# Corso di High Performance Computing

## Esercitazione MPI del 21/4/2017

Moreno Marzolla

*Ultimo aggiornamento: 2017/04/21*

Per svolgere l'esercitazione è possibile collegarsi al server `disi-hpc.csr.unibo.it` tramite ssh, usando come *username* il proprio indirizzo mail istituzionale completo, e come password la propria password istituzionale (cioè quella usare 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 surgenti di questa esercitazione è possibile usare i comandi:

```
wget http://www.moreno.marzolla.name/teaching/HPC/ex2-mpi.zip
unzip ex2-mpi.zip
cd ex2-mpi/
```

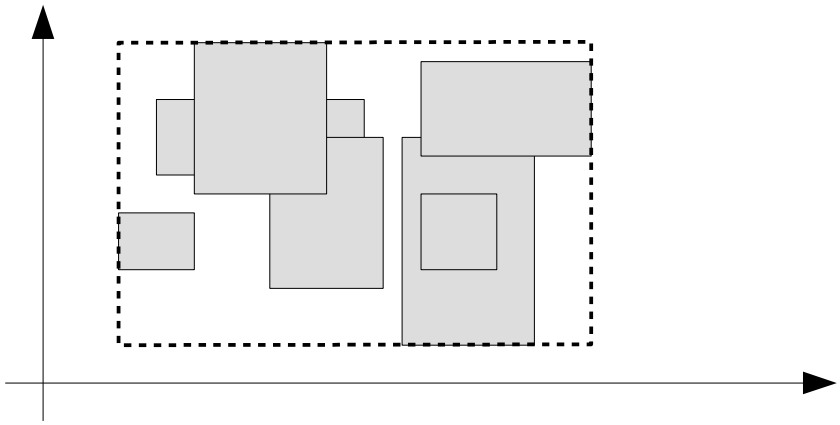
Alcuni degli esercizi producono immagini in formato PPM (*Portable Pixmap*) che le macchine Windows dei laboratori non sono in grado di visualizzare. È necessario convertire tali immagini in un formato diverso (ad esempio, PNG) dando sul server il comando:

```
convert image.ppm image.png
```

per poi copiare il file `image.png` sul proprio PC usando il programma Winscp (già installato).

## 1. Calcolo del bounding box di un insieme di rettangoli

Scopo di questo esercizio è il calcolo del *bounding box* di un insieme di rettangoli. Il bounding box è il rettangolo di area minima che contiene tutti i rettangoli dati; un esempio è mostrato nella figura seguente (il bounding box è tratteggiato)



Le coordinate dei rettangoli sono indicate in un file di testo, con il formato seguente. La prima riga contiene il numero $N$ di rettangoli; seguono $N$ righe, ciascuna composta da quattro valori `x1[`*i*`] y1[`*i*`] x2[`*i*`] y2[`*i*`]` di tipo `float`, separati da spazi. Le righe rappresentano le coordinate degli

angoli opposti di ciascun rettangolo: (`x1[i]`, `y1[i]`) sono le coordinate dell'angolo in alto a sinistra dell'*i*-esimo rettangolo, mentre (`x2[i]`, `y2[i]`) sono quelle dell'angolo in basso a destra.

Viene fornito un programma `mpi-bbox.c` che risolve il problema in modo essenzialmente sequenziale, dato che solo il processo master effettua le computazioni. Scopo di questa esercitazione è di parallelizzare il programma in modo che tutti i processi MPI cooperino per il calcolo del bounding box. In particolare, il programma deve funzionare secondo i passi seguenti:

1. Il master legge i dati dal file di input, inserendo le coordinate negli array `x1[]`, `y1[]`, `x2[]`, `y2[]`; si può inizialmente assumere che il numero di rettangoli $N$ sia un multiplo del numero $P$ di processi MPI.
2. Il master comunica il valore $N$ ai processi (usando `MPI_Bcast`), e distribuisce le coordinate tra usando `MPI_Scatter`; in questo modo ogni processo riceve i dati di $N/P$ rettangoli.
3. Ciascun processo calcola il *bounding box* dei rettangoli a lui assegnati.
4. Il master usa `MPI_Reduce` per calcolare i minimi/massimi delle coordinate dei bounding box locali, determinando in questo modo il bounding box complessivo.

