Lab 6d - heat con MPI

https://elly2023.smfi.unipr.it/mod/page/view.php?id=5404

OBIETTIVO
Parallelizzazione di mpi_heat.c tramite scomposizione del dominio 1D in sottodomini.

MPI HEAT

Attività svolte
Dopo aver creato la directory di lavoro "*mkdir ~/HPC2324/mpi/heat/*", ho copiato al suo interno i file da "*cp /hpc/home/roberto.alfieri/SHARE/mpi/heat/* .*"

Ho individuato nel codice la scomposizione di dominio, il calcolo relativo ai sottodomini e la comunicazione tra i task.
Nota: WNY (Whole NY = 256) è il numero complessivo di righe che viene suddiviso per il numero di task (+2 per le lo scambio d ei bordi tra task adiacenti).

```
49    MPI_Init(&argc, &argv);
50    MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);
51    MPI_Comm_size(MPI_COMM_WORLD, &mpi_size);
52    MPI_Get_processor_name(hostname,&namelen);
53
54  // scomposizione di dominio:
55  // dividiamo le righe tra i rank MPI
56    prev_rank = (mpi_rank-1+mpi_size) % mpi_size;
57    next_rank = (mpi_rank+1) % mpi_size;
58    NX      =  WNX;                       // Local NX: numero colonne per rank
59    NY      =  WNY/mpi_size+2;            // Local NY: numero righe   per rank
```

```
80  // scambio delle righe di bordo
81    // downward
82    MPI_Sendrecv( &(h_T_old[NX*(NY-2)]), NX, MPI_FLOAT, next_rank, tag, //send
83                  &(h_T_old[NX*0]),      NX, MPI_FLOAT, prev_rank, tag, //recv
84                  MPI_COMM_WORLD, MPI_STATUS_IGNORE);
85    // upward
86    MPI_Sendrecv( &(h_T_old[NX*1]),    NX, MPI_FLOAT, prev_rank, tag, //send
87                  &(h_T_old[NX*(NY-1)]), NX, MPI_FLOAT, next_rank, tag, //recv
88                  MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

Osservazione (Comunicazione MPI):
- Al termine di ogni iterazione è necessario che ogni task scambi le righe di bordo con i task adiacenti.
- Al termine della simulazione occorre ricomporre il dominio sul rank 0.

Ho testato il funzionamento e la correttezza del programma con una piccola griglia e poche iterazioni (-s), tramite i seguenti comandi:
module load gnu openmpi
mpicc -O2 mpi_heat.c -o mpi_heat

mpirun mpi_heat -h

```
[martina.genovese@ui01 heat]$ mpirun mpi_heat -h
--------------------------------------------------------
Primary job  terminated normally, but 1 process returned
a non-zero exit code.. Per user-direction, the job has been aborted.
--------------------------------------------------------

mpi_heat [-c ncols] [-r nrows] [-s nsteps] [-h]

mpi_heat [-c ncols] [-r nrows] [-s nsteps] [-h]
```

mpirun mpi_heat -r 24 -c 24 -s 4

```
[martina.genovese@ui01 heat]$ mpirun mpi_heat -r 24 -c 24 -s 4
--------------------------------------------------------------------------
[[62685,1],3]: A high-performance Open MPI point-to-point messaging module
was unable to find any relevant network interfaces:

