CERFACS

ONERA
THE FRENCH AEROSPACE LAB

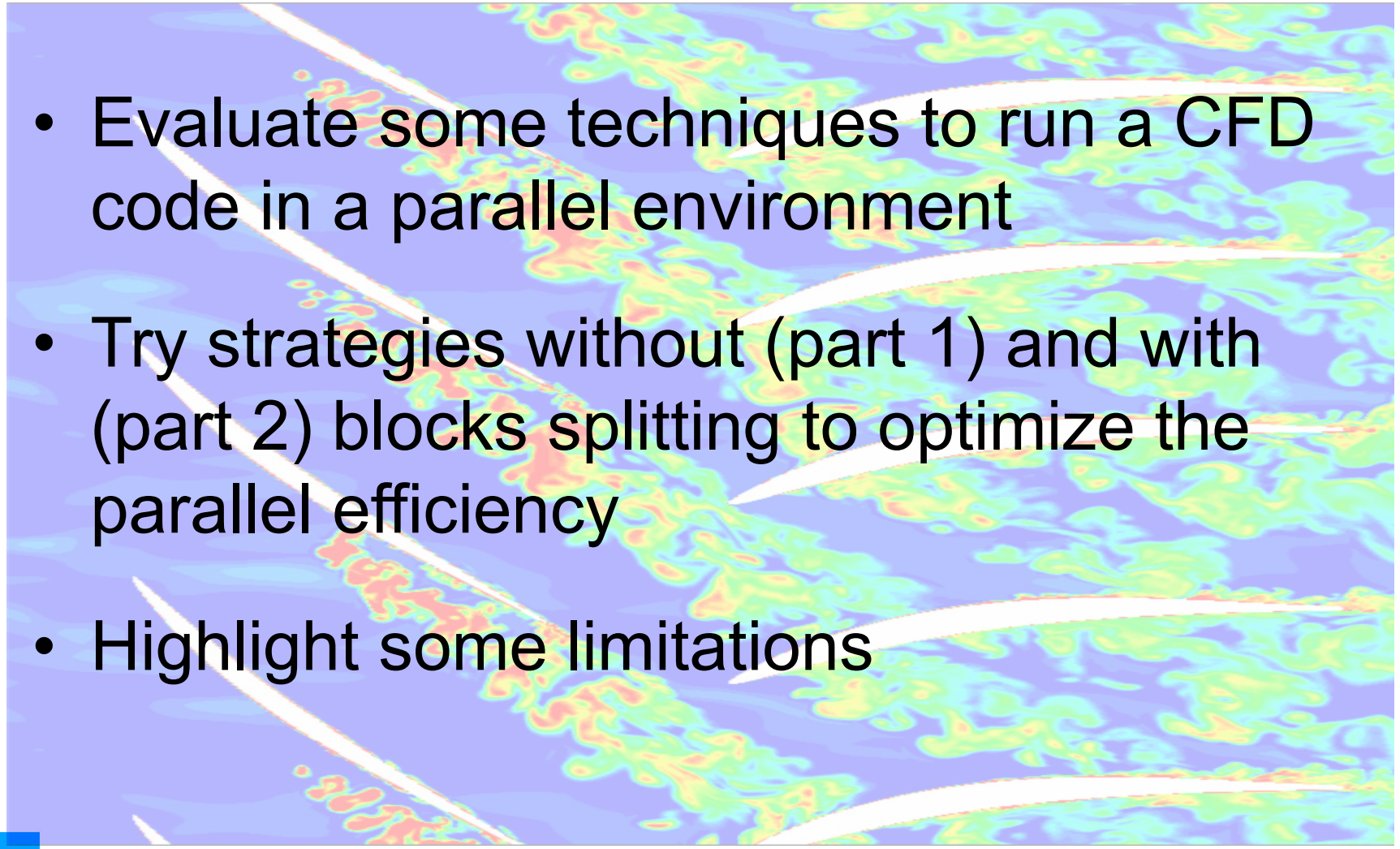
Hands-on: HPC with *elsA*

N. Gourdain, F. Sicot, CERFACS, CFD team

M. Gazaix, ONERA, DSNA Dpt.

PRACE Autumn School 2013 - Industry Oriented HPC Simulations, September 21-27, University of Ljubljana, Faculty of Mechanical Engineering, Ljubljana, Slovenia

