Task 3/4 - x:7  r:41
Task 0/4 - x:7  r:45
Task 1/4 - x:7  r:20
Task 2/4 - x:7  r:30
Task 3/4 - x:7  r:40
Task 0/4 - x:7  r:10
Task 1/4 - x:7  r:16
Task 2/4 - x:7  r:79
Task 3/4 - x:7  r:41
Task 0/4 - x:7  r:54
Task 1/4 - x:7  r:55
Task 2/4 - x:7  r:71
Task 3/4 - x:7  r:87
Task 0/4 - x:7  r:39
Task 2/4 - x:7  r:76
Task 1/4 - x:7  r:16
Task 3/4 - x:7  r:36
Task 0/4 - x:7  r:56
Task 1/4 - x:7  r:4
Task 2/4 - x:7  r:12
Task 3/4 - x:7  r:20
Task 0/4 - x:7  r:95
Task 2/4 - x:7  r:85
Task 1/4 - x:7  r:86
Task 3/4 - x:7  r:85
Task 0/4 - x:7  r:86
Task 2/4 - x:7  r:3
Task 3/4 - x:7  r:97
Task 1/4 - x:7  r:9
Task 0/4 - x:7  r:16
Task 1/4 - x:7  r:30
Task 2/4 - x:7  r:32
Task 3/4 - x:7  r:34
Task 0/4 - x:7  r:27
Task 2/4 - x:7  r:70
Task 3/4 - x:7  r:42
Task 1/4 - x:7  r:98
Task 0/4 - x:7  r:26
Task 2/4 - x:7  r:59
```

```
Task 3/4 - x:7  r:93
Task 1/4 - x:7  r:26
Task 0/4 - x:7  r:92
Task 2/4 - x:7  r:27
Task 3/4 - x:7  r:68
Task 1/4 - x:7  r:85
Task 0/4 - x:7  r:43
Task 2/4 - x:7  r:29
Task 3/4 - x:7  r:54
Task 1/4 - x:7  r:4
Task 0/4 - x:7  r:78
Task 2/4 - x:7  r:50
Task 3/4 - x:7  r:68
Task 1/4 - x:7  r:31
Task 0/4 - x:7  r:12
Task 2/4 - x:7  r:69
Task 3/4 - x:7  r:99
Task 1/4 - x:7  r:38
Task 0/4 - x:7  r:8
Task 2/4 - x:7  r:21
Task 3/4 - x:7  r:85
Task 1/4 - x:7  r:57
Task 0/4 - x:7  r:94
Task 1/4 - x:7  r:70
Task 2/4 - x:7  r:38
Task 3/4 - x:7  r:7
Task 0/4 - RECEIVED 7 from 3, exiting ...
Task 1/4 - RECEIVED 7 from 3, exiting ...
Task 2/4 - RECEIVED 7 from 3, exiting ...
Task 3/4 - RECEIVED 7 from 3, exiting ...
```

Altri esercizi

- mpi_DataMining_bcast.c
  Nel programma il Task 0 invia a tutti un intero x (ad esempio x=7), mentre tutti gli altri processi generano ogni secondo un intero random tra 0 e 100.

Chi trova il numero x per primo invia un messaggio agli altri con il numero trovato e termina. Tutti gli altri processi ricevono e stampano il valore ricevuto e l'ID del processo mittente, quindi terminano.

In particolare ho fatto le seguenti modifiche al codice:

```
35    // solo il Task 0 (root) definisce l'intero x=0
36    if (MPIrank == root) {
37       x=7;
38    }
39    // il task 0 invia l'inero x a tutti gli altri tramite MPI_Bcast
40    MPI_Bcast(&x, 1, MPI_INT, root, MPI_COMM_WORLD);
```

Output

```
[martina.genovese@ui01 base]$ module load intel impi
[martina.genovese@ui01 base]$ mpicc mpi_DataMining_bcast.c -o mpi_DataMining_bcast
[martina.genovese@ui01 base]$ mpirun mpi_DataMining_bcast
Task 0/4 MPI_Irecv started
Task 0/4 - x:7  r:17
Task 2/4 MPI_Irecv started
Task 2/4 - x:7  r:97
Task 3/4 MPI_Irecv started
Task 3/4 - x:7  r:37
Task 1/4 MPI_Irecv started
Task 1/4 - x:7  r:57
Task 0/4 - x:7  r:74
Task 3/4 - x:7  r:62
```

…

```
Task 3/4 - x:7  r:7
Task 1/4 - RECEIVED 7 from 3, exiting ...
Task 0/4 - RECEIVED 7 from 3, exiting ...
Task 2/4 - RECEIVED 7 from 3, exiting ...
Task 3/4 - RECEIVED 7 from 3, exiting ...
```

- mpi_token.c

mpi4_sendrecv.c

Nel programma tutti i task creano un token vuoto (int token=0) quindi il task 0 crea un token (token=1) che viene fatto circolare tra gli atri task fino a che non ritorna al task 0. Il task che riceve il token stampa il proprio rank.

Codice

```
15  int main(int argc, char **argv)
16  {
17    MPI_Status status;
18    int rank, size;
19    int prev_rank, next_rank;
20    int token = 0;
21
22    int data_send, data_recv;  /* data to communicate */
23
24    MPI_Init(&argc, &argv);
25    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
26    MPI_Comm_size(MPI_COMM_WORLD, &size);
27
28    if (rank == 0) {
29       token=1;
30    }
31
32    prev_rank = (rank-1+size) % size;
33    next_rank = (rank+1)      % size;
34
35    data_send = token;
36
37    MPI_Sendrecv(&data_send, 1, MPI_INT, next_rank, 666,
38                 &data_recv, 1, MPI_INT, prev_rank, 666, MPI_COMM_WORLD, &status);
39
40    printf("Process %d/%d receives token %d from process %d.\n", rank, size, token, status.MPI_SOURCE);
41
42    MPI_Finalize();
43    return 0;
44  }
```

Output

```
[martina.genovese@ui01 base]$ mpicc mpi_token.c -o mpi_token
[martina.genovese@ui01 base]$ mpirun mpi_token
Process 0/4 receives token 1 from process 3.
Process 1/4 receives token 0 from process 0.
Process 2/4 receives token 0 from process 1.
Process 3/4 receives token 0 from process 2.
```