



**Barcelona  
Supercomputing  
Center**

*Centro Nacional de Supercomputación*

# ALYA Parallel I/O Tutorial

Damien Dosimont

Barcelona, 29/11/2017



**Barcelona  
Supercomputing  
Center**

*Centro Nacional de Supercomputación*

# OVERVIEW

# Parallelizing Alya's I/O

## « Current situation:

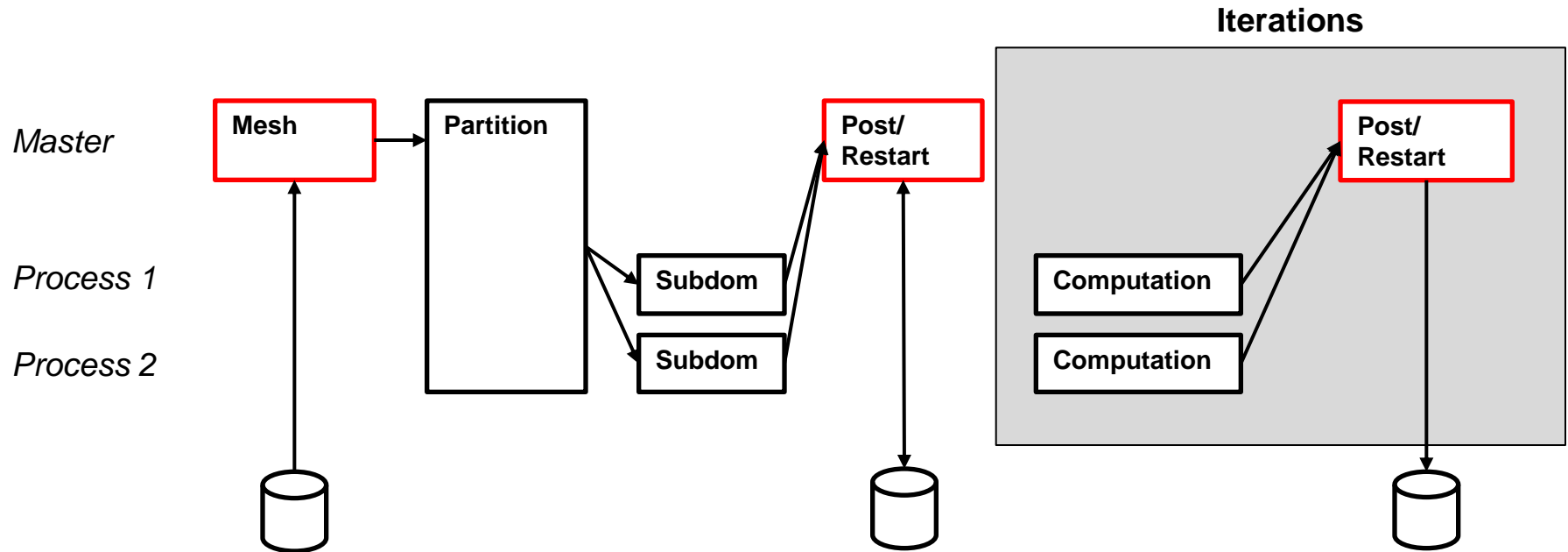
- Alya's I/O are currently performed sequentially
- Postprocess (alya2pos) is sequential as well

## « We propose to parallelize both using MPI-IO:

- Improve Alya I/O performance
  - Reduce mesh reading time
  - Reduce post/restart files reading and writing time
  - Reduce memory footprint on master process
  - Reduce communications to/from master and maintain a whole parallel workflow
- Improve postprocess performance
  - Reduce reading and writing time
  - VTK files management (mesh + fields : nodes/elements)