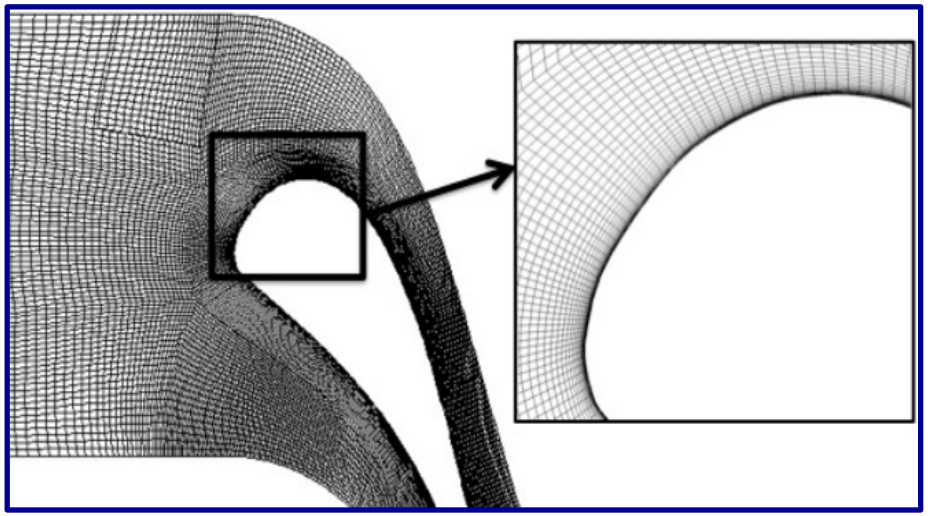
Objective

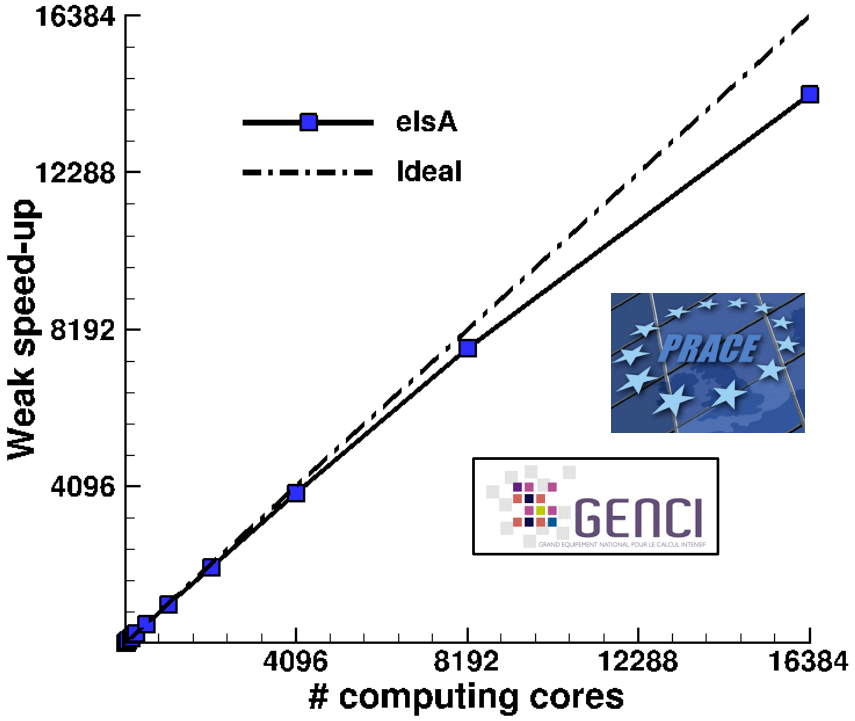


- Evaluate some techniques to run a CFD code in a parallel environment
- Try strategies without (part 1) and with (part 2) blocks splitting to optimize the parallel efficiency
- Highlight some limitations

→ a structured code for aerodynamics (ensemble logiciel pour la simulation en Aérodynamique)

- Mainly developed by ONERA [1] and CERFACS
- Industrial use (AIRBUS, EUROCOPTER, SAFRAN, EDF, etc.)
- But a research tool too (ONERA, CERFACS, ISAE, ECL)
- Compressible finite volume flow solver with multi-block structured meshes
- **Massively parallel** capabilities (MPI) [2]
- **(U)RANS, DES and LES** approaches





[1] Cambier and Veuillot, AIAA, 2008

[2] Gourdain *et al.* J. Comp. Sc. Discovery, 2009

ONERA THE FRENCH AEROSPACE LAB

Let's start with some boring things

Some information about the code… sorry!

elsA is not HMI*-oriented:

- **Industries build their own interface (the code is plug into a design tool chain)**
- **Students can be a bit disappointed by this approach compared to commercial products (ANSYS, NUMECA)**
- **But it is the code used by the aeronautic industry for the design of aircraft, gas turbines, etc.**

* Human Machine Interface

CERFACS ONERA THE FRENCH AEROSPACE LAB

→ Log files (parallel computing)

When running on several processors, elsA automatically creates log files called

```
elsA_MPI_Pid_<PID>_N_<MPI rank>
```

where

`PID` is the process identification number. It changes at each run.

`MPI rank` is the rank of the MPI process ranging from 0 to #proc - 1

These log files contain the same data as in sequential mode except that each processor only reads its allocated meshes.