Module: OpenFabrics (openib)
  Host: ui01

Another transport will be used instead, although this may result in
lower performance.
--------------------------------------------------------------------------
0.68 0.72 0.73 0.73 0.73 0.73 0.73 0.73 0.73 0.73 0.73 0.73 0.73 0.73 0.73 0.73
0.73 0.73 0.73 0.73 0.72 0.43
0.38 0.48 0.50 0.51 0.51 0.51 0.51 0.51 0.51 0.51 0.51 0.51 0.51 0.51 0.51 0.51
0.51 0.51 0.51 0.50 0.48 0.38
0.12 0.17 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18
0.18 0.18 0.18 0.18 0.17 0.12
0.03 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04
0.04 0.04 0.04 0.04 0.04 0.03
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.03 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04
0.04 0.04 0.04 0.04 0.04 0.03
0.12 0.17 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18
0.18 0.18 0.18 0.18 0.17 0.12
0.38 0.48 0.50 0.51 0.51 0.51 0.51 0.51 0.51 0.51 0.51 0.51 0.51 0.51 0.51 0.51
0.51 0.51 0.51 0.50 0.48 0.38
# WNX:24, WNY:24, MAX_ITER:4, MPI_RANK:0, MPI_SIZE:4, NX:24, NY:8, ui01.hpc.unip
r.it
MPI, 4, 24, 24, 4,  0.000067
# WNX:24, WNY:24, MAX_ITER:4, MPI_RANK:1, MPI_SIZE:4, NX:24, NY:8, ui01.hpc.unip
r.it
# WNX:24, WNY:24, MAX_ITER:4, MPI_RANK:2, MPI_SIZE:4, NX:24, NY:8, ui01.hpc.unip
r.it
# WNX:24, WNY:24, MAX_ITER:4, MPI_RANK:3, MPI_SIZE:4, NX:24, NY:8, ui01.hpc.unip
r.it
[ui01.hpc.unipr.it:123320] 3 more processes have sent help message help-mpi-btl-
base.txt / btl:no-nics
[ui01.hpc.unipr.it:123320] Set MCA parameter "orte_base_help_aggregate" to 0 to
see all help / error messages
```

Generazione Plot Scaling
Ho modificato ed eseguito lo script mpi_heat_scaling.slurm facendo variare il numero di task e anche la dimensione del problema con lati di 2048, 4096 e 8192 punti. (l'esecuzione ha richiesto circa 7 minuti)
mpi_heat_scaling.slurm

```
#!/bin/bash
#SBATCH --output=%x.o%j
#SBATCH --partition=cpu_guest
#SBATCH --qos=cpu_guest
#SBATCH --ntasks=32
##SBATCH --nodes=1
##SBATCH --ntasks-per-node=4
#SBATCH --time=0-00:15:00
##SBATCH  --exclude=wn20

echo  "#SLURM_JOB_NODELIST      : $SLURM_JOB_NODELIST"
echo  "#SLURM_JOB_CPUS_PER_NODE : $SLURM_JOB_CPUS_PER_NODE"

module load gnu openmpi

mpicc -O2 mpi_heat.c -o  mpi_heat

for N in 2048 4096 8192
do
    for NT in 1 2 4 8 16 32
     do
        mpirun -n $NT mpi_heat -c $N -r $N 1> /dev/null
    done
done  2> mpi_heat_scaling.dat
```

mpi_heat_scaling.dat (solo l'inizio)