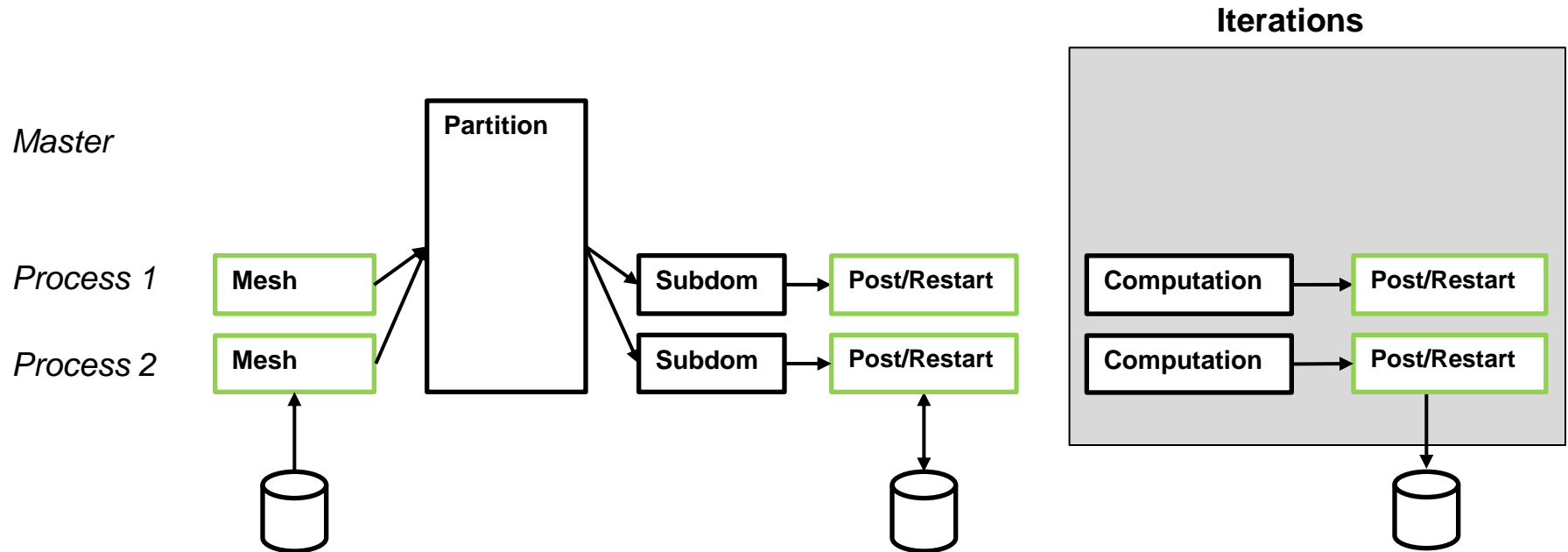
# Parallelizing Alya's I/O

Alya with **sequential IO** (simplified)



# Parallelizing Alya's I/O

**\*\*New\*\***: Alya with **parallel IO** (simplified)





- ❧ alya: MPI-IO feature, already implemented
- ❧ alya-mpio-tools: tool set to manipulate mesh/post/rst files
  - alya2mpio: convert ASCII mesh files to MPI-IO compliant binary files
  - mpio2vtk: convert MPI-IO mesh/post/rst to vtk (parallel) files
  - mpio2txt: dump MPI-IO post process files to txt
  - libalya-mpio-reader: API to write your own converter (sequential or MPI)



**Barcelona  
Supercomputing  
Center**

*Centro Nacional de Supercomputación*

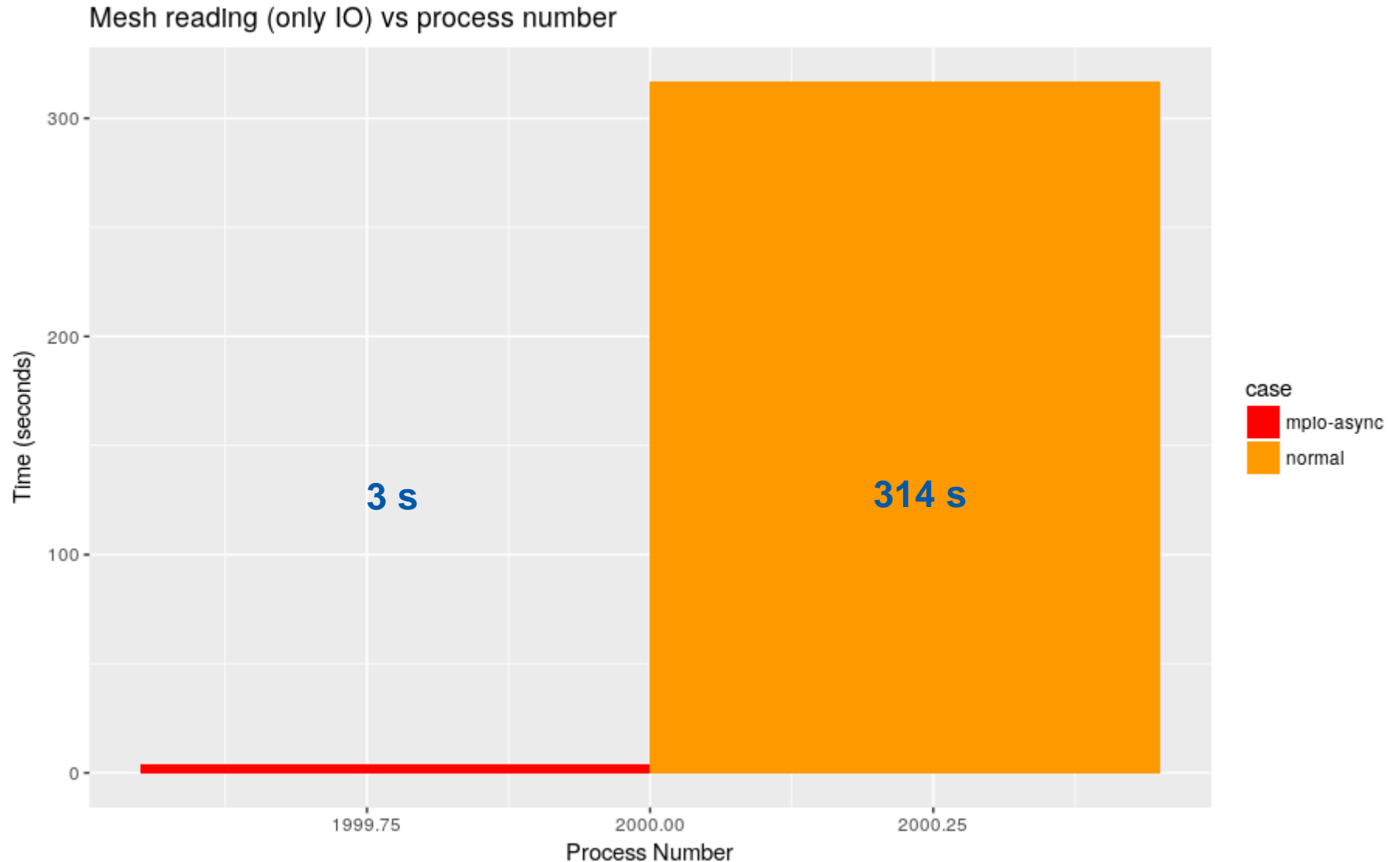
# CURRENT RESULTS

# Example: Case CRM (Oriol), 2000 MPI processes on MN4

- ⌘ POINTS = 46 922 218
- ⌘ ELEMENTS = 158 528 254
- ⌘ BOUNDARIES = 7 025 777
  
- ⌘ ASCII Mesh = 13 GB
- ⌘ MPI-IO format Mesh = 8 GB (integers 8)