In order to only create the rank 0 log file, set the environment variable

```
ELSA_MPI_LOG_FILES=OFF
```

→ Log files (parallel computing)

```
================================================================
----------------- Heap Memory Usage Summary -----------------
================================================================

----------------- 1- Local to processor ----------------------
----------------------------------------------------------------
================================================================
Maximum total allocated memory   :     45364288 bytes
----------------------------------------------------------------

----------------------------------------------------------------
----------- 2- Global reduction (   4 processor config) -------
----------------------------------------------------------------
Global  total allocated memory   :    200142712 bytes
================================================================
----------------- MPI Message Summary       ------------------
----------------------------------------------------------------

----------------------------------------------------------------
----------------- 1- Local to processor ----------------------
----------------------------------------------------------------
      Number of MPI send        = 3.9323200e+05
      Size  of exchanged buffer = 1.0334268e+10
Cell      Msg Nb : 3.9322800e+05  Cell      Msg size = 1.0334120e+10
Interface Msg Nb : 4.0000000e+00  Interface Msg size = 1.4873600e+05
Node      Msg Nb : 0.0000000e+00  Node      Msg size = 0.0000000e+00
================================================================

----------------------------------------------------------------
----------- 2- Global reduction (   4 processor config) -------
----------------------------------------------------------------
Total Number of MPI buffer send  : 1.9661600e+06
Total Size   of exchanged buffer : 4.3633579e+10 bytes

============== End MPI Message info ==========================

# elsA : normal run termination (0)

----------------------------------------------------------------
This task (proc : 0) took 2.1318939e+03 seconds  (resolution = 1.000000e-06 s)
----------------------------------------------------------------
```

Max. memory (local proc.)

Max. memory (all procs)

Size of MPI messages (local procs.)

Size of MPI messages (all procs.)

Total time of the task

→ Processor assignation (block2proc dictionary)

A convenient way to assign a given block to a given processor is to use a Python dictionary:

```
conf.set('mpi_block2proc', 'script_block2proc')
```

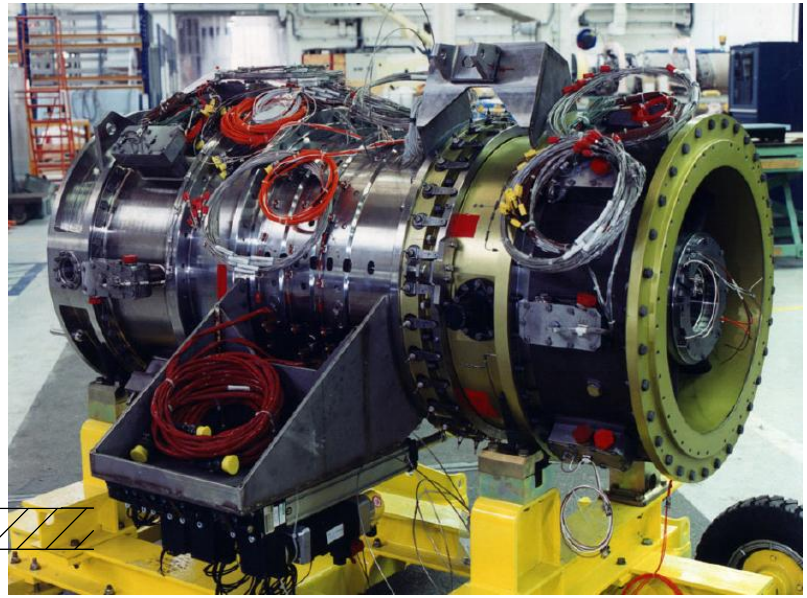
where `'script_block2proc'` actually refers to a python script named `script_block2proc.py` that contains a specific dictionary:

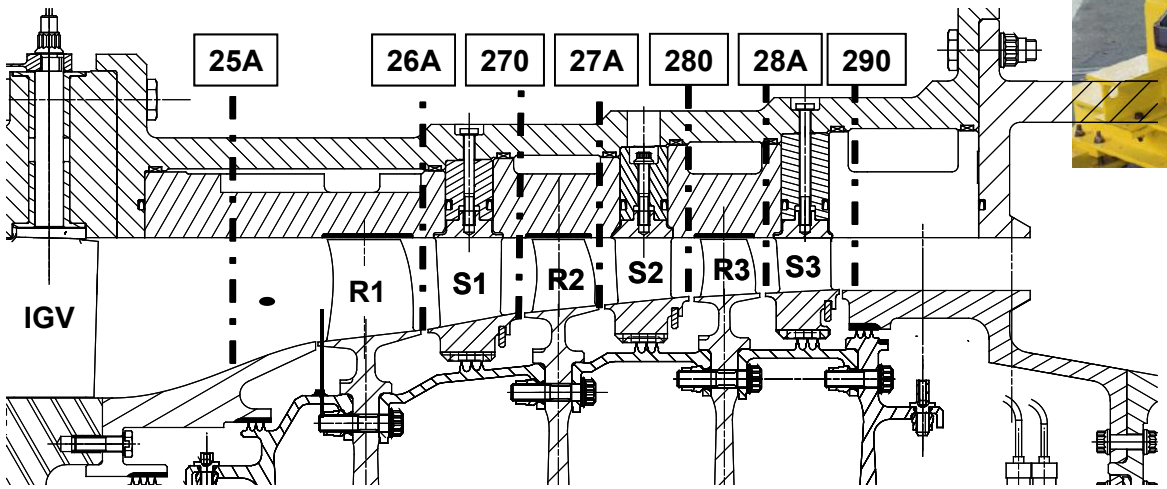
```
dict_block2proc = {
   0 :   25,
   1 :    0,
   2 :   26,
   3 :   30,
   4 :   31,
   5 :   41,
...}
```

The keys correspond to the block numbers (starting from 0) and the values correspond to the MPI ranks (starting from 0).