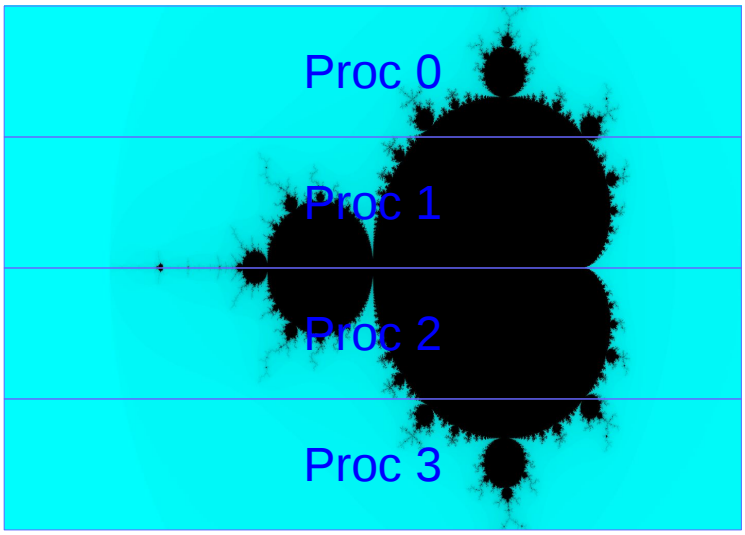
Nell'archivio dell'esercitazione è fornito anche un programma `bbox-gen.c` che può essere usato per generare dei file di input composti da rettangoli generati casualmente.

Dopo aver risolto il problema assumendo che $N$ sia multiplo di $P$, modificare il codice per funzionare correttamente per valori di $N$ arbitrari. Si suggerisce di far gestire la porzione di rettangoli eccedente (cioè il resto della divisione intera $N/P$) al master, in modo da poter usare comunque `MPI_Scatter` per distribuire i dati rimanenti.

## 2. Insieme di Mandelbrot

Il file `mpi-mandelbrot.c` contiene lo scheletro di una implementazione MPI dell'algoritmo che calcola l'insieme di Mandelbrot; non si tratta di una versione realmente parallela, in quanto il processo master è l'unico che esegue computazioni. Il programma produce un file `mandebrot.ppm` contenente una immagine in formato PPM (*Portable Pixmap*) dell'insieme di Mandelbrot.

Scopo di questo esercizio è la realizzazione di una versione realmente parallela del programma, in cui tutti i processi MPI cooperano al calcolo dell'immagine. In particolare, si richiede di partizionare l'immagine a blocchi per righe, in modo che ogni processo calcoli una "fetta" dell'immagine, come schematizzato nella figura seguente



Suggerimento: ciascun processo alloca e calcola una porzione di immagine di dimensione xsize × ysize / *P*, dove *P* è il numero di processi MPI utilizzati. Il master ricostruisce tutte le porzioni con una operazione `MPI_Gather()`. E' possibile comunicare ciascuna porzione di bitmap trattandola come una sequenza di (xsize × ysize / *P* × 3) elementi di tipo `MPI_BYTE`. Assumere inizialmente che la dimensione verticale dell'immagine sia un multiplo di *P*; una volta che si è ottenuto un programma funzionante, si provi a modificarlo per farlo funzionare correttamente con dimensione verticale arbitraria.

## *3. Broadcast tramite comunicazioni punto-punto*

A lezione abbiamo accennato, senza entrare nel dettaglio, che le operazioni di comunicazione collettiva MPI possono essere realizzate in modo efficiente; ad esempio, per effettuare una operazione di broadcast tra *P* processi MPI, anziché effettuare *P* - 1 operazioni `MPI_Send()` si può sfruttare una comunicazione strutturata "ad albero" per completare l'operazione in (log *P*) fasi.

Scopo di questo esercizio è di implementare una versione semplificata di `MPI_Bcast()` utilizzando solo `MPI_Send()` e `MPI_Recv()`, strutturando la comunicazione ad albero. La funzione che vogliamo implementare deve avere la seguente segnatura:

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

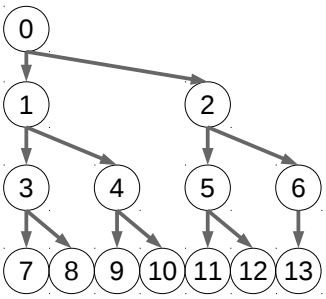
e deve in pratica produrre lo stesso risultato di:

```
MPI_Bcast( v, 1, MPI_INT, 0, MPI_COMM_WORLD )
```

In altre parole, `my_Bcast(&v)` fa in modo che il processo 0 mandi un singolo valore intero *v* a tutti gli altri processi MPI. Per fare questo:

- Ogni processo $p > 0$ riceve *v* dal processo $(p - 1)/2$;
- Ogni processo (incluso il master) invia *v* ai processi $(2p + 1)$ e $(2p + 2)$ (purché i destinatari esistano, cioè il loro id sia $< P$).

Ad esempio, nel caso $P = 14$ si otterrebbe lo schema seguente (si noti che quanto sopra funziona qualunque sia il numero *P* di processi)




Il file `mpi-my-bast.c` contiene lo scheletro di un programma che fa uso della funzione `my_Bcast()` di cui sopra; si noti che la funzione è implementata usando `MPI_Bcast()`. Riscrivere la funzione usando `MPI_Send()` e `MPI_Recv()` come descritto sopra, e testarla per vari valori di *P*. Se lo si ritiene utile si può sfruttare il fatto che usando `MPI_PROC_NULL` come id del destinatario di una `MPI_Send()`, l'operazione viene ignorata.