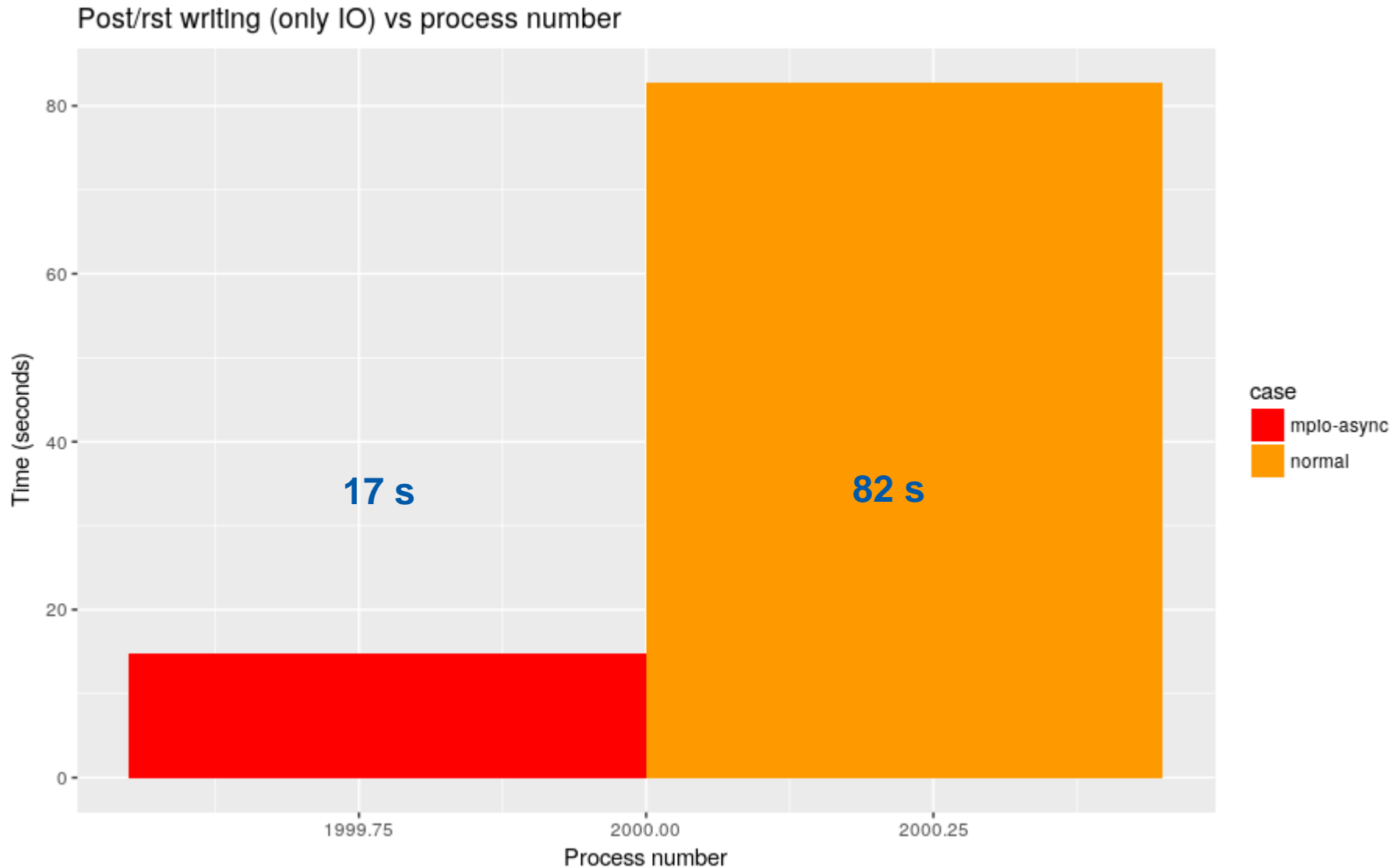


# Mesh reading



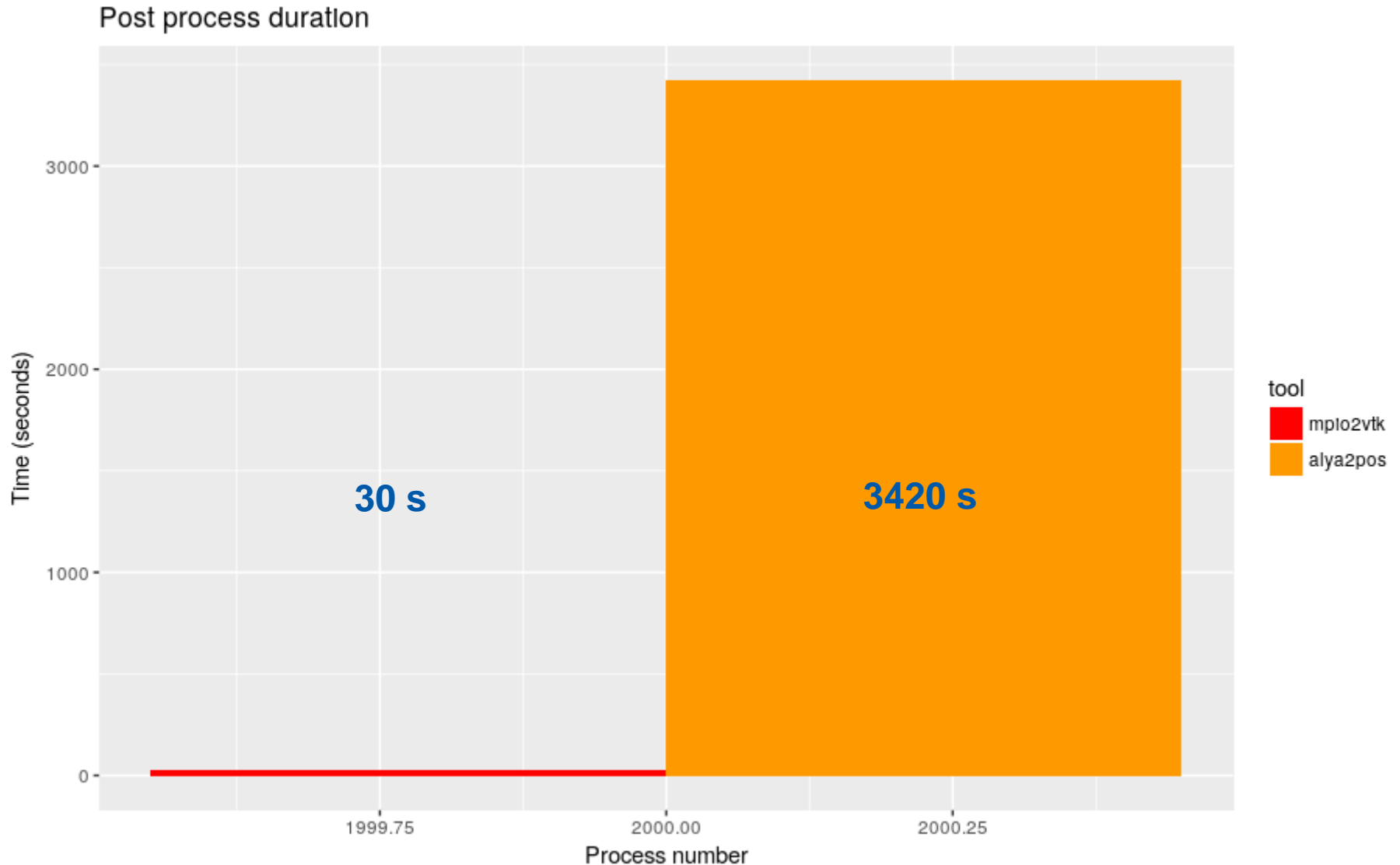
Speed-up: 104.6

# Postprocess files writing



Speed-up: 4.8

# Postprocess files conversion (only one iteration)



Speed-up: 114



**Barcelona  
Supercomputing  
Center**  
*Centro Nacional de Supercomputación*

# TUTORIAL (ON MARENOSTRUM IV)

# Download, configure and compile Alya

## Connect in ssh to MareNostrum 4

```
$ ssh mn1.bsc.es
```

## Create working directories

```
$ mkdir tutorial-mpio  
$ cd tutorial-mpio
```

## Get, configure and compile Alya

```
$ svn co file:///gpfs/projects/bsc21/svnroot/Alya/Trunk alya  
$ cd alya/Executables/unix  
$ cp configure.in/config_ifort.in config.in  
$ ./configure -x nastin parall  
$ salloc -p interactive  
$ make metis4  
$ make -j4  
$ cd ~
```



# Download the example

## Download the example

```
$ cp -r /gpfs/projects/bsc21/damien/share/tutorial/mpio/example ~/scratch/
```

## Go to the example directory

```
$ cd ~/scratch/example/elbow/elbow
```

## Compare with step-0

```
$ diff ../elbow ../step-0
```

# I/O Operations

- « Two separate and independent operations are available in Alya:
  - Parallel mesh reading
  - Post-process and restart files reading/writing
- « Mesh reading requires to convert your mesh into binary files that are compliant with MPI-IO
- « Post-process/restart I/O does not require anything beside configuring the .dat file



**Barcelona  
Supercomputing  
Center**

*Centro Nacional de Supercomputación*