```
[martina.genovese@ui01 heat]$ more mpi_heat_scaling.dat
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:0, MPI_SIZE:1, NX:2048, NY:2050, wn24
MPI, 1, 2048, 2048, 1000,  7.540130
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:0, MPI_SIZE:2, NX:2048, NY:1026, wn24
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:1, MPI_SIZE:2, NX:2048, NY:1026, wn24
MPI, 2, 2048, 2048, 1000,  3.934358
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:2, MPI_SIZE:4, NX:2048, NY:514, wn24
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:1, MPI_SIZE:4, NX:2048, NY:514, wn24
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:3, MPI_SIZE:4, NX:2048, NY:514, wn24
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:0, MPI_SIZE:4, NX:2048, NY:514, wn24
MPI, 4, 2048, 2048, 1000,  2.826716
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:2, MPI_SIZE:8, NX:2048, NY:258, wn24
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:5, MPI_SIZE:8, NX:2048, NY:258, wn24
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:1, MPI_SIZE:8, NX:2048, NY:258, wn24
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:7, MPI_SIZE:8, NX:2048, NY:258, wn24
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:4, MPI_SIZE:8, NX:2048, NY:258, wn24
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:6, MPI_SIZE:8, NX:2048, NY:258, wn24
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:0, MPI_SIZE:8, NX:2048, NY:258, wn24
# WNX:2048, WNY:2048, MAX_ITER:1000, MPI_RANK:3, MPI_SIZE:8, NX:2048, NY:258, wn24
```

Ho determinato il plot di scaling eseguendo il programma mpi_heat_scaling.py, opportunatamente modificato per plottare i dati delle tre diverse dimensioni (2048, 4096, 8192).

mpi_heat_scaling.py

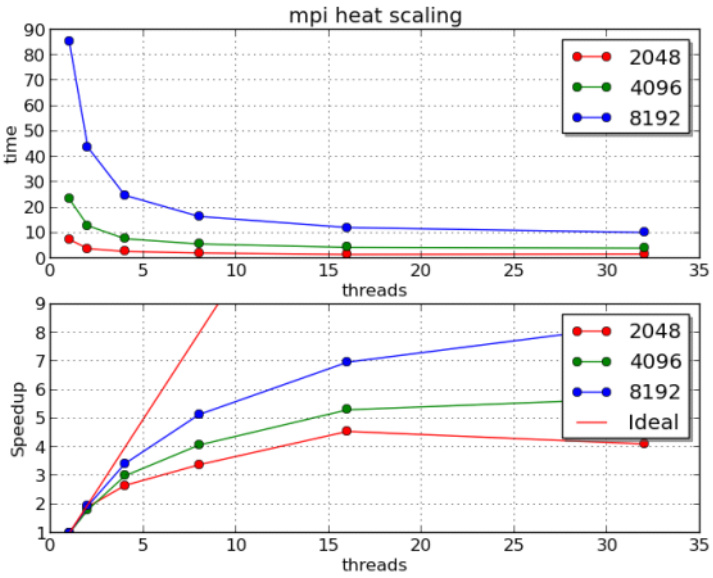
mpi_heat_scaling.py

```
#!/usr/bin/env python2

import matplotlib
matplotlib.use('Agg') # backend per png
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv("mpi_heat_scaling.dat" , comment="#", names=["nt", "r", "c", "iter", "t"])

#print (df)

df1 = df[df["r"] == 2048]
df2 = df[df["r"] == 4096]
df3 = df[df["r"] == 8192]

print (df1)
print (df2)
print (df3)

plt.subplot(2,1,1)

plt.title('mpi heat scaling')

plt.grid()
plt.xlabel('threads')
plt.ylabel('time')
plt.plot(df1.nt,df1.t,'-o', label='2048', color='red')
plt.plot(df2.nt,df2.t,'-o', label='4096', color='green')
plt.plot(df3.nt,df3.t,'-o', label='8192', color='blue')
plt.legend(shadow=True,loc="best")

plt.subplot(2,1,2)

plt.grid()
plt.xlabel('threads')
plt.ylabel('Speedup')
plt.plot(df1.nt,df1.t[0]/df1.t,'-o', label='2048', color='red')
plt.plot(df2.nt,df2.t[0]/df2.t,'-o', label='4096', color='green')
plt.plot(df3.nt,df3.t[0]/df3.t,'-o', label='8192', color='blue')
plt.plot(range(1,10),range(1,10),'-r', label='Ideal')
plt.legend(shadow=True,loc="best")

plt.savefig('mpi_heat_scaling.png')
```

Ouptut (esecuzione mpi_heat_scaling.py)

