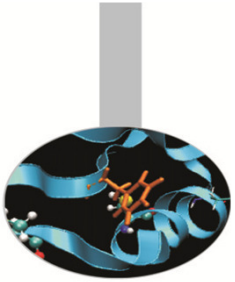
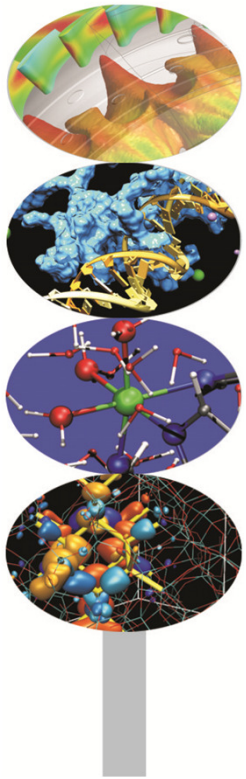
CINECA SCAI
SuperComputing Applications and Innovation

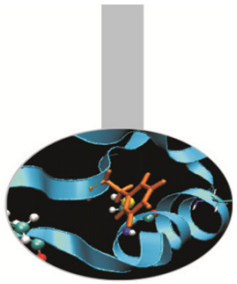


Advanced MPI
- *exercises* -



CINECA

Compiling notes



To compile programs that make use of MPI library:

```
mpif90/mpicc/mpiCC -o <executable> <file 1> <file 2> … <file n>
```

Where: `<file n>` - program source files

`<executable>` - executable file

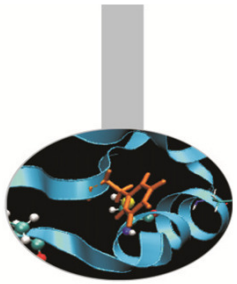
To start parallel execution on one node only:

```
mpirun -np <processor_number> <executable> <exe_params>
```

To start parallel execution on many nodes:

```
mpirun -np <processor_number> -machinefile <node_list_file> \
       <executable> <exe_params>
```

E4 – example – Mandelbrot set



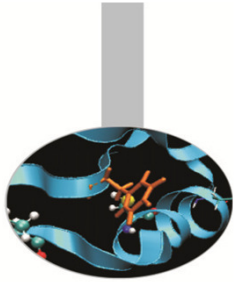
In 1979 Benoît Mandelbrot, who was working at Thomas J. Watson Research Center of IBM, was studying what would have been later known as the *Mandelbrot set*. This mathematical object may be easily studied only by means of numerical computing, with the added support of computer graphics.

Defining the Mandelbrot set is quite easy:

Given the transformation $z \rightarrow z^2$ in the complex plane, iterate it at each point of the circle of radius 2 centred in the origin.

The Mandelbrot set is the set of points that do not diverge outside this circle.

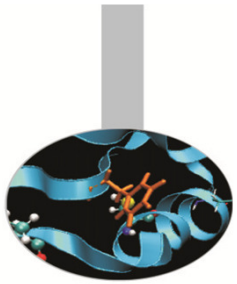
E4 – example – Mandelbrot set



Of course points inside the circle with radius 1 always remain in the set, but there is no simple rules to decide whether the other points do belong to the set or not. The border of the set has fractal properties. Moreover, because of chaos behavior coming from exponent operations, points starting very closed together may diverge considerably.

The example program computes the Mandelbrot set in a given area (inside the radius two circle) and creates an image on the basis of how many iterations are needed to send a point outside the circle. The result is a well known image that can also be used to effectively check the correctness of the program.

E4 – example – Mandelbrot set



The image is generated in PGM or PPM formats because they are very easy to remember and realize.

