University of Liège

Department of Aerospace and Mechanical Engineering

```
#pragma omp parallel for num_threads(nbt)
for (int i=0; i<m; i++)
{
```

Numerical Simulation of Nonlinear Mechanical Problems using Metafor

Romain BOMAN

Senior Assistant

```
int idx2=0;
for(int nbt=trange.getMin(); nbt<=trange.getMax(); nbt+=trange.getStep())
{
    idx2++;
    double tstart = omp_get_wtime();
    test.execute(nbt);
    double tstop = omp_get_wtime();
```

CECI Meeting - 16th May 2014

Outline

1. Our in-house FE code : Metafor
2. Libraries and tools
3. Numerical examples
4. Future works

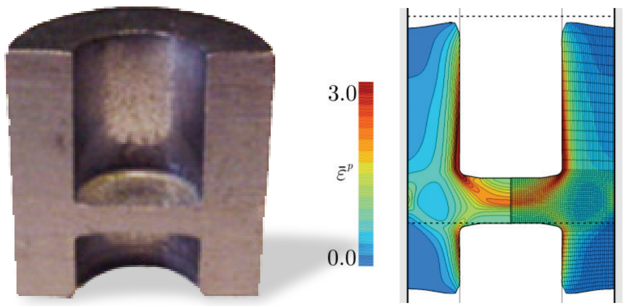
Outline

1. **Our in-house FE code : Metafor**
2. Libraries and tools
3. Numerical examples
4. Future works

Metafor

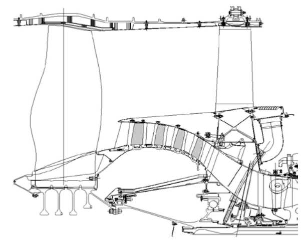
Implicit Finite Element code for the simulation of solids submitted to large deformations

