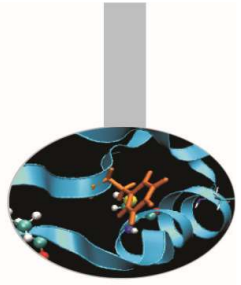
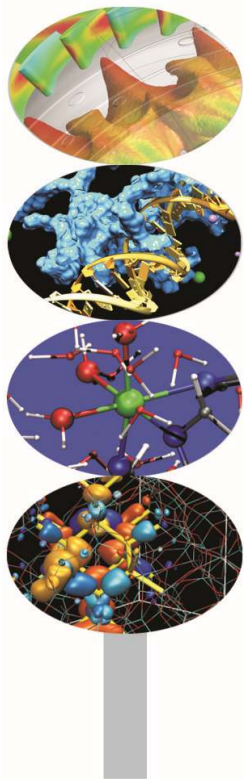
CINECA SCAI
SuperComputing Applications and Innovation



MPI introduction

- *exercises* -

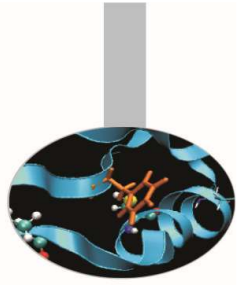


P. Ramieri

May 2015

CINECA

Startup notes



To access the server:

```
ssh a08tra??@login.eurora.cineca.it
```

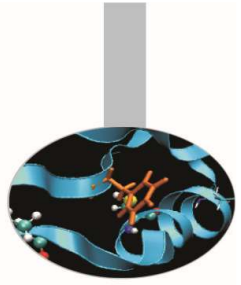
To reserve space on the server:

```
qsub -I -A train_cplm2015 -l select=1:ncpus=4:mpiprocs=4 \
       -l walltime=3:00:00
```

To configure the MPI environment:

```
module load autoload openmpi
```

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

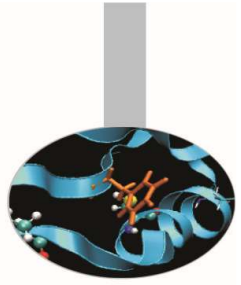
To start parallel execution on one node only:

```
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>
```

Hello world! (Fortran)

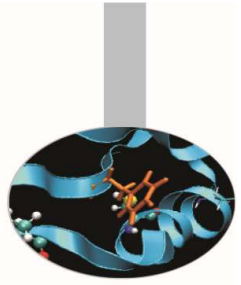


As an ice breaking activity try to compile and run the *Hello* program, either in C or in Fortran.

The most important lines in Fortran code are emphasized:

```
PROGRAM HelloWorld
    INCLUDE 'mpif.h'
    INTEGER my_rank, p
    INTEGER source, dest, tag
    INTEGER ierr, status(MPI_STATUS_SIZE)
    .   .   .
    CALL MPI_Init(ierr)
    CALL MPI_Comm_rank(MPI_COMM_WORLD, my_rank, ierr)
    CALL MPI_Comm_size(MPI_COMM_WORLD, p, ierr)

    WRITE(*,FMT="(A,I)") “Hello world from process ”, my_rank

    CALL MPI_Finalize(ierr)
END PROGRAM HelloWorld
```

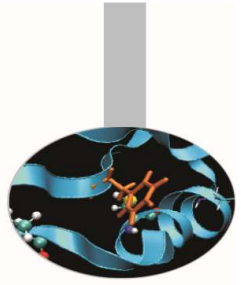
Hello world! (C/C++)



The most important lines in C code are emphasized:

```
#include "mpi.h"

int main( int argc, char *argv[])
{
    int my_rank, numprocs;
    int dest, tag, source;
    MPI_Status status;

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);

    printf("Hello world from process %d\n",my_rank);

    MPI_Finalize();
    return 0;
}
```

Hello world! (output)



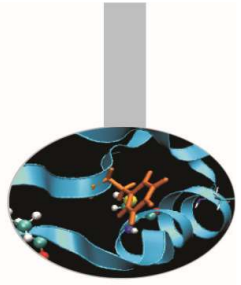
If the program is executed with one process the output is:

```
Hello world from process 0
```

If the program is executed with four processes the output is:

```
Hello world from process 0
Hello world from process 1
Hello world from process 2
Hello world from process 3
```

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

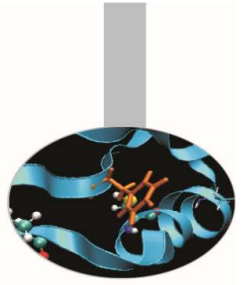
To start parallel execution on one node only:

```
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>
```

E1 – exercise – ping-pong

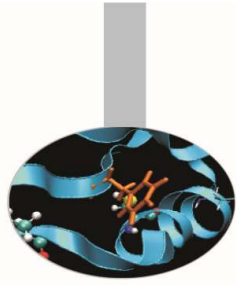


ping-pong is perhaps the simplest example of point to point communication.