Let's choose a guinea pig?

- High pressure compressor core designed by SNECMA
- Representative of a modern civil application (CFM56)





Gourdain *et al.*, IJHPCA, 2010

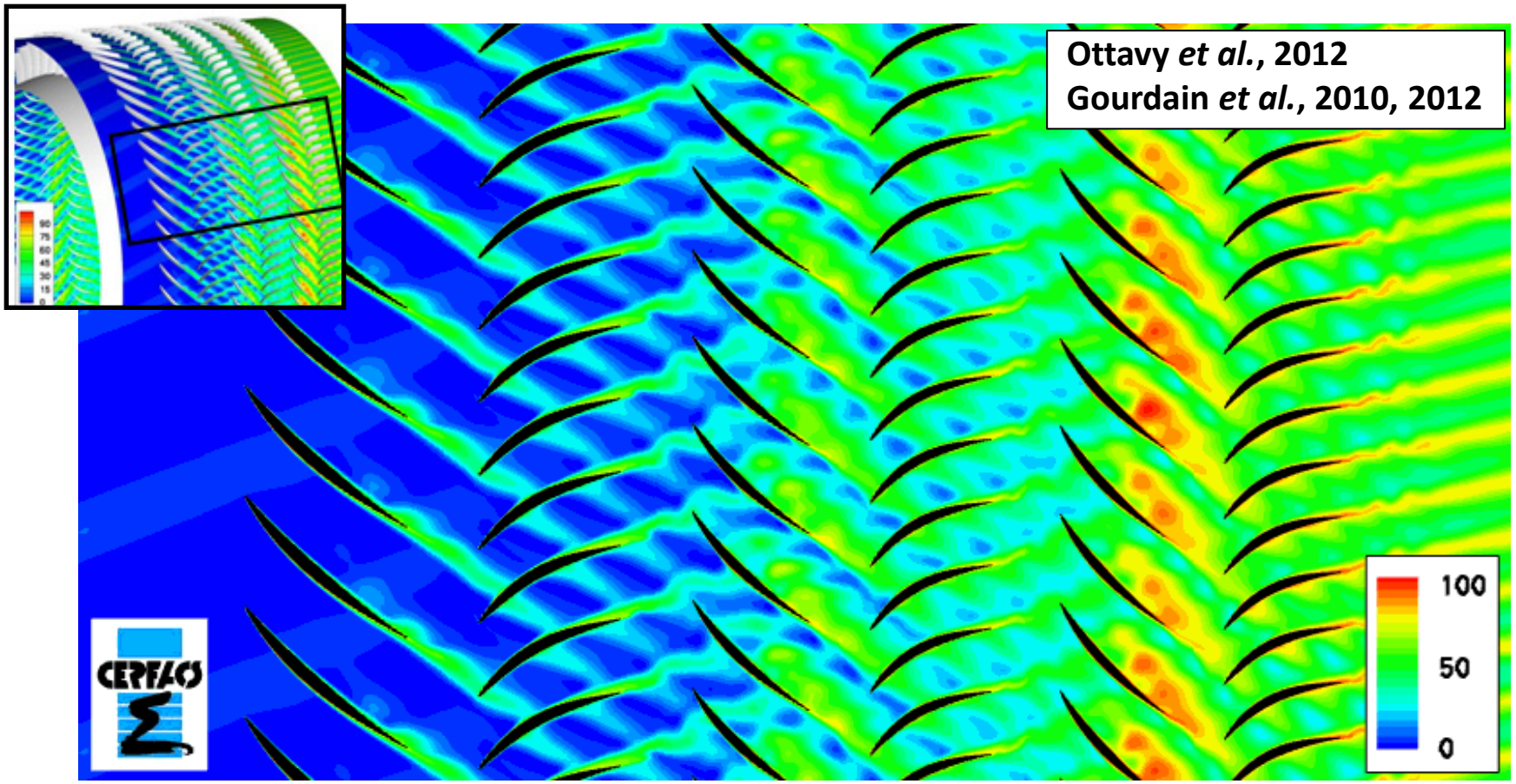
Gourdain et al.,J. Turbomach., 2012

Ottavy *et al.*, J. Prop. And Power, 2012

CERFACS

ONERA
THE FRENCH AEROSPACE LAB

Let's choose a guinea pig?