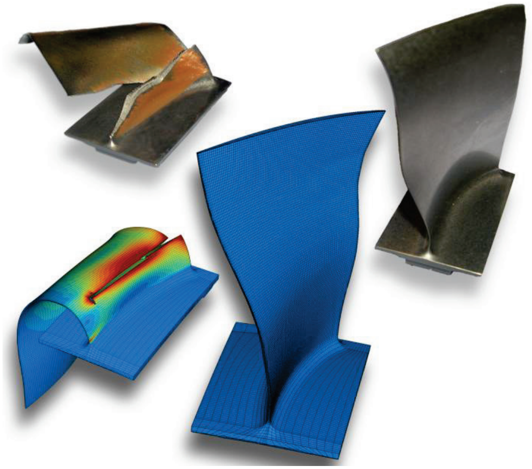
Metal Forming Applications



- ALE formalism, remeshing,
- thermomechanical time integration schemes

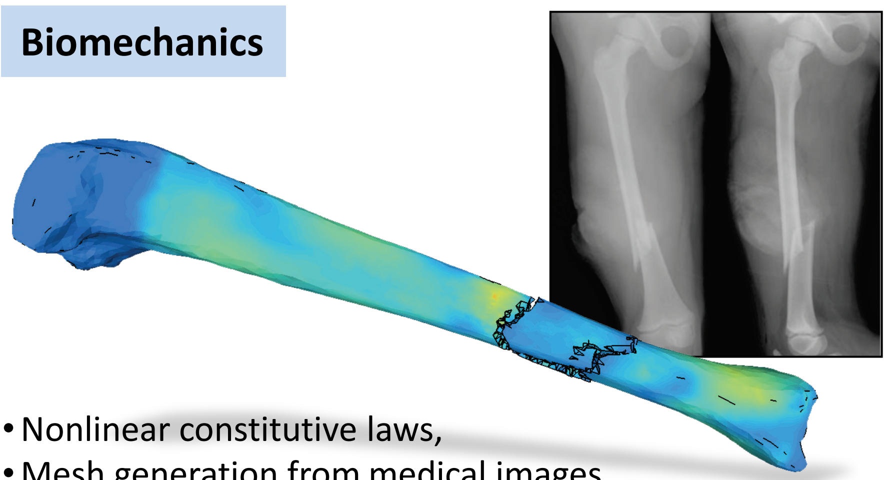
Crash / Impact





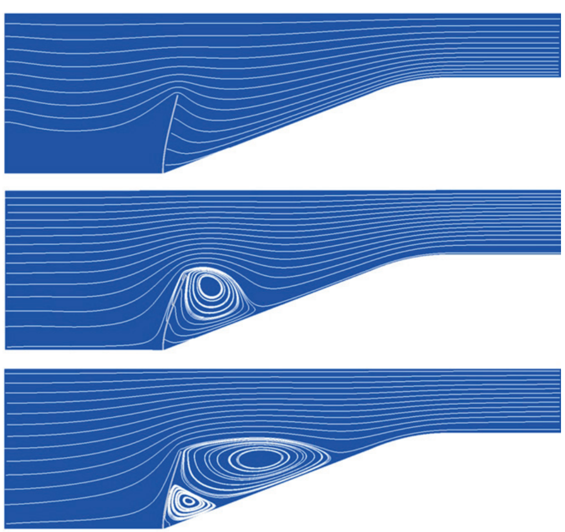
- fracture modelling,
- crack propagation,
- contact algorithms

Biomechanics



- Nonlinear constitutive laws,
- Mesh generation from medical images
- Enhanced finite elements

Fluid Structure Interaction



- Fluid elements
- Monolothic scheme

Metafor

Version history

- 1992 : Version 1 (Fortran):
  J.-P. Ponthot's PhD thesis
- 1999 : As many versions as researchers
- 2000 : Rewritten as an Oofelie solver (C++)
- Today : Still one single version for 11 developers

How big is it?

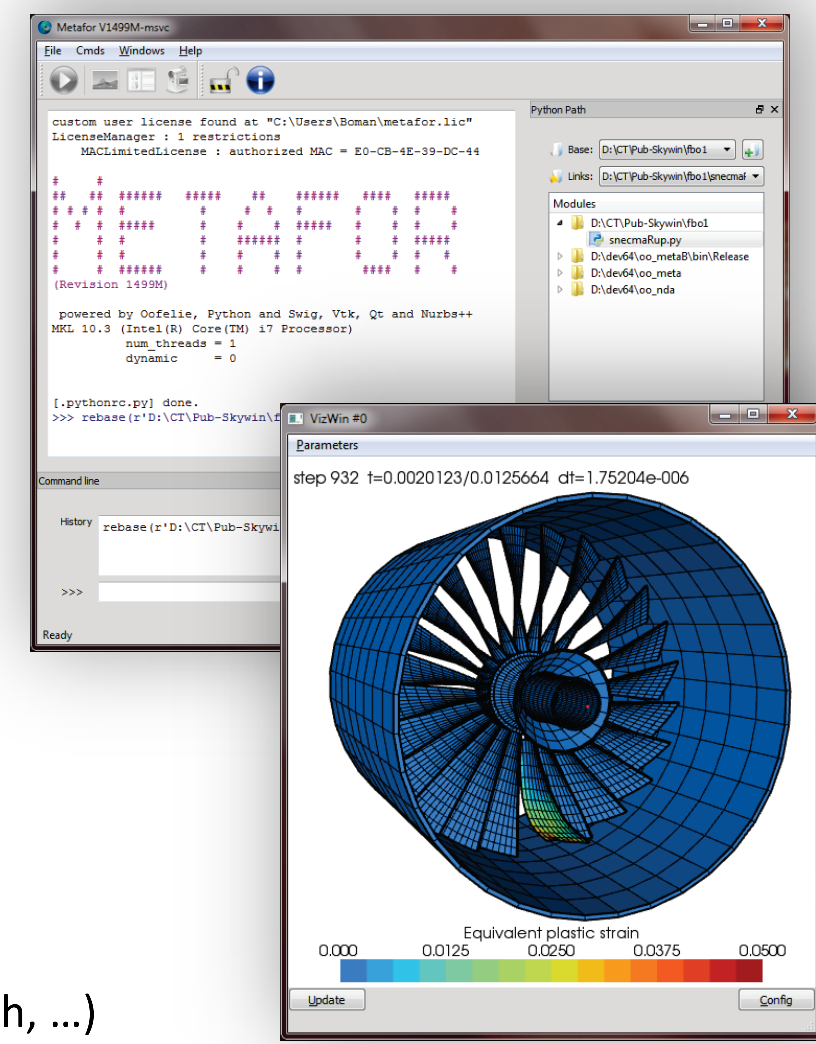
- ~1500 C++ classes
- ~300k lines of code
- 52 libraries (.dll/.so)
- ~2200 FE models in the test suite

Operating Systems

Windows, Linux, MacOS

Users

- Researchers (PhD, FYP), students (FE course)
- Companies (ArcelorMittal, Techspace Aero, GDTech, ...)



