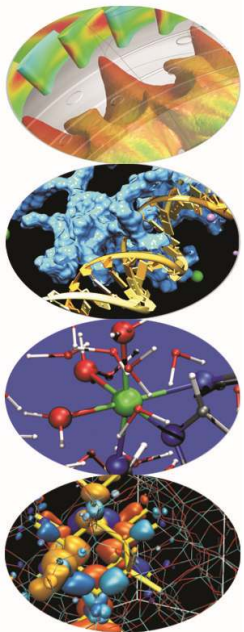# Debugging

- Exercises -

P.Dagna, M.Cremonesi

May 2015

Integral of sin(x)

The serial program sin_trapez_bug computes the integral of sin(x) from 0 to π using the trapezoidal rule.

Compiling and running we obtain the following wrong result

```
 i   x      f(x)  int(f(x))
 1 0.00   1.00 0.00
 2 0.00   1.00 0.00
 3 0.00   1.00 0.00
 4 0.00   1.00 0.00
 5 0.00   1.00 0.00
 6 0.00   1.00 0.00
 7 0.00   1.00 0.00
 8 0.00   1.00 0.00
 9 0.00   1.00 0.00
10 0.00   1.00 0.00
11 3.14  -1.00 0.00
```

Correct result should be like that

```
 i   x      f(x)  int(f(x))
 1 0.00   1.00 0.00
 2 0.31   0.95 0.31
 3 0.63   0.81 0.58
 4 0.94   0.59 0.80
 5 1.26   0.31 0.94
 6 1.57   0.00 0.99
 7 1.88  -0.31 0.94
 8 2.20  -0.59 0.80
 9 2.51  -0.81 0.58
10 2.83  -0.95 0.31
11 3.14  -1.00 0.00
```

Use GDB debugger to find out and resolve the error

Integral of sin(x)

- Hints
  - Remember to compile using the "–g" flag
  - Run the debugger with : gdb executable_name
  - During the debugging phase control the values of variables "i, nm, pi, x2".

Source code : `sin_trapez_bug`

MPI message exchange

- The following sample program executes a message passing between two processes, extra processes are ignored.

- During message passing something goes wrong and the program hangs.
  - Use the GDB debugger to try to find out and solve the bug.

- Hints :
  - Remember to compile using the "–g" flag
  - Run the program with "`mpirun –np 4 xterm –e gdb executable_name`"
  - During the debugging phase control that the parameters of the functions "MPI_send" "MPI_Recv" match.

- **Source code : `mpi_send_recv_bug`**

MPI sum
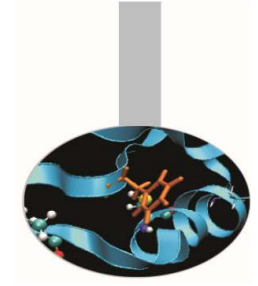
- The sample program *mpi_sum_bug* executes in parallel the sum of the elements of an array using domain decomposition.

- The master process initializes the array and sends to the other processes their own part to compute the partial sum.

- Once the partial sums are completed it's computed the final sum using a reduction operation.

MPI sum

- The result for final sum is wrong.

  – Use the GDB debugger to try to find out and solve the bug.

  – The number of MPI tasks when running the application must be divisible by 4.

- Hints :

  – Remember to compile using the "–g" flag

  – During the debugging phase control if the portion of the array is correctly passed and received among master and slaves and if the reduction operation is correct.

- **Source code : `mpi_sum_bug.c`**

MPI heat2D

- This example is based on a simplified two-dimensional heat equation domain decomposition. The initial temperature is computed to be high in the middle of the domain and zero at the boundaries.