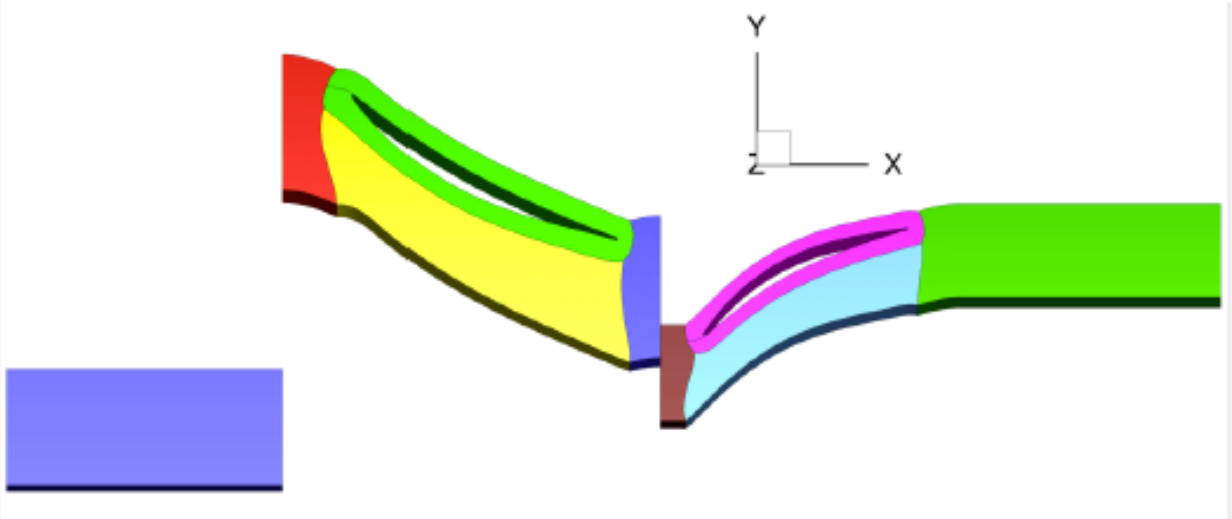


Entropy flow field, nominal operating conditions

ONERA
THE FRENCH AEROSPACE LAB

Now it's time to work

Consider the following 9-blocks turbomachinery configuration:



What is the best distribution of the blocks among the processors?

Simplistic answer: use 9 processors! Sadly, it is efficient only if the blocks all have the same size. This is not the case here. One has to find a good load balance (i.e. actual number of cells/proc should be as close as possible to ideal number).

CERFACS

ONERA
THE FRENCH AEROSPACE LAB

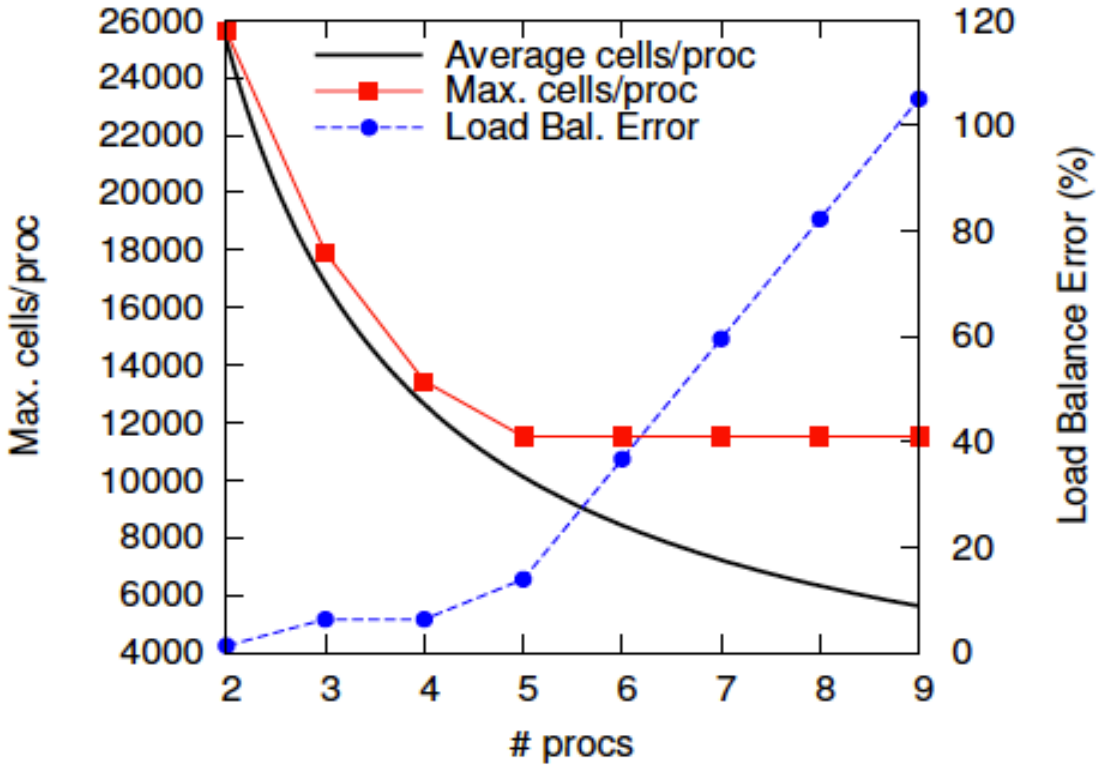
STEP 1: no splitting

- **Go into directory TP1, then into Work (cd TP1/Work)**
- **Edit the file “main.py” (this file contains all the numerical parameters)**
  - **select the number of procs. you want: NB_PROC=[1;8]**
- **Edit the file “MyJob” and set the corresponding number of procs. (1 to 8)**
- **Run the simulation and get the time from the *elsA* log file**

STEP 1: no splitting

Perform all the load balance distribution ranging from 2 to 9 processors.