# WRITE POSTPROCESS/RESTART FILES

# Configure the .dat file to enable MPI-IO

Open elbow.dat with your favorite text editor (vim, emacs, etc.)

```
$ vim elbow.dat
```

Add the following lines within the parallel service section (which starts line 19)

```
PARALL_SERVICE:      On
  IO:                 On
  POSTPROCESS:        On
  RESTART:            On
  END_IO
PARTITION:           FACES
PARTITIONING:
  METHOD:              METIS 4
END_PARTITIONING
END_PARALL_SERVICE
```

Compare with step-1

```
$ diff ../elbow ../step-1
```

# Configure your job

## Get the job files

```
$ cp ../jobs/* .
```

## Compare with step-2

```
$ diff ../elbow ../step-2
```



# Analyzing the job file content

```
$ cat run.sh
```

Part defining job's name, outputs, tasks, time etc.

```
#!/bin/bash
#SBATCH --job-name=elbow-mpio-tutorial
#SBATCH --output=output.out
#SBATCH --error=output.err
#SBATCH --ntasks=4
#SBATCH --time=00:10:00
```

Define in which file ROMIO hints are located

```
export ROMIO_HINTS=./io_hints
```

Environment variables helping in optimizing MPI-IO performance (this configuration only suits with MN4 and should be adapted on other machines)

```
export I_MPI_EXTRA_FILESYSTEM_LIST=gpfs
export I_MPI_EXTRA_FILESYSTEM=on
```

This line runs alya

```
srun ~/tutorial-mpio/alya/Executables/unix/Alya.x elbow
```

# Analyzing io\_hints file content

```
$ cat io_hints
```

Enable the collective buffering (read and write) to improve I/O collective operations performance

```
romio_cb_read enable  
romio_cb_write enable
```

Other hints can be added but all have not been tested yet...