```
[martina.genovese@ui01 heat]$ python mpi_heat_scaling.py
     nt     r     c  iter         t
MPI   1  2048  2048  1000  7.540130
MPI   2  2048  2048  1000  3.934358
MPI   4  2048  2048  1000  2.826716
MPI   8  2048  2048  1000  2.223397
MPI  16  2048  2048  1000  1.654733
MPI  32  2048  2048  1000  1.831598
     nt     r     c  iter         t
MPI   1  4096  4096  1000 23.496877
MPI   2  4096  4096  1000 12.960177
MPI   4  4096  4096  1000  7.825592
MPI   8  4096  4096  1000  5.759054
MPI  16  4096  4096  1000  4.423157
MPI  32  4096  4096  1000  4.110167
     nt     r     c  iter         t
MPI   1  8192  8192  1000 85.417846
MPI   2  8192  8192  1000 43.807306
MPI   4  8192  8192  1000 24.903024
MPI   8  8192  8192  1000 16.591424
MPI  16  8192  8192  1000 12.239739
MPI  32  8192  8192  1000 10.282704
```

Plot (mpi_heat_scaling.png)



Analisi delle prestazioni e Motivazione della scalabilità limitata
Per dimensioni del problema piccole, ho poca computazione rispetto alla comunicazione che quindi diventa prevalente, di conseguenza non c'è nessun miglioramento nell'aggiungere nuovi task (come possiamo notare dalla curva rossa, matrice 2048, che è motlo lontana da quella ideale)
Invece aumentado le dimensioni del problema, in questo modo aggiungo calcolo, ovvero computazione, e il tempo di comunicazione (overhead) diventa sempre meno rilevante. Lo speedup si avvicina sempre più alla linea ideale.

(Nota) *Per scalabilità di un problema si ci riferisce alla capacità di gestire un aumento della dimensione o della complessità senza subire un degrado significativo delle prestazioni o dell'efficacia. In altre parole, un problema è considerato scalabile se, aumentando la quantità di dati, il sistema riesce a mantenere performance adeguate, anziché diventare lento o inaffidabile.*

HEAT OVERLAP

Attività svolte

OTTIMIZZAZIONE HEAT: SOVRAPPOSIZIONE COMUNICAZIONE/COMPUTAZIONE
Ho creato una copia di mpi_heat.c, rinominandola in mpi_heat_overlap.c, progettata per effettuare una mitigazione dell'overhead dovuto alla comunicazione sovrapponendo computazione e comunicazione:

In particolare ho aggiunto/modificato le seguenti righe di codice:

- Primitive non-bloccanti: ho utilizzato le primitive non bloccanti per lo scambio di messaggi tra i processi. In questo modo viene sovrapposta la comunicazione alla computazione, riducendo così i tempi di attesa.
- Tra le due iterazioni di Jacobi vengono effettuate le chiamate MPI_WAIT per garantire che i dati della comunicazione precedente siano stati ricevuti. Questo è fondamentale poiché il calcolo dei nuovi valori di temperatura dipende dai dati ricevuti dai processi vicini attraverso lo scambio delle righe di confine.

```
96          // scambio delle righe di bordo
97          // utilizzo di primitive non bloccanti per lo scambio delle righe
98          MPI_Request requestS1;
99          MPI_Request requestR1;
100         MPI_Request requestS2;
101         MPI_Request requestR2;
102
103         // downward (non bloccante)
104         MPI_Isend(&(h_T_old[NX*(NY-2)]), NX, MPI_FLOAT, next_rank, tag, MPI_COMM_WORLD, &requestS1);
105         MPI_Irecv(&(h_T_old[NX*0]), NX, MPI_FLOAT, prev_rank, tag, MPI_COMM_WORLD, &requestR1);
106
107         // upward (non bloccante)
108         MPI_Isend(&(h_T_old[NX*1]), NX, MPI_FLOAT, prev_rank, tag, MPI_COMM_WORLD, &requestS2);
109         MPI_Irecv(&(h_T_old[NX*(NY-1)]), NX, MPI_FLOAT, next_rank, tag, MPI_COMM_WORLD, &requestR2);
110
111         // esecuzione di jacobi per le parti interne della griglia
112         Jacobi_Iterator_CPU_internal(h_T_old, h_T_new, NX, NY);
113
114         // ritorno sulle primitive non bloccanti con attesa del completamento della comunicazione
115         MPI_Wait(&requestR1, &status);
116         MPI_Wait(&requestR2, &status);
117
118         // esecuzione di jacobi per le parti esterne della griglia
119         Jacobi_Iterator_CPU_external(h_T_old, h_T_new, NX, NY);
```

- sistema di checkpoint: viene salvato lo stato di avanzamento della simulazione nel file colormap.dat. Se il flag_load è impostato su true e il rank MPI è 0, il programma carica i dati dal file colormap.dat nella matrice h_T_whole, suddividendo poi il dominio per ogni task tramite MPI_Scatter.