- The grid is decomposed by the master process and then distributed by rows to the worker processes. At each time step, worker processes must exchange border data with neighbors. Upon completion of all time steps, the worker processes return their result to the master process.
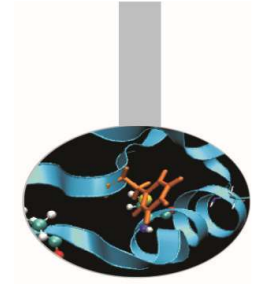
MPI heat2D

- Two data files are produced: an initial data set and a final data set.

- An X graphic of these two states displays after all calculations have completed.

- At the end of simulation something goes wrong and the program crashes.

  – Use the GDB debugger to try to find out and resolve the bug.

  – The number of MPI tasks when running the application must be divisible by 4.

- **Source code : `mpi_heat2D_bug.c draw_heat2D.c`**

MPI heat2D

- **Hints :**
  - Compile the program this way:
 - ```
    mpicc -g -o mpi_heat_2D_bug \
         mpi_heat_2D.c draw_heat2D.c -lX11
    ```
  - Run the program with:
    ```
    mpirun –np 4 xterm –e gdb executable_name
    ```
  - During the debugging phase run the code without inserting breakpoints and look at behavior and messages of MPI processes during finalization phase.

# MPI heat2D

- Once you have solved this bug run the code again and check if the images of initial and final step correspond to the following ones.
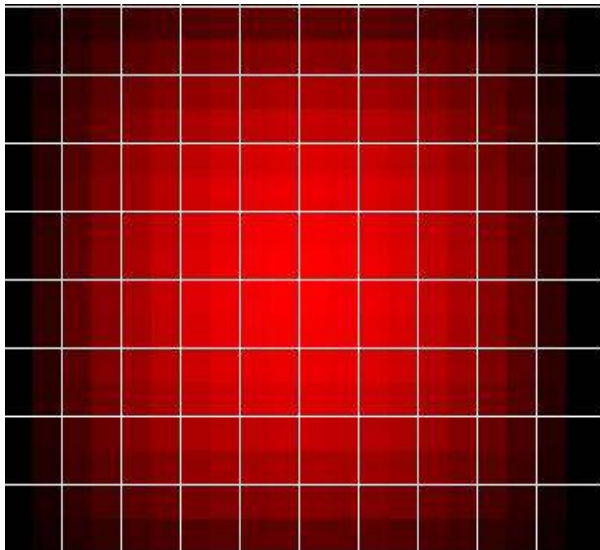


Initial state
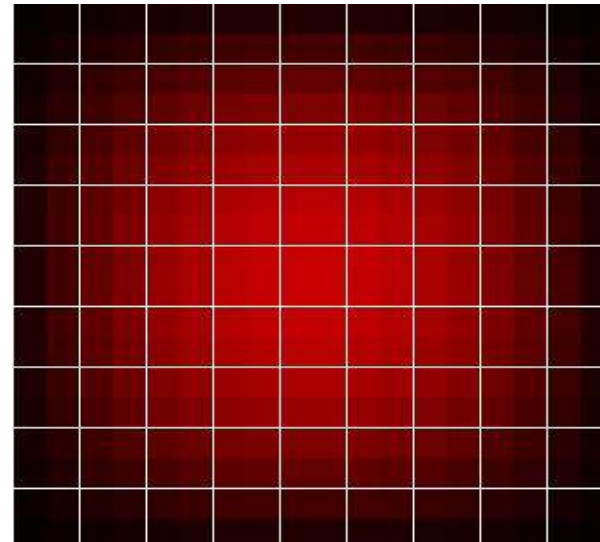


Final state

# MPI heat2D
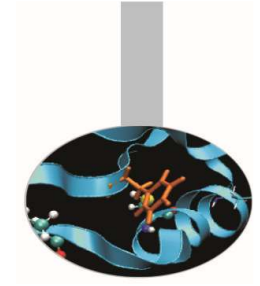
- Unfortunately images don't match.

- Hints :

  - Check the calls to MPI_Send and MPI_Recv from master and slave processes

  - Check if the data of the 2D grid are correctly exchanged