They may increase but decrease as well the performance.

Look at <ftp://ftp.mcs.anl.gov/pub/romio/users-guide.ps> for more info

# Launch the job

```
$ sbatch run.sh
```

# Analyzing the job result

```
$ ls -l *mpio.bin
```

You can see two types of files:

- post.mpio.bin are the post process files (processed mesh and fields)
- rst.mpio.bin are the restart files (identical to the last iteration post files)



**Barcelona  
Supercomputing  
Center**

*Centro Nacional de Supercomputación*

# CONVERT POST-PROCESS TO VTK



# Convert your files to vtk

```
$ cat post.sh
```

Part defining job's name, outputs, tasks, time etc.

```
#!/bin/bash
#SBATCH --job-name=elbow-mpio-post
#SBATCH --output=output.out
#SBATCH --error=output.err
#SBATCH --ntasks=4
#SBATCH --time=00:10:00
export ROMIO_HINTS=./io_hints
export I_MPI_EXTRA_FILESYSTEM_LIST=gpfs
export I_MPI_EXTRA_FILESYSTEM=on
```

Load the alya-mpio-tools module

```
module load alya-mpio-tools
```

Run the tool (in parallel)

```
srun mpio2vtk elbow
```

# Launch the job

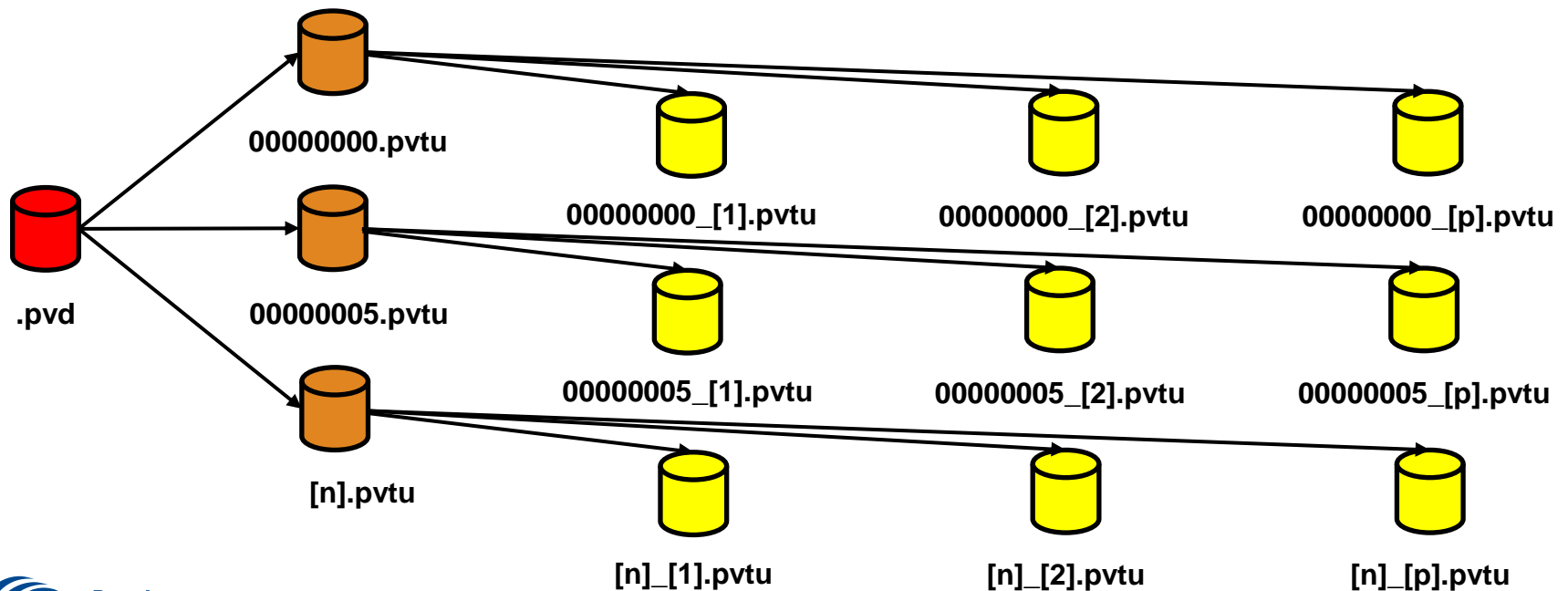
```
$ sbatch post.sh
```

# Analyzing the job result

```
$ ls -lR vtk
```

You can see three type of files:

- elbow.pvd is the master file
- elbow\_[n].pvtu is the master file for the step [n]
- elbow\_[n]\_[p].vtu is the vtk file for the step [n] and the part [p]