What is the best number of processors for this configuration?



It makes no sense to use more than 5 processors.

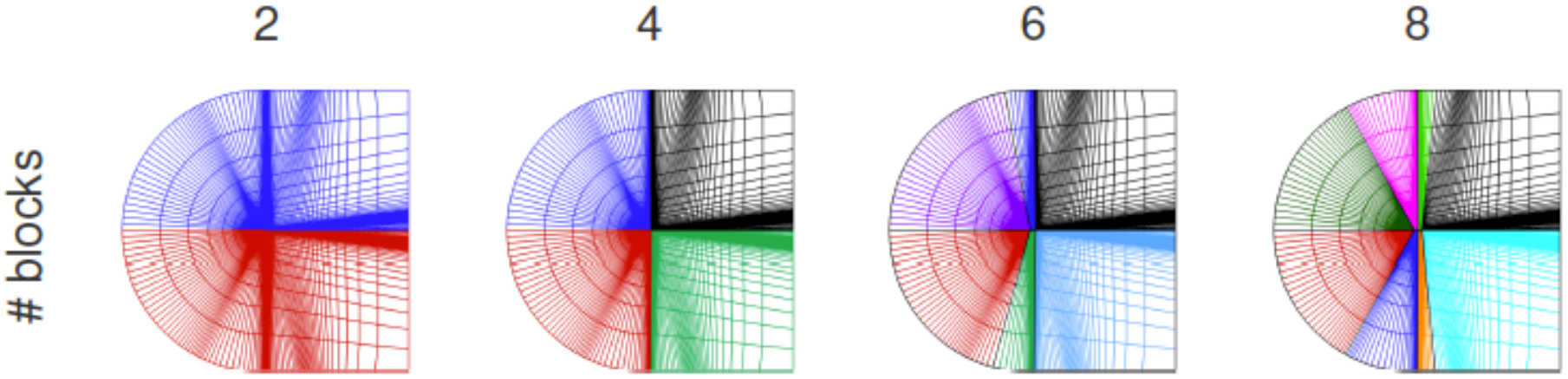
STEP 1: no splitting

Which are your conclusions?

- **What is the ideal number of processors?**
- **What can you do to improve the max. number of processors?**

Splitting tool

It is possible to split the computational domain into more blocks in order to run on more processors:



The process is purely sequential and therefore runs on a single processor:

1. read just the size of all the blocks
2. run partitioning algorithm
3. split the blocks one after the other

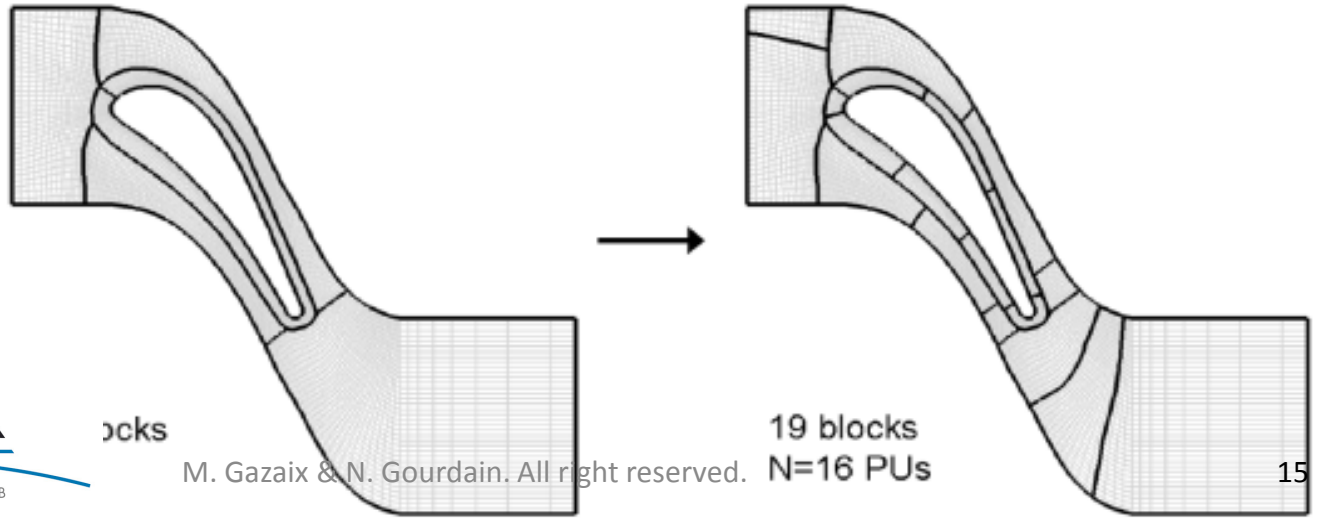
The memory should be large enough to handle the biggest block. As the original mesh is already multi-blocks due to meshing constraints, this is never an issue.