In a two process execution of a ping-pong program the process 0 sends a message to process 1 and this sends it back to process 0. This could be easily generalized in a round robin fashion if more than two processes are engaged.

Try modifying the Hello World example in order of realizing round robin communications.

E2 – example – Pi by quadrature

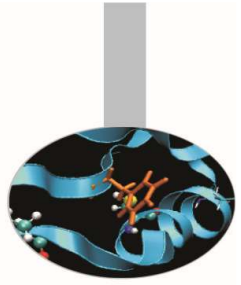


It is known that the mathematical constant π can be approximated by computing the following formula:

$$\pi = 4 \int_0^1 \frac{1}{1+x*x} dx$$

The value of the above integral can be approximated by numerical integration, i.e. by computing the mean value of the function $f(x) = \frac{1}{1+x*x}$ in a number of points and multiplying per the x range. This can be easily done in parallel by dividing the [0,1] range into a number of intervals.

E2 – example – Pi by quadrature

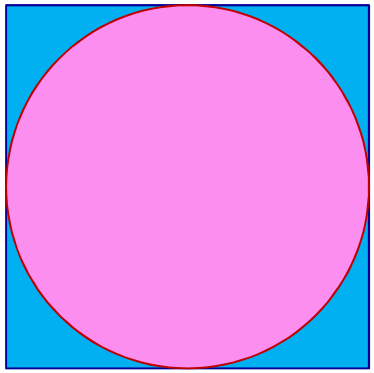


Thus the program may be sketched this way:

- (if my_rank == 0) get number of intervals for quadrature
- Broadcast number of intervals to all the processes
- Assign the intervals to the processes (they should not overlap)
- Sum function values in the centre of each interval
- Divide by interval range and multiply by 4

Source code: *Pi_integral*

E3 – exercise – Montecarlo Pi

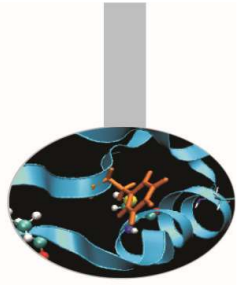


The value of the constant π can be approximated also by recalling that, given A_c the area of the circle and A_s the area of the circumscribing square:

$$\pi = 4 \frac{A_c}{A_s}$$

Thus π could be approximated in a MonteCarlo style by counting the number of the random points in a square that are contained in the inscribed circle.

E3 – exercise – Montecarlo Pi

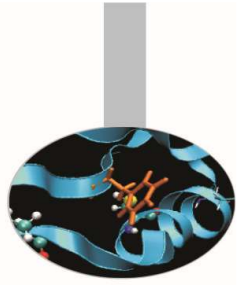


Therefore the program may be written this way:

- Divide a square in an number of parts (as many as the processes)
- Generate a number of random points in the area of each process
- Calculate how many points fall in the inscribed circle
- Sum up number of points in the square
- Sum up number of points in the circle
- Divide the two numbers

Source code: Pi_area

E4 – example – Mandelbrot set



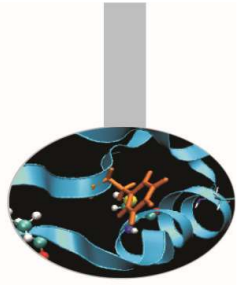
In 1979 Benoît Mandelbrot, who was working at Thomas J. Watson Research Center of IBM, was studying what would have been later known as Mandelbrot set. This mathematical object may be easily studied only by means of numerical computing, with the added support of computer graphics.

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.

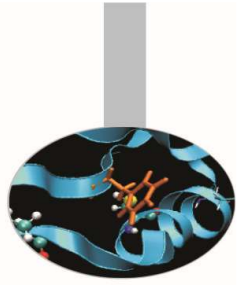
E4 - example – Mandelbrot set



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. In fact 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:

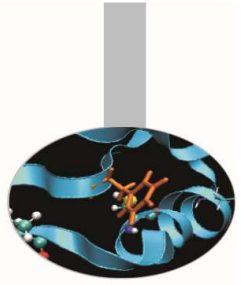
Row 1 – P3

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

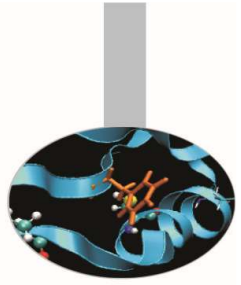
Parallel computation by domain decomposition

Gather results

Produce image

Source code: Mandel

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:

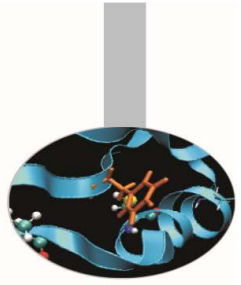
Decide matrix sizes

Decide how to distribute computation

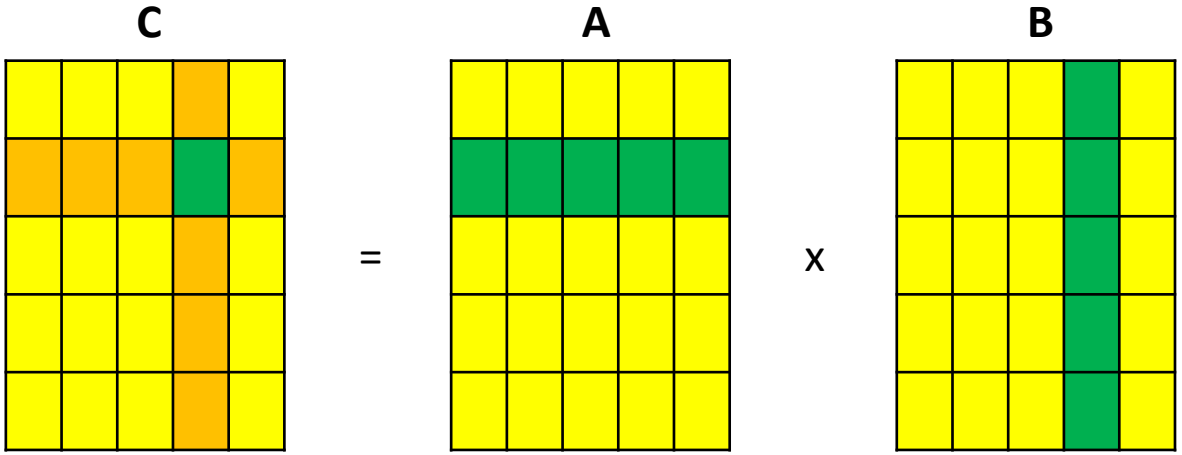
Parallel computation

Collect results

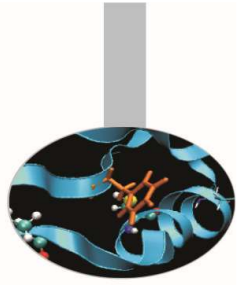
E5 - exercise – Matrix multiply



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

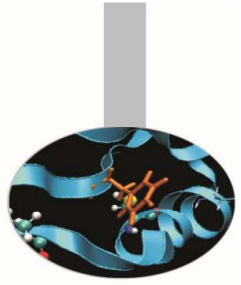


John Conway's LIFE game has been described since 1970 on Scientific American. It consists in a very large checkerboard where there is a initial configuration of marked (or alive) cells. At each iteration per each cell the number F of the alive cells (taken among the 8 adjacent ones) is counted and the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

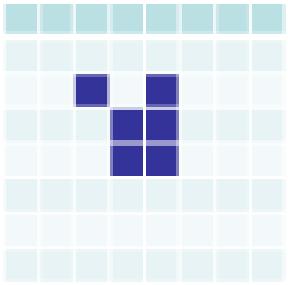
The cell survives if	2 <= F <= 3
The cell dies if	4 <= F or F <= 1
The cell gets alive if	F = 3

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

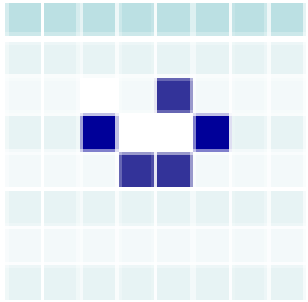
E6 – example – Life game

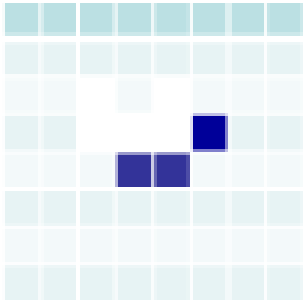


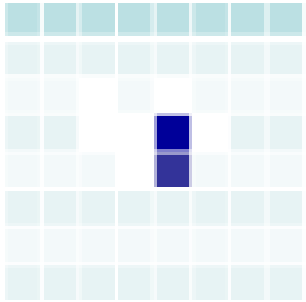
As an example given a very simple initial configuration:



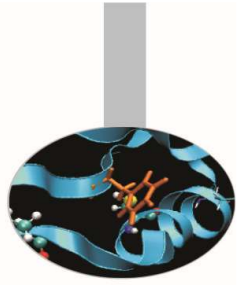
The evolution at next steps are:







E6 - example – Life game

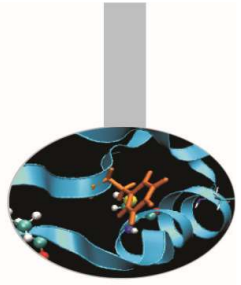


Programming difficulties for implementing a LIFE game are closed to issues encountered for programming PDE solvers with regular meshes.

- 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 1 cell boundary
- 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/*