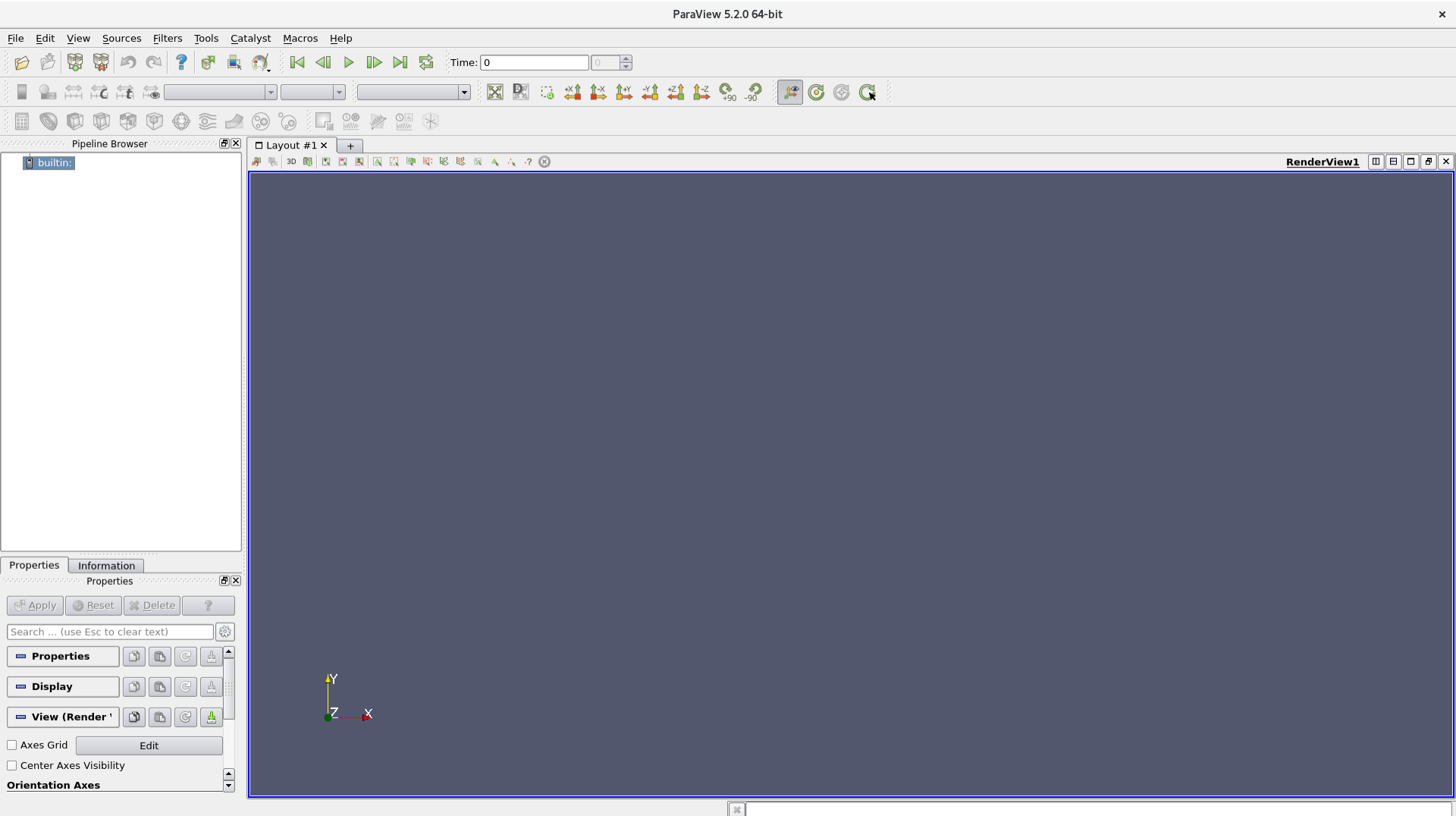
# Export only some iterations

The option `-i` enables to export only some iterations

```
srun mpio2vtk -i 0 -i 10 elbow
```