PGM format:

Row 1 – P2

Row 2 - <rows> <columns>

Row 3 - <Maximum value>

… <point values> …

PPM format:

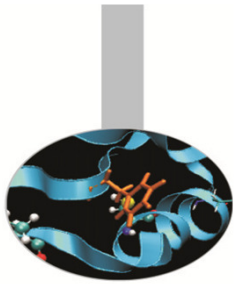
Row 1 – P3

Row 2 - <rows> <columns>

Row 3 - <Maximum value>

… <R G B point values> …

E4 – example – Mandelbrot set



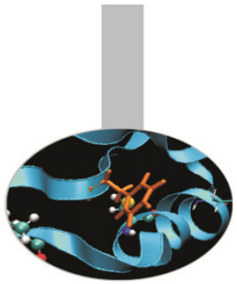
The program could thus be sketched this way:

- Define area in complex plane (squared for simplicity)
- Define image size (squared for simplicity)
- Define maximum iterations per point
- Broadcast data to all processes
- Parallel computation by domain decomposition
- Gather results
- Produce image

Source code: *Mandel*

Reference: *http://mrob.com/pub/muency.html*

E5 – exercise – Matrix multiply



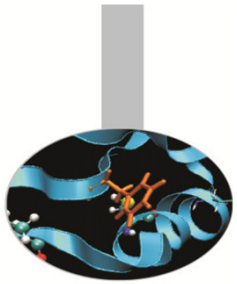
Matrix row-column multiply is an example of program that can be easily parallelized and should have embarassingly parallel behavior too.

Given the matrices A(L,M), B(M,N), C(L,N) try writing a parallel program that computes C = A x B

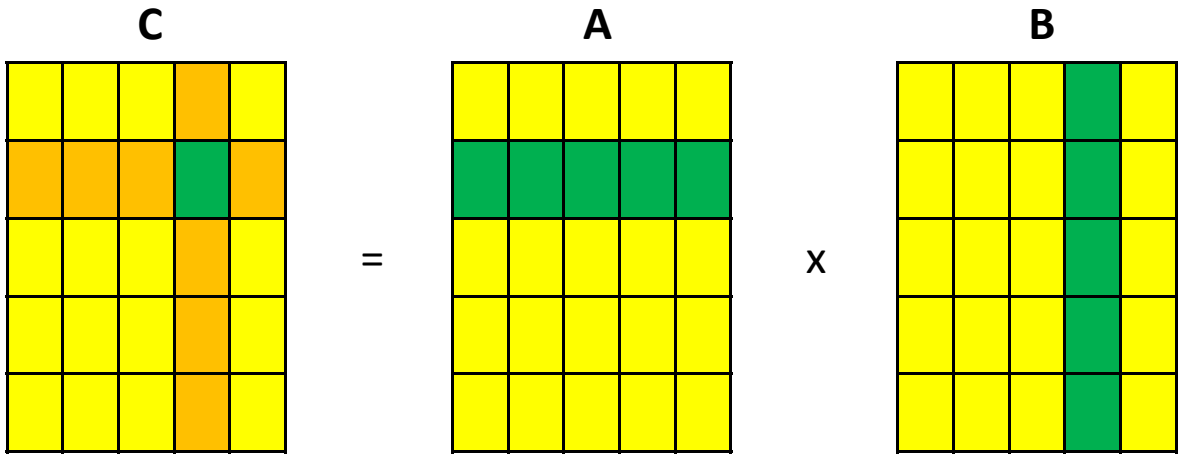
The program could be written this way:

- Decide matrix sizes
- Decide how to distribute computation
- Parallel computation
- Collect results

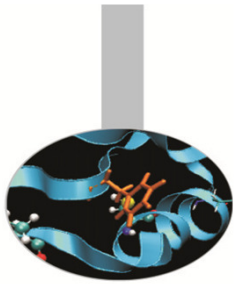
E5 – exercise – Matrix multiply



Hint: given a range of rows and columns in the resulting matrix, their values depend only on a corresponding range of rows in A matrix and columns in B matrix