Outline

1. Our in-house FE code : Metafor
2. **Libraries and tools**
3. Numerical examples
4. Future works

Libraries and tools

SUBVERSION

Version management

CMake

Makefiles/project generator

vtk

3D display of meshes

Qt

Widgets, threads (GUI)

python™

Interpreter & user subroutines

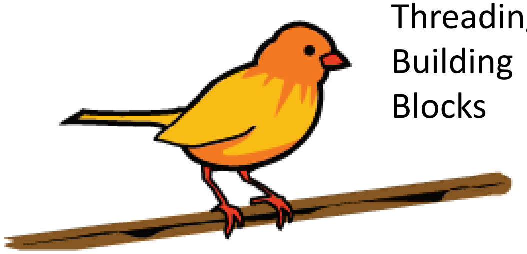
SWIG

Python interface generator

intel®

Compiler, BLAS

Parallel linear solver



Threading Building Blocks

Parallelism (shared memory)

Libraries and tools

Version management

Makefiles/project generator

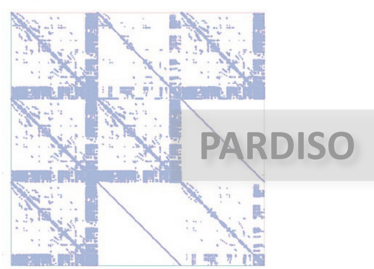
3D display of meshes

Widgets, threads (GUI)

Interpreter & user subroutines

Python interface generator

Compiler, BLAS



Parallel linear solver

Threading Building Blocks

Parallelism (shared memory)

Why TBB?

Shared Memory Parallel Programming with OpenMP

Example of a loop over the set of Finite Elements…

```
class Element;                // a Finite Element
std::vector<Element*> els;    // a set of Finite Elements
```

ORIGINAL C++ LOOP

```
std::for_each(els.begin(), els.end(),
              [&](Element *e)
{
    e->getForces();
});
```

OpenMP

OPENMP LOOP

```
int nbelm = els.size();
#pragma omp parallel for
for(int i=0; i<nbelm; ++i)
{
    Element *e = els[i];
    e->getForces();
}
```

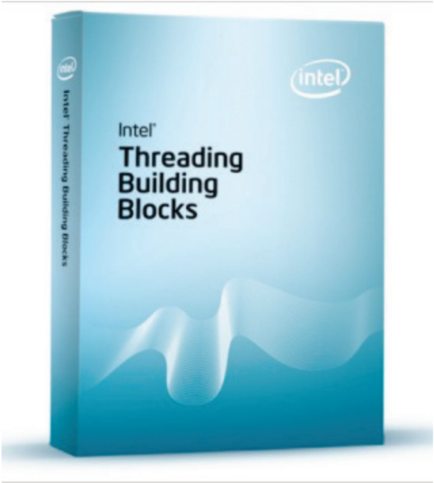
- C++ iterators are forbidden
- `size_t` cannot be used as a loop counter
- Calling a function in a `for` statement is not allowed
- Possibly bad load balancing
- Nested parallelism difficult to handle.

Back to 1992!

Why TBB?

Intel Threading Building Blocks

- High Level Parallel programming library (SMP)
- Open Source!
- Highly portable (msvc, gcc, even old versions)
- Object oriented – similar to STL – `tbb` namespace)
- Task-oriented (instead of threads)
- Thread-safe C++ containers
- Efficient overloaded memory allocators (`new`, `delete`)
- Takes into account OpenMP libraries (Pardiso solver)



ORIGINAL C++ LOOP

```
std::for_each(els.begin(), els.end(),
                    [&](Element *e)
{
     e->getForces();
});
```