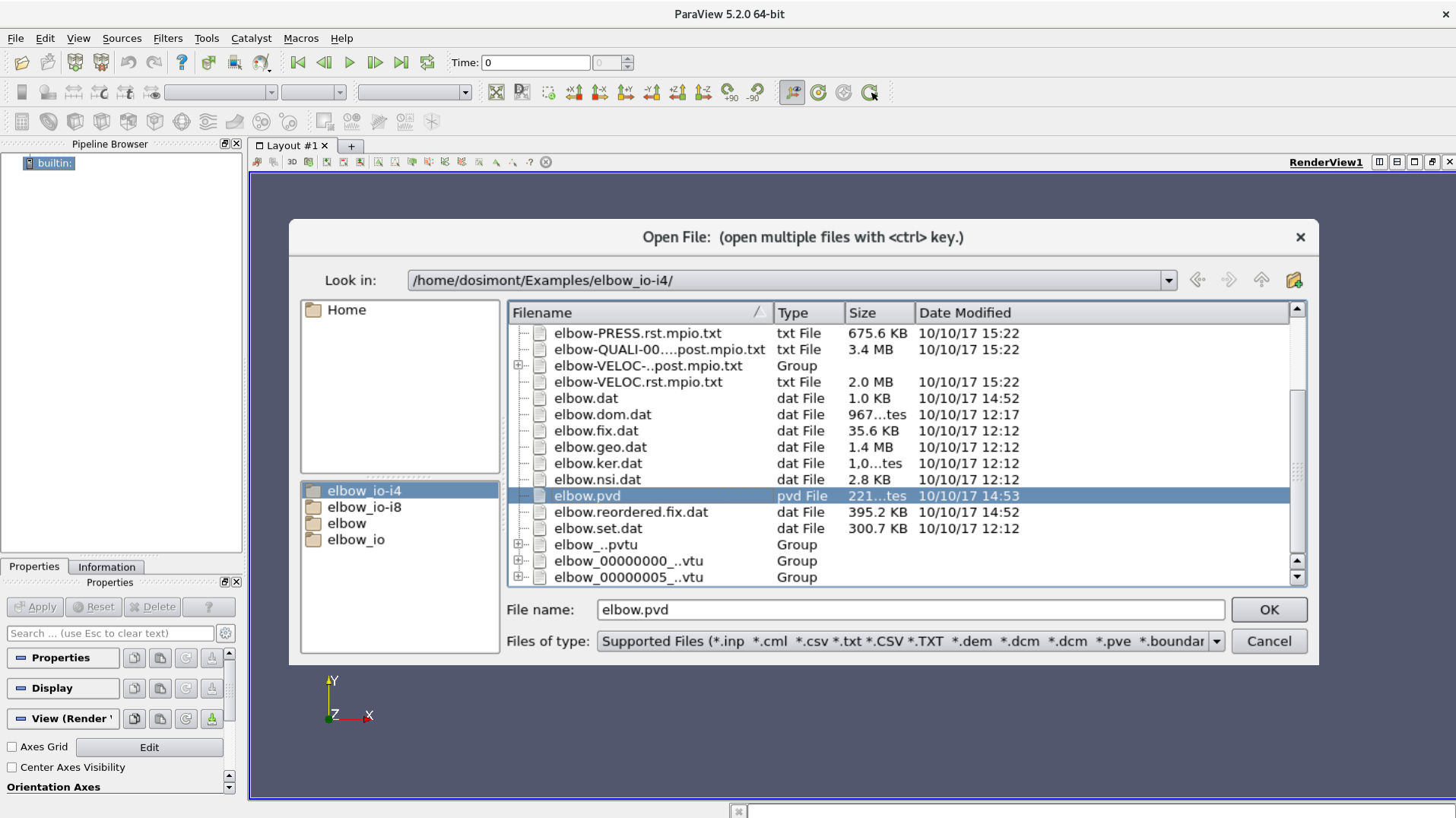
will only export the steps 0 and 10.