E6 – example – Life game



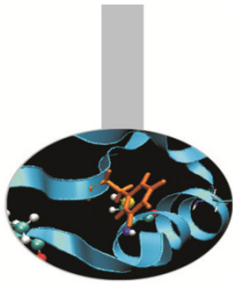
John Conway's LIFE game has been described since 1970 on Scientific American. It consists in a very large checkerboard where there is a initial configuration of marked (or alive) cells. At each iteration per each cell the number F of the alive cells (taken among the 8 adjacent ones) is counted and the cell is marked alive or not according to the following rules:

The cell survives if 2 <= F <= 3

The cell dies if 4 <= F or F <= 1

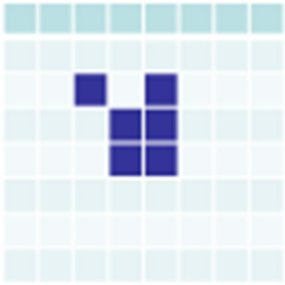
The cell gets alive if F = 3

The game rules are very simple but it is very difficult to predict the population evolution.

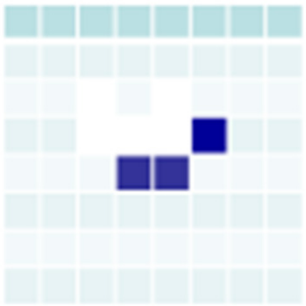


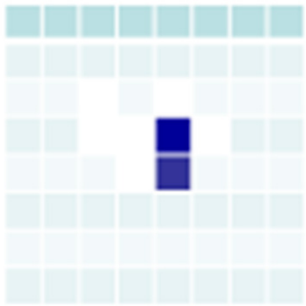
E6 – example – Life game

As an example given a very simple initial configuration:

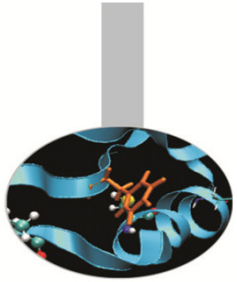


The evolution at next steps are:





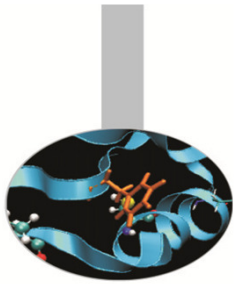
E6 - example – Life game



Programming difficulties for implementing a LIFE game are closed to issues encountered for programming PDE solvers with regular meshes.

A sequential program may be written in the following way:

- Decide board sizes (squared for simplicity) and number of iterations
- Allocate matrix A(:,:) for current state
- Allocate matrix B(:,:) for next state
- Choose an initial configuration
- Iterate:
  - store next state in matrix B by applying rules on matrix A
  - swap matrices



E6 - example – Life game

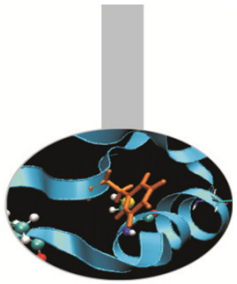
Issues for parallel version:

- Decide board decomposition: divide board in disjoined portions
- At each portion of checkerboard add boundary 1 cell wide
- Distribute portions (with boundaries) to processes
- Iterate:
  - store next state in matrix B by applying rules on matrix A
  - send edges of portions to proper processes
  - receive boundary updates
  - swap matrices

Source code: *LifeGame*

Reference*: http://www.bitstorm.org/gameoflife/*

E7 – exercise – Heat equation



The distribution of heat over time is described by the so called heat equation:

$\frac{df}{dt} = \alpha\left(\frac{d^2f}{dx^2} + \frac{d^2f}{dy^2}\right)$ for a function f(x,y,t).

This formula may be discretized in a regular grid G(:,:) by computing the new value G1(x,y) in a point (x,y) at each time step as:

G1(x,y) = G(x,y) + CX * (G(x+1,y) + G(x-1,y) - 2.0 * G(x,y))

+ CY * (G(x,y+1) + G(x,y-1) - 2.0 * G(x,y))

For each point in the grid the next value depends on the values of the four up and down, left and right adjacent points.

Source code: *Heat*