Mesh partitioning

Two mesh partitioning algorithms are available in elsA that only consider geometric information (no information from the communication graph is used):

The Recursive Edge Bisection algorithm first splits the largest block on its longest edge, and repat until the number of blocks equals the desired number of CPUs indicated by the user

The Greedy Algorithm loops over blocks looking for the largest one (after splitting when necessary) and allocating it to the PU with the smaller number of cells until all blocks are allocated.



STEP 2: block splitting tool

- **Go into directory TP2, then into Work (cd TP1/Work)**
- **Edit the file “main.py” (this file contains all the numerical parameters)**
  - **select the number of procs. you want: NB_PROC=[8;32]**
  - **Select the splitting algorithm “load_balance_algo”**

STEP 2: block splitting tool

The choice of split algorithm is made by the `'load_balance_algo'` key:

`'no_split'` only load balance

`'greedy'` Greedy algorithm (default)

`'greedy_ijk'` not yet documented (sic!), a better Greedy algorithm?

`'recursive_bisection'` Recursive Edge Bisection algorithm

- **Try "greedy" or "recursive_bisection"**

STEP 2: block splitting tool

First the compute() method must NOT be called. Then 3 methods can be used:

1. load_balance() takes the number of processors in argument, launches the splitting algorithm
2. print_script () prints the new scripts with the new topology
3. split_init_file () splits the initial files (meshes, initial condition, boundary...) and writes them

```
# conf.compute()

nproc = 4   # number of processors

conf.load_balance(nproc)   # load balance
conf.print_script()        # print new scripts
conf.split_init_file()     # write new meshes
```

STEP 2: block splitting tool

The load balancing can be launched as a regular elsA computation:

```
> elsa -f split.py | tee out_split
```

The algorithm tells how the blocks are split:

```
-----------------------------------------------------------
Split Algo Info : split (oldId =   0, DIR_I, pos = 105)
     # --> newId =   2 ( 105 X   85 X    2)
     # --> oldId =   0 ( 105 X   85 X    2)
Split Algo Info : split (oldId =   1, DIR_I, pos = 105)
     # --> newId =   3 ( 105 X   85 X    2)
     # --> oldId =   1 ( 105 X   85 X    2)
-----------------------------------------------------------
```

- **Run the script "run_split.sh"**

STEP 2: block splitting tool

The splitter generates two kinds of scripts:

1. `script_*_u.py` are written in current Python-elsA syntax
2. same names but without the `_u` suffix: old and deprecated syntax, can be ignored.

The main scripts is `script_cfd_u.py`. Only the singletons are declared in this file. It imports other new scripts:

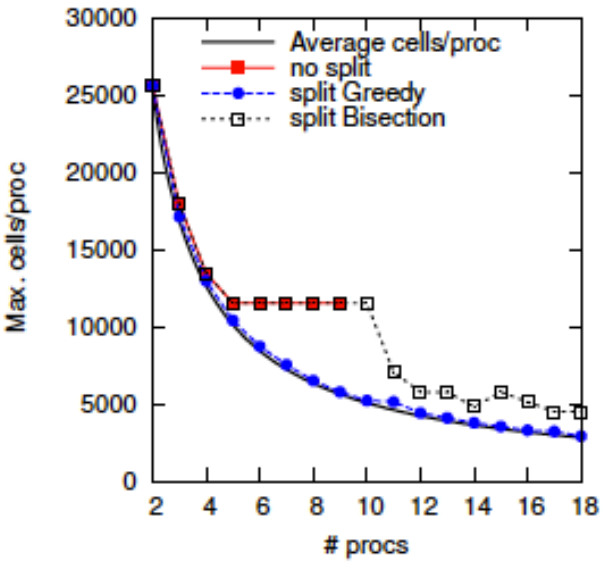
- `script_bndphys_u.py`: physical boundary type declaration
- `script_topology.py`: physical and join boundary declaration
- `script_extract.py`: extraction declaration

There is a `script_block_u.py` file, but it is not imported by default. Actually, the blocks are created automatically.