TBB LOOP

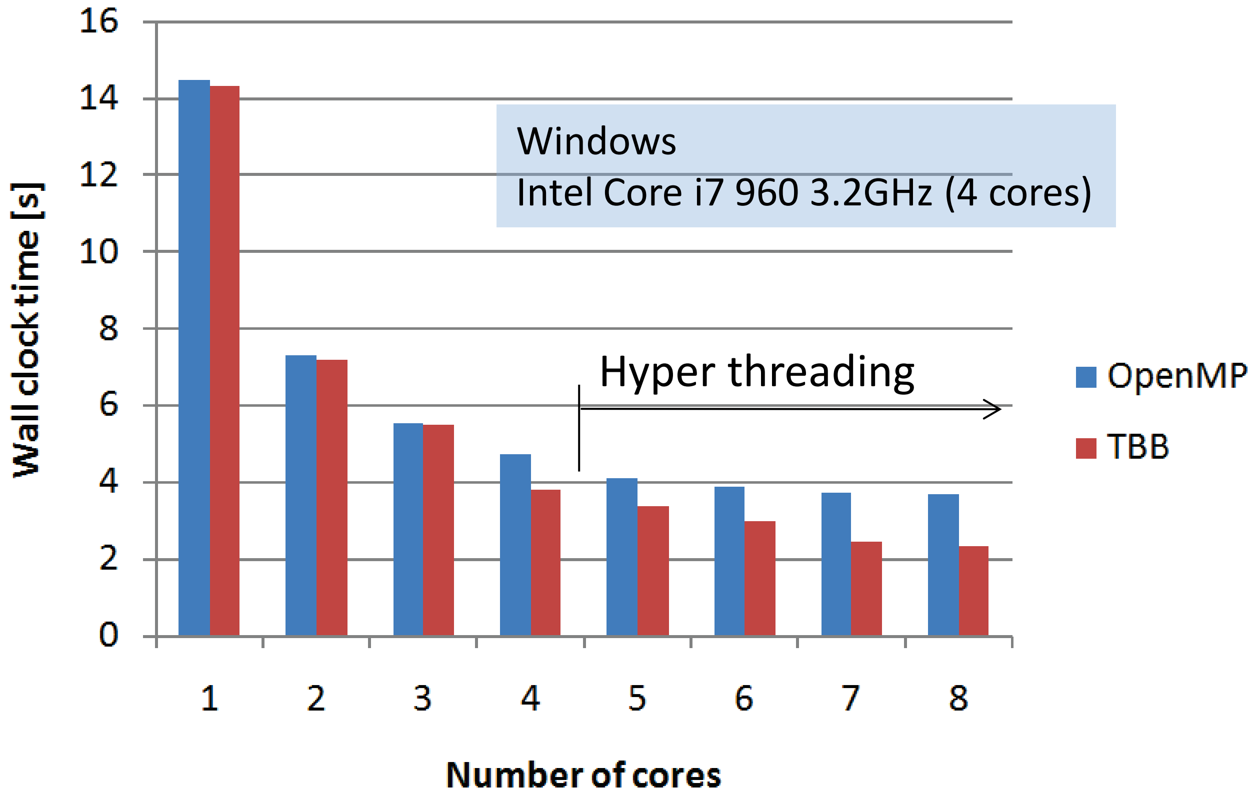
```
tbb::parallel_do(els.begin(), els.end(),
                    [&](Element *e)
{
     e->getForces();
});
```

Same syntax as modern C++

Why TBB?

... and it is more **efficient** and **reliable** than OpenMP

Example: Matrix-vector product : a = B c (size = *n*)



- *n*=10000
  ➔ size(B)=763 Mb
- multiplication performed 100x

Libraries and tools

SUBVERSION

Version management

CMake

Makefiles/project generator

vtk

3D display of meshes

Qt

Widgets, threads (GUI)

python™

Interpreter & user subroutines

SWIG

Python inter
generator

intel

Compiler, BLAS

Parallel linear solver

Threading Building Blocks

Parallelism (shared memory)

Python interface

The main objects of Metafor are available through a python interface for 2 main reasons

- **Input files are written in python**
  - **Less code**: no complicated home-made parser required
  - **Full language**: use of loops, conditional statements, objects in input files
  - **Extensibility**: calls to external libraries (Qt, wxWidgets, numpy, …)
  - **Glue language**: calls to external codes (gmsh, SAMCEF, Abaqus, matlab, etc.)
  - **Safety**: errors are correctly handled (even C++ exceptions!)

- **Extension of the code (“user subroutines”)**
  - A lot of C++ classes (boundary conditions, postprocessing commands, geometrical entities, materials, meshers, etc.) can be derived by the user in Python in the input file.