```
68      // sistema di checkpoint
69      if (flag_load && mpi_rank == 0) {
70          fprintf(stderr, "Loading the colormap...\n");
71          load_colormap(h_T_whole, WNX, WNY);
72          print_colormap(h_T_whole, WNX, WNY);
73          fprintf(stderr, "\n");
74
75          int i,j;
76          for (i=2; i<NY; ++i) {
77              for (j=0; j<NX; ++j) {
78                  h_T_old[NX*i + j] = h_T_whole[NX*i + j];
79              }
80          }
81
82      }
83
84      MPI_Scatter(h_T_whole, NX*(NY-2), MPI_FLOAT, h_T_new, NX*(NY-2), MPI_FLOAT, 0, MPI_COMM_WORLD);
```

- Raccolta dei dati: MPI_Gather raccoglie i dati parziali da tutti i processi MPI e li aggrega in un'unica matrice h_T_whole sul processo radice. Durante l'avanzamento della simulazione, i risultati parziali vengono scritti nel file colormap.dat se il flag_write è impostato su true e il rank MPI è 0.

```
125         // raccolta dati parziali
126         if (flag_write) MPI_Gather(&(h_T_old[NX]), NX*(NY-2), MPI_FLOAT, h_T_whole,  NX*(NY-2), MPI_FLOAT,  0,  MPI_COMM_WORLD);
127
128         if (flag_write && mpi_rank == 0) {
129             save_colormap(h_T_whole, WNX, WNY);
130         }
```

- Stampa dei dati: alla fine della simulazione, se il flag_write è falso e il numero massimo di iterazioni (MAX_ITER) è diverso da zero, se il rank MPI è zero i risultati vengono raccolti e stampati.

```
145     if (!flag_write && MAX_ITER != 0) MPI_Gather(&(h_T_old[NX]), NX*(NY-2), MPI_FLOAT, h_T_whole,  NX*(NY-2), MPI_FLOAT,  0,
        MPI_COMM_WORLD);
146
147     if (mpi_rank == 0) print_colormap(h_T_whole,WNX,WNY);
```

mpi_heat_overlap.c

```
/*
mpi_heat.c


module load gnu openmpi
mpicc -O2 mpi_heat.c -o mpi_heat


mpirun mpi_heat -h
mpirun mpi_heat -r 24 -c 24 -s 4
*/


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <mpi.h>
#include <mpi_heat.h>


int WNX = 256;         // --- Number of discretization points along the x axis
int WNY = 256;         // --- Number of discretization points along the y axis
int MAX_ITER = 1000;  // --- Number of Jacobi iterations
int NX,NY;
```

```c
#include "mpi_heat.h"


// variabili globali per MPI
int mpi_rank=0, mpi_size=0;
int prev_rank=0, next_rank=0;
int tag = 999;
MPI_Status status;
char hostname[MPI_MAX_PROCESSOR_NAME]; int namelen;

float *h_T_new, *local_h_T_new;
float *h_T_old, *local_h_T_old;
float *h_T_temp;
float *h_T_whole;

/********/
/* MAIN */
/********/
int main(int argc, char **argv)
{
    int iter,i,j;

    double t1, t2;

    options(argc, argv);         /* optarg management */

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &mpi_size);
    MPI_Get_processor_name(hostname,&namelen);

    // scomposizione di dominio:
    // dividiamo le righe tra i rank MPI
    prev_rank = (mpi_rank-1+mpi_size) % mpi_size;
    next_rank = (mpi_rank+1) % mpi_size;
    NX =  WNX;  // Local NX: numero colonne per rank
    NY =  WNY/mpi_size+2;  // Local NY: numero righe   per rank

    // stampa tutti i dati della simulazione
    fprintf (stderr,"# WNX:%d, WNY:%d, MAX_ITER:%d, MPI_RANK:%d, MPI_SIZE:%d, NX:%d, NY:%d, %s \n", WNX, WNY, MAX_ITER,
mpi_rank, mpi_size, NX, NY, hostname);

    h_T_new = (float *)calloc(NX * NY, sizeof(float));
    h_T_old = (float *)calloc(NX * NY, sizeof(float));
    if (mpi_rank == 0)  h_T_whole  = (float *)calloc(WNX * WNY, sizeof(float));


