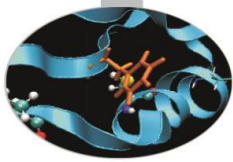
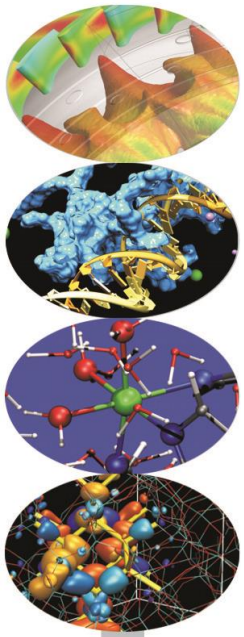
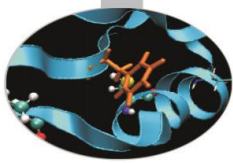


MPI introduction
- *exercises* -





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

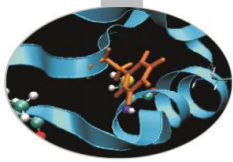
`<executable>` - executable file

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)

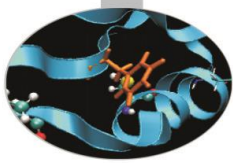
As an ice breaking activity try to compile and run the *Hello* program, either in C or in Fortran.

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:

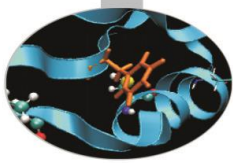
```
#include "mpi.h"

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)

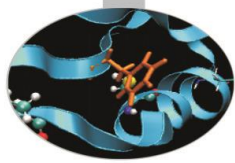
If the program is executed with one process the output is:

```
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
```

E1 – exercise – ping-pong

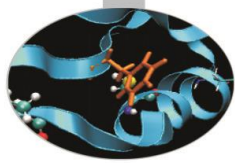


ping-pong is perhaps the simplest example of point to point communication.

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

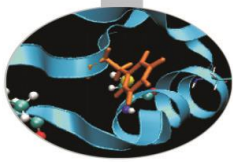


It is known that the mathematical constant π can be approximated by computing the following formula:

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

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*