Python interface

Python scripts as input files

One Python class is **automatically** created by SWIG for each C++ class

materials.i

```
%module materials
%{
#include "ElasticMat.h"
%}

%include "ElasticMat.h"
```

SWIG →

materials.pyd (Compiled Python module)

materials.py (Shadow classes)

INPUT SCRIPT

```
from materials import *

mat = ElasticMat(E, nu)

model.setMaterial(mat)
model.run()
```

Adding one new material class requires to add **only two lines** in the input file of SWIG (`material.i`) to make it available in Python!

Python interface

Python inheritance of C++ classes : "user subroutines"

SWIG can generate the (huge and complex) code required to derive a C++ class in Python!

C++ ELASTIC MATERIAL

```
class ElasticMat
{
public:
   virtual
   T computeStress(T &strain);
};
```

PYTHON ELASTICPLASTIC MATERIAL

```
from materials import *

class ElasticPlasticMat(ElasticMat):

   def computeStress(self, strain):
      # compute the stresses
      # from the strains
      # using python cmds here…
      return stress
```

This python code will be called from the compiled C++ code!

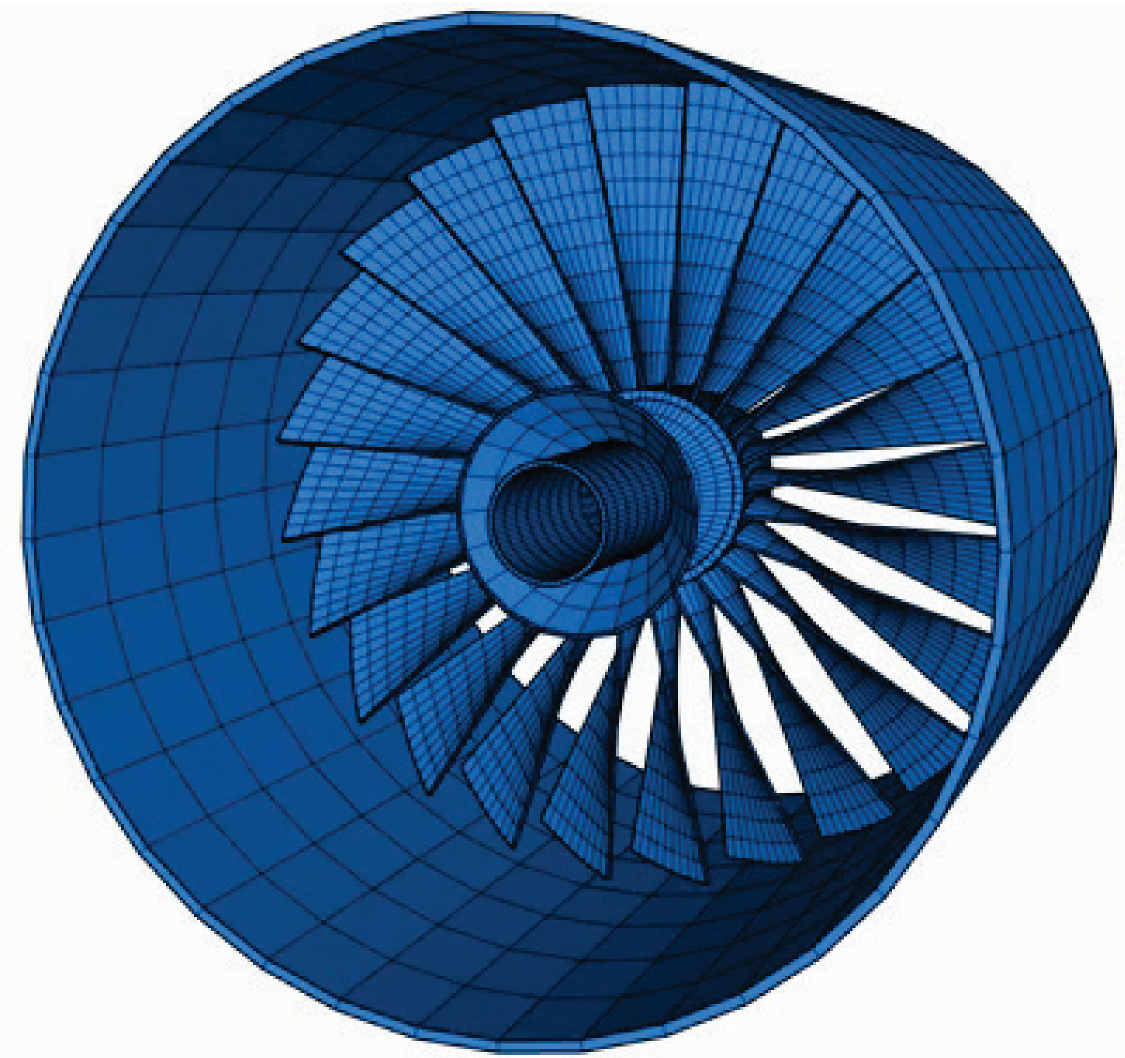
A new material law is available without any compiler!