    // sistema di checkpoint
    if (flag_load && mpi_rank == 0) {
        fprintf(stderr, "Loading the colormap...\n");
        load_colormap(h_T_whole, WNX, WNY);
        print_colormap(h_T_whole, WNX, WNY);
        fprintf(stderr, "\n");


        int i,j;
        for (i=2; i<NY; ++i) {
            for (j=0; j<NX; ++j) {
                h_T_old[NX*i + j] = h_T_whole[NX*i + j];
            }
        }

    }

    MPI_Scatter(h_T_whole, NX*(NY-2), MPI_FLOAT, h_T_new, NX*(NY-2), MPI_FLOAT, 0, MPI_COMM_WORLD);


    t1=MPI_Wtime();

    for(iter=0; iter<MAX_ITER; iter=iter+1)
    {
        // Init_center(h_T_old,  NX, NY);
        // Init_left(h_T_old,    NX, NY);
        if (mpi_rank==0)   Init_top(h_T_old,     NX, NY);
        // copy_rows(h_T_old, NX, NY);
        // copy_cols(h_T_old, NX, NY);


        // scambio delle righe di bordo
        // utilizzo di primitive non bloccanti per lo scambio delle righe
        MPI_Request requestS1;
        MPI_Request requestR1;
        MPI_Request requestS2;
        MPI_Request requestR2;

        // downward (non bloccante)
        MPI_Isend(&(h_T_old[NX*(NY-2)]), NX, MPI_FLOAT, next_rank, tag, MPI_COMM_WORLD, &requestS1);
        MPI_Irecv(&(h_T_old[NX*0]), NX, MPI_FLOAT, prev_rank, tag, MPI_COMM_WORLD, &requestR1);

        // upward (non bloccante)
        MPI_Isend(&(h_T_old[NX*1]), NX, MPI_FLOAT, prev_rank, tag, MPI_COMM_WORLD, &requestS2);
        MPI_Irecv(&(h_T_old[NX*(NY-1)]), NX, MPI_FLOAT, next_rank, tag, MPI_COMM_WORLD, &requestR2);

        // esecuzione di jacobi per le parti interne della griglia
        Jacobi_Iterator_CPU_internal(h_T_old, h_T_new, NX, NY);
```

```
         // ritorno sulle primitive non bloccanti con attesa del completamento della comunicazione
        MPI_Wait(&requestR1, &status);
     MPI_Wait(&requestR2, &status);

         // esecuzione di jacobi per le parti esterne della griglia
        Jacobi_Iterator_CPU_external(h_T_old, h_T_new, NX, NY);

        h_T_temp=h_T_new;
        h_T_new=h_T_old;
        h_T_old=h_T_temp;

        // raccolta dati parziali
        if (flag_write) MPI_Gather(&(h_T_old[NX]), NX*(NY-2), MPI_FLOAT, h_T_whole,  NX*(NY-2), MPI_FLOAT,  0,
MPI_COMM_WORLD);

        if (flag_write && mpi_rank == 0) {
            save_colormap(h_T_whole, WNX, WNY);
        }

        // sincronizzazione (attendo che root completi il salvataggio)
        MPI_Barrier(MPI_COMM_WORLD);

    }

    t2=MPI_Wtime();

    // WNX/WNY = Number of discretization points along the x/y axis
    if(mpi_rank == 0)   fprintf(stderr,"MPI, %d, %d, %d, %d,  %f\n", mpi_size,  WNX, WNY, MAX_ITER, t2-t1);

    // sincronizzazione
    MPI_Barrier(MPI_COMM_WORLD);

    if (!flag_write && MAX_ITER != 0) MPI_Gather(&(h_T_old[NX]), NX*(NY-2), MPI_FLOAT, h_T_whole,  NX*(NY-2), MPI_FLOAT,
0,  MPI_COMM_WORLD);

    if (mpi_rank == 0) print_colormap(h_T_whole,WNX,WNY);

    free(h_T_new);
    free(h_T_old);
    if (mpi_rank == 0 ) free(h_T_whole);