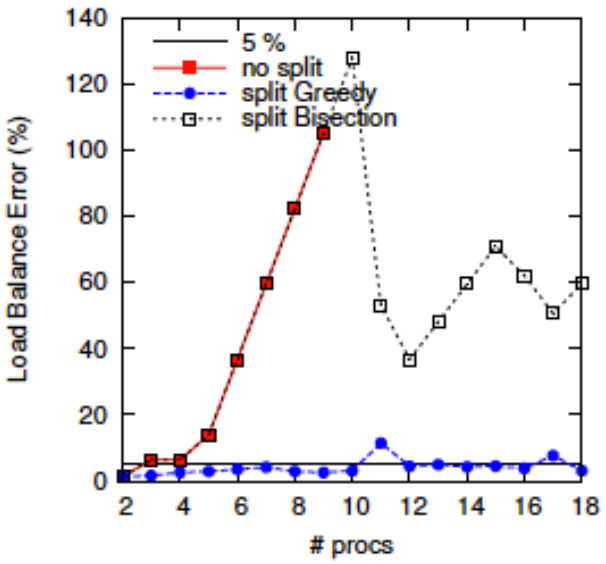
STEP 2: block splitting tool

- Go into directory TP2, then into Work (cd TP1/Work)
- Edit the file “main.py” (this file contains all the numerical parameters)
  - select the number of procs. you want: NB_PROC=[8;32]
  - Select the splitting algorithm “load_balance_algo”
- **Edit the file “MyJob” and set the corresponding number of procs. (8 to 32)**
- **Run the simulation and get the time from the *elsA* log file**

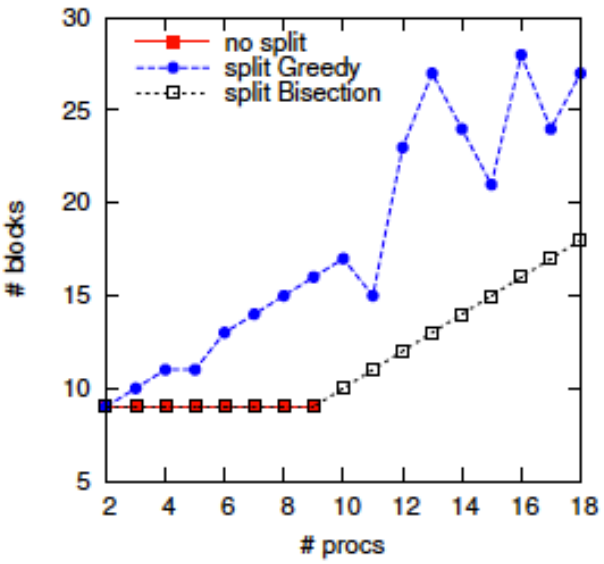
STEP 2: block splitting tool



(a) Max. cells/proc



(b) Load balance error



(c) Number of blocks

The Greedy algorithm generates more blocks than requested processors. The load balance error remains (almost always) under 5 %.

The Recursive Bisection algorithm generates as many blocks as requested procs: it starts splitting above 9 procs. The load balance error remains high.

CERFACS

ONERA
THE FRENCH AEROSPACE LAB

More things, if we have time…

Sometimes, one wants to run on twice or three times less processors because of cluster constraints for instance.

The `ELSA_MPI_MODULO_PROC` environment variable allow to divide the total number of processors.

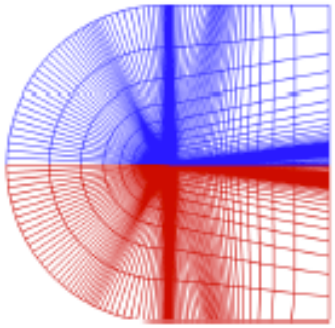
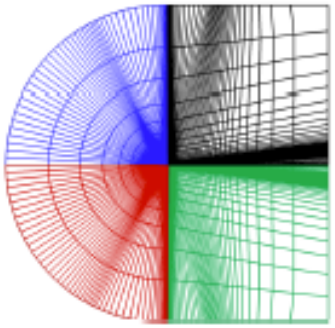
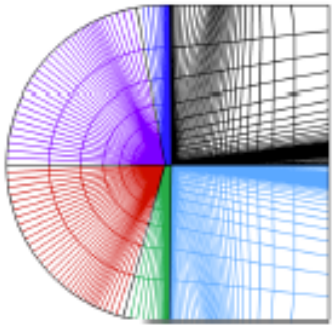
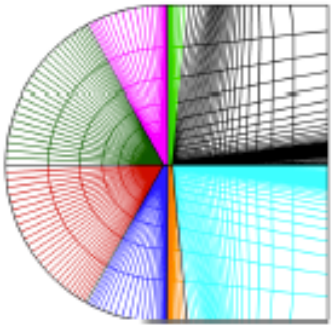
For instance if the configuration is split to run on 24 processors, setting `ELSA_MPI_MODULO_PROC=3` will automatically map the blocks to a 8-processors run.

CERFACS

ONERA
THE FRENCH AEROSPACE LAB

Some side effects

Splitting the configuration increases the number of nodes and ghost cells:

	2	4	6	8
# blocks				
# nodes	35,530 ref	35,700 +0.4 %	35,870 +0.9 %	36,040 +1.4 %
# DoF	74,624 ref	76,032 +1.8 %	77,440 +3,8 %	78,848 +5.6 %

CERFACS

ONERA
THE FRENCH AEROSPACE LAB

Conclusion

All techniques presented in this hands-on are the basis to run much more complex simulations, such as the flow simulation below (10^9 grid points, 1,024 processors)