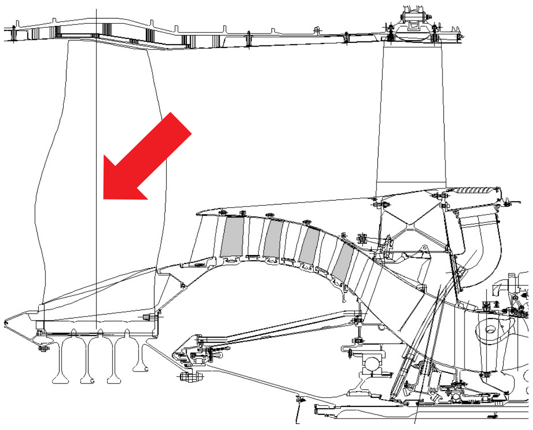
Very useful for students (Final Year Projects)

Outline

1. Our in-house FE code : Metafor
2. Libraries and tools
3. **Numerical examples**
4. Future works

Fan Blade containment test

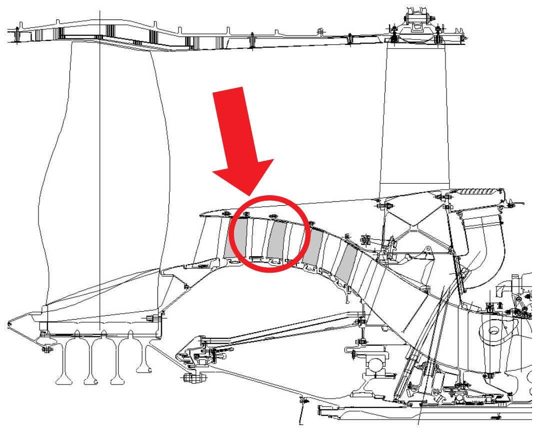




Snecma
SAFRAN Group

- Numerical simulation of an engine validation test (FBO : Fan Blade Out)
- Implicit algorithm (Chung Hulbert)
- Fixed bearing & moving shaft connected by springs
- Frictional contact interactions between blades and casing