    MPI_Finalize();

    return 0;
}
```

Ho poi compilato ed eseguito il codice tramite lo script mpi_heat_overlap.slurm, ottenendo in output il file mpi_heat_overlap .dat con il seguente contenuto:

mpi_heat_overlap.dat

```
[martina.genovese@ui01 heat]$ sbatch mpi_heat_overlap.slurm
[martina.genovese@ui01 heat]$ more mpi_heat_overlap.dat
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:0, MPI_SIZE:1, NX:8182, NY:2050, wn20
MPI, 1, 8182, 2048, 100,  2.165628
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:0, MPI_SIZE:2, NX:8182, NY:1026, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:1, MPI_SIZE:2, NX:8182, NY:1026, wn20
MPI, 2, 8182, 2048, 100,  1.135754
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:0, MPI_SIZE:4, NX:8182, NY:514, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:2, MPI_SIZE:4, NX:8182, NY:514, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:3, MPI_SIZE:4, NX:8182, NY:514, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:1, MPI_SIZE:4, NX:8182, NY:514, wn20
MPI, 4, 8182, 2048, 100,  0.642876
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:3, MPI_SIZE:8, NX:8182, NY:258, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:0, MPI_SIZE:8, NX:8182, NY:258, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:1, MPI_SIZE:8, NX:8182, NY:258, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:2, MPI_SIZE:8, NX:8182, NY:258, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:4, MPI_SIZE:8, NX:8182, NY:258, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:5, MPI_SIZE:8, NX:8182, NY:258, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:6, MPI_SIZE:8, NX:8182, NY:258, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:7, MPI_SIZE:8, NX:8182, NY:258, wn20
MPI, 8, 8182, 2048, 100,  0.376564
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:9, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:0, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:1, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:2, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:4, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:12, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:3, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:5, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:10, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:11, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:13, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:14, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:7, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:15, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:6, MPI_SIZE:16, NX:8182, NY:130, wn20
# WNX:8182, WNY:2048, MAX_ITER:100, MPI_RANK:8, MPI_SIZE:16, NX:8182, NY:130, wn20
MPI, 16, 8182, 2048, 100,  0.204877
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:0, MPI_SIZE:1, NX:32768, NY:2050, wn20
MPI, 1, 32768, 2048, 100,  8.524255
```

```
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:0, MPI_SIZE:2, NX:32768, NY:1026, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:1, MPI_SIZE:2, NX:32768, NY:1026, wn20
MPI, 2, 32768, 2048, 100,  4.442591
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:0, MPI_SIZE:4, NX:32768, NY:514, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:1, MPI_SIZE:4, NX:32768, NY:514, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:2, MPI_SIZE:4, NX:32768, NY:514, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:3, MPI_SIZE:4, NX:32768, NY:514, wn20
MPI, 4, 32768, 2048, 100,  2.600480
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:7, MPI_SIZE:8, NX:32768, NY:258, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:0, MPI_SIZE:8, NX:32768, NY:258, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:3, MPI_SIZE:8, NX:32768, NY:258, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:4, MPI_SIZE:8, NX:32768, NY:258, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:5, MPI_SIZE:8, NX:32768, NY:258, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:1, MPI_SIZE:8, NX:32768, NY:258, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:2, MPI_SIZE:8, NX:32768, NY:258, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:6, MPI_SIZE:8, NX:32768, NY:258, wn20
MPI, 8, 32768, 2048, 100,  1.510406
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:6, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:0, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:1, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:5, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:3, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:4, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:8, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:12, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:10, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:13, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:2, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:7, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:14, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:15, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:11, MPI_SIZE:16, NX:32768, NY:130, wn20
# WNX:32768, WNY:2048, MAX_ITER:100, MPI_RANK:9, MPI_SIZE:16, NX:32768, NY:130, wn20
MPI, 16, 32768, 2048, 100,  0.902028
```