# Open your results in Paraview

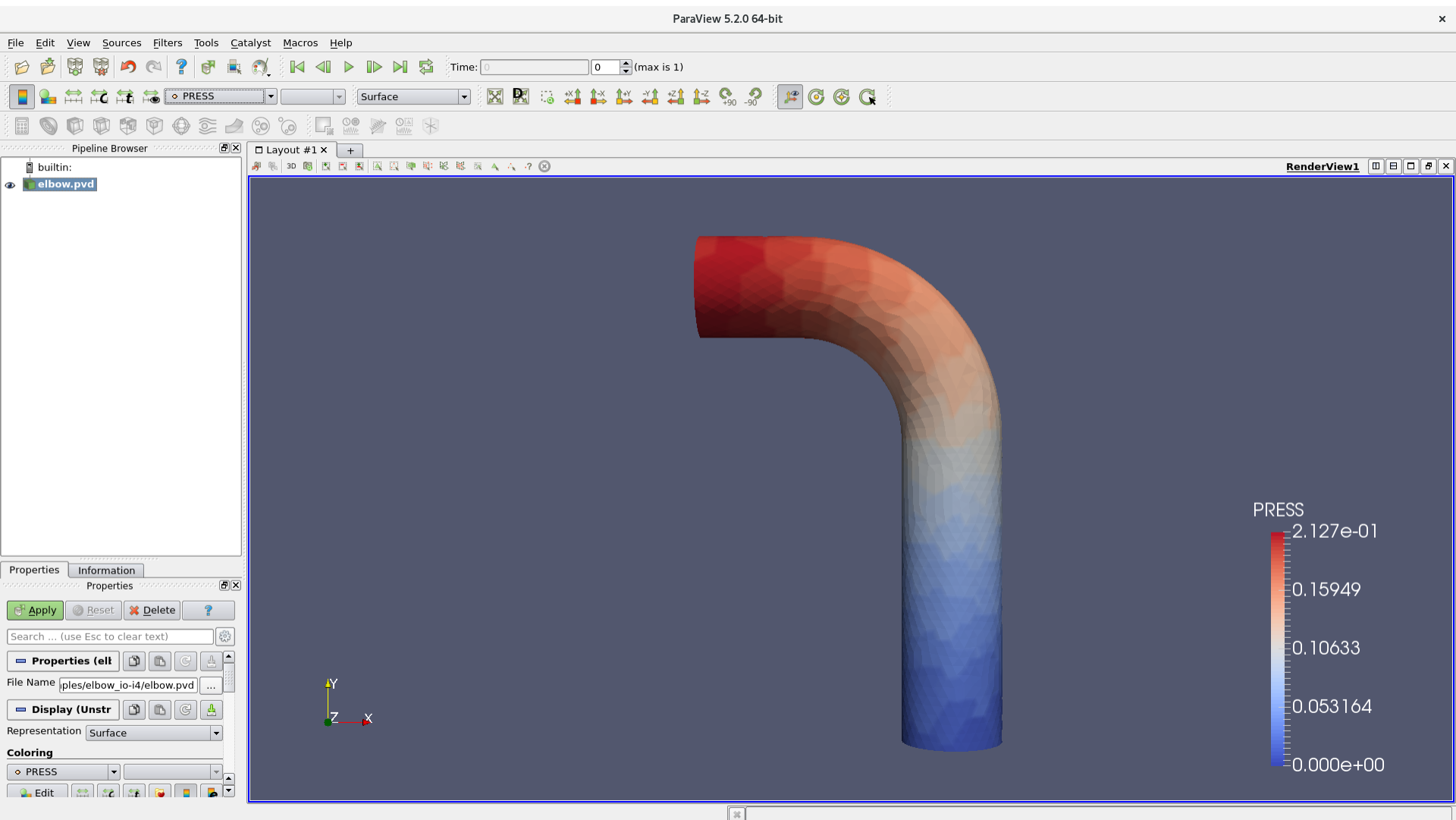




# Open your results in Paraview



# Open your results in Paraview





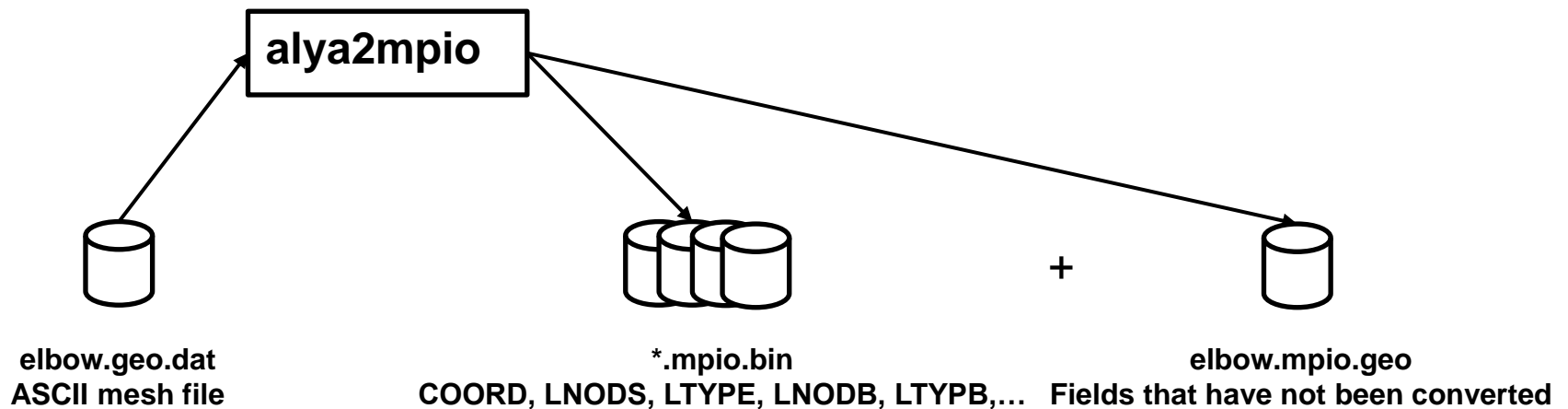
**Barcelona  
Supercomputing  
Center**

*Centro Nacional de Supercomputación*

# MESH READING

# Convert the example mesh file

- ❧ Original mesh elbow.geo.dat file is in ASCII
- ❧ MPI-IO is bad at dealing with ASCII
  - It cannot read ASCII characters directly and determine ends of lines
  - File cannot be partitioned properly to be read by several processes
- ❧ We have designed a binary format compliant with MPI-IO
  - Let's convert the ASCII geometry file using alya2mpio



# Convert the example mesh file

Let's clean the directory

```
$ ~/tutorial-mpio/alya/Utils/user/alya-clean  
$ rm -fr vtk
```

Load the alya-mpio-tools module

```
$ module load alya-mpio-tools
```

Convert the mesh files

```
$ alya2mpio elbow
```

Compare with step-3

```
$ diff ../elbow ../step-3
```

# Analyzing the conversion result

```
$ ls -l *.mpio.*
```

You can see two types of files:

- .mpio.bin are the binary mesh files
- elbow.mpio.geo is an ASCII file that contains the unconverted fields

# Update the dom.dat file

« The mesh reading MPI-IO feature works as it follows:

- First, it reads the mpio.bin mesh files in parallel
- Second, it reads the file pointed by the include in the .dom.dat domain file in sequential

```
GEOMETRY
GROUPS=100
INCLUDE  elbow.geo.dat
END_GEOMETRY
```

« In order to prevent Alya from reading the mesh twice (once in parallel, once in sequential), you have to replace the included elbow.geo.dat file by elbow.mpio.geo. This one is generated by the converter, and only contains fields that have not been exported in the binary files.



# Update the dom.dat file

Open elbow.dom.dat with your favorite text editor (vim, emacs, etc.)

```
$ vim elbow.dom.dat
```

Replace the following part (line 19-22)

```
GEOMETRY
  GROUPS=100
  INCLUDE  elbow.geo.dat
END_GEOMETRY
```

By this

```
GEOMETRY
  GROUPS=100
  INCLUDE  elbow.mpio.geo
END_GEOMETRY
```

Compare with step-4

```
$ diff ../elbow ../step-4
```

# Configure the .dat file to enable MPI-IO

Open elbow.dat with your favorite text editor (vim, emacs, etc.)

```
$ vim elbow.dat
```

Add the following lines within the parallel service section (which starts line 19)

```
PARALL_SERVICE:      On
  IO:                On
    GEOMETRY:        On
  POSTPROCESS:      On
  RESTART:          On
END_IO
PARTITION:          FACES
PARTITIONING:
  METHOD:            METIS 4
END_PARTITIONING
END_PARALL_SERVICE
```

Compare with step-5

```
$ diff ../elbow ../step-5
```

# Launch the job

```
$ sbatch run.sh
```



**Barcelona  
Supercomputing  
Center**

*Centro Nacional de Supercomputación*

**Thank you!**

For further information please contact  
[damien.dosimont@bsc.es](mailto:damien.dosimont@bsc.es)