Buckling of a blade in LP compressor

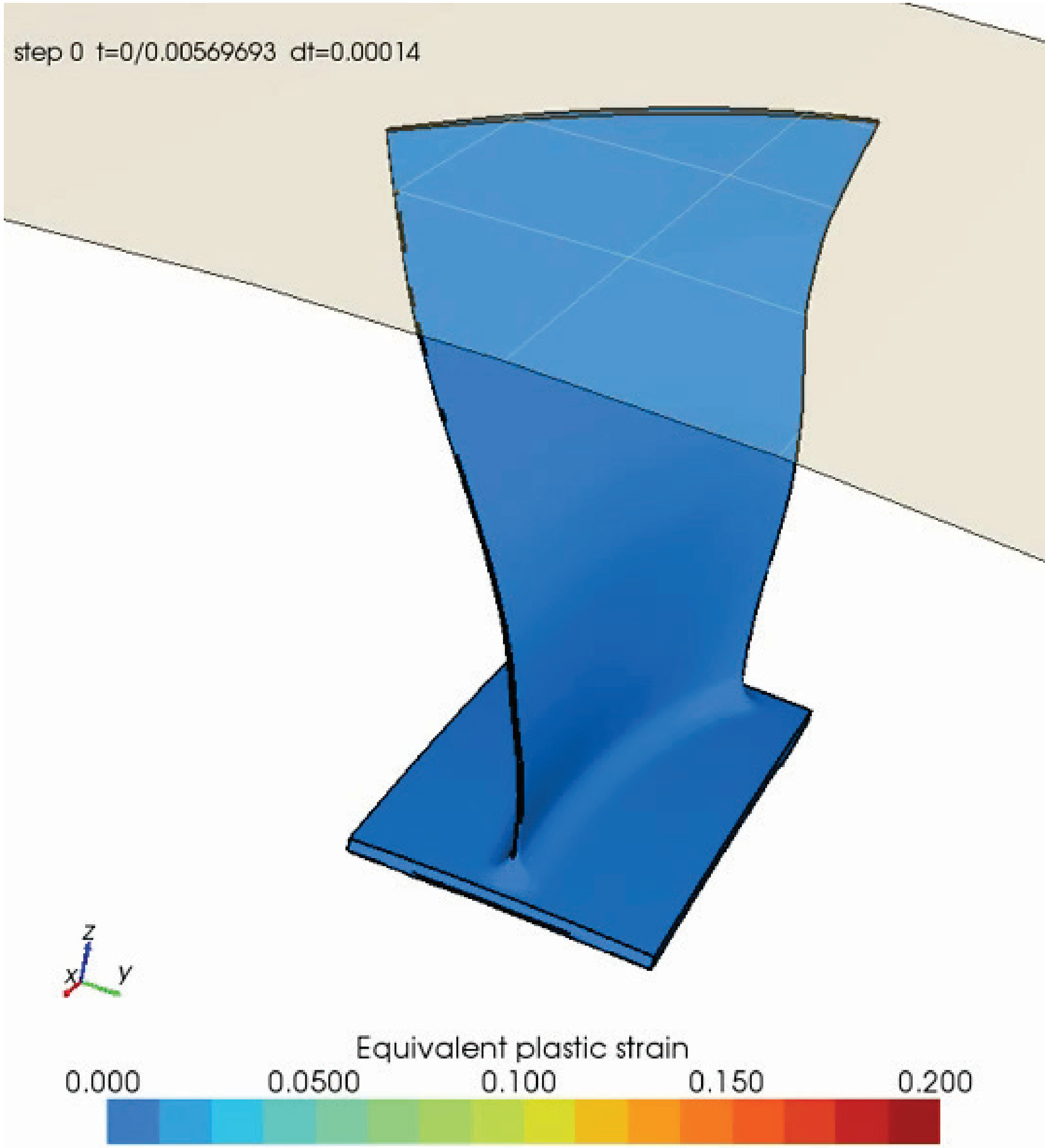


- Thermo elasto visco plastic model with damage / EAS finite elements
- We expect a lower buckling force compared to a purely elastic model

➔ the total weight of the engine can be safely decreased.

CPU time :

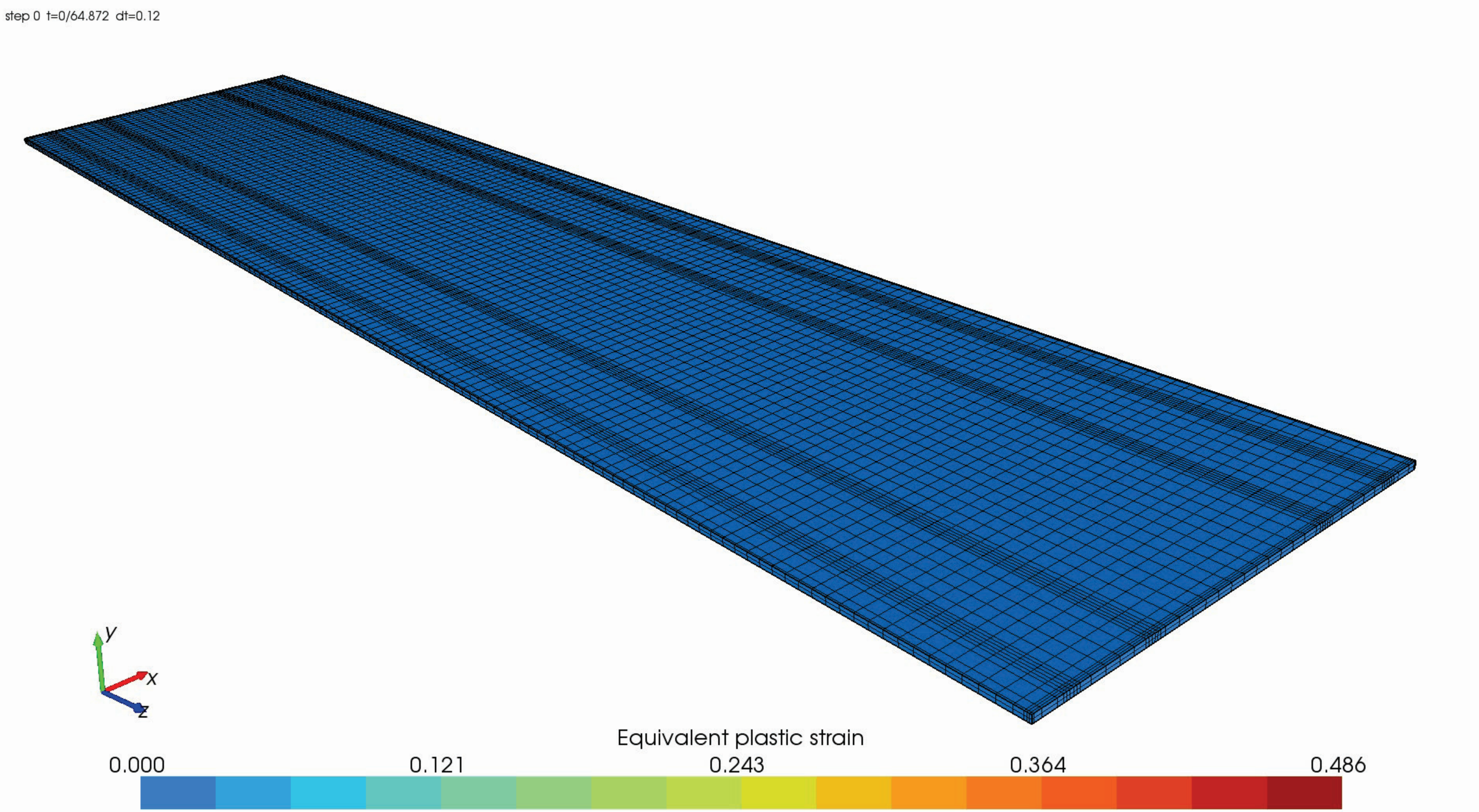
- 1 day 16 hours (1 core)
- 5h35’ (12 cores)



Roll forming simulation

Sheet Metal Forming : Simulation of a 15-stand forming mill

ArcelorMittal



Outline

1. Our in-house FE code : Metafor
2. Libraries and tools
3. Numerical examples
4. **Future works**

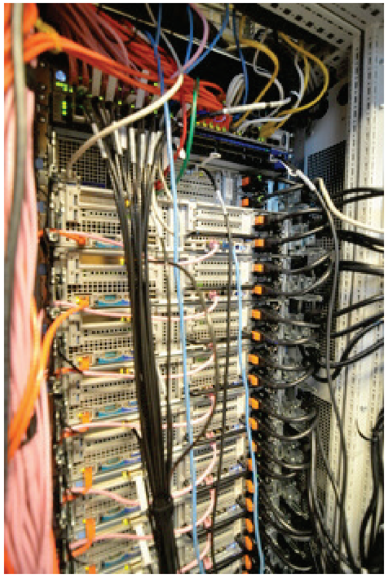
Future?

Shared Memory Processing (TBB)

Go on with TBB parallelisation of loops…

- The contact algorithms are still sequential.
- Some materials are not thread safe.

Distributed Memory Processing (MPI?)



Nothing done until today…

- Domain decomposition should be implemented.
- How to manage contact efficiently?

```
void mxv(int m, int n, double *a, double *b, double *c, int nbt, int tmax)
{
    #pragma omp parallel for num_threads(nbt)
    for (int i=0; i<m; i++)
    {
        for(int t=0; t<tmax; ++t)
        {
        a[i] = 0.0;
        for (int j=0; j<n; j++)
            a[i] += b[i*n+j]*c[j];
        }
    }
}
```

```
#pragma omp parallel for num_threads(nbt)
for (int i=0; i<m; i++)
{
    for(int t=0; t<tmax; ++t)
    {
        a[i] = 0.0;
        for(int j=0; j<n; j++)
            a[i] += b[i*n+j]*c[j];
    }

// loop on thread nb
int idx2=0;
for(int nbt=trange.getMin(); nbt<=trange.getMax(); nbt+=trange.getStep())
{
    idx2++;
    double tstart = omp_get_wtime();
    test.execute(nbt);
    double tstop = omp_get_wtime();
    double cpu = tstop-tstart;

    OMPData res = OMPData(idx1, idx2, siz, nbt, test.getMem(), cpu, test.flops(nbt))

    std::cout << res;
```