`MFEM-MGIS-MFront`, a `MFEM` based library for non linear solid thermomechanics

DE LA RECHERCHE À L'INDUSTRIE

MFEM Workshop

20/10/2021

T. Helfer, G. Latu

Outline

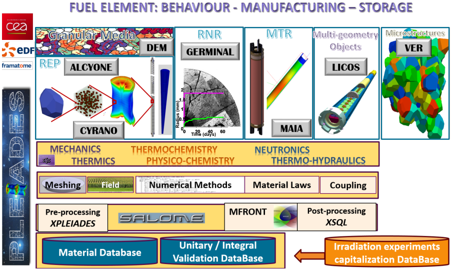
- **Context and goals**
- **A small tutorial**
- **Design and features of the `MFEM-MGIS` project**
- **Feed-backs on some issues using `MFEM`**
- **Examples**
- **Conclusions and perspectives**

Context and goals

Fuel modelling and the Pleiades platform

- French Atomic Energy Commission (CEA), public institution.
- Our department: modelling and simulation of nuclear fuel.

The Pleiades Platform



Existing mechanical solver not well parallelized

- A wide range of materials (ceramics, metals, composites).
- A wide range of mechanical phenomena and behaviours.
  - Creep, swelling, irradiation effects, phase transitions, etc..
- A wide range of mechanical loadings.

Goals of MFEM-MGIS

- Build a HPC general purpose non linear multi-physics library,
  development began end of 2020.
  - Primary focus is **non linear solid mechanics** and **heat transfer**.
  - Expected modelling (long-term): Shells, Beams,
    Phase-field approaches of brittle fracture, micromorphic models, Cosserat plasticity, strongly coupled thermo-chemical-mechanical or thermo-hydro-mechanical phenomena.

- A two-pillar library with *opensource* commitment (MFEM-MGIS *LGPL 3.0*):
  - `MFEM`: HPC finite element solver. *(LGPL 2.1)*
  - `MGIS/MFront`: constitutive laws, material modelling. *(LGPL 3.0 / GPL 3.0)*

A small tutorial

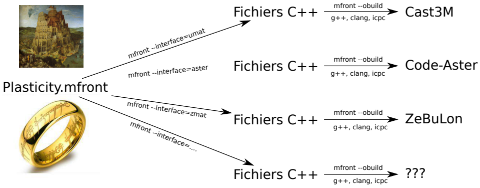
What are we talking about ? Example of end-user API

```cpp
// loading the mesh and building the non linear problem
mfem_mgis::NonLinearEvolutionProblem problem(
    {{"MeshFileName", mesh_file},  {"FiniteElementFamily", "H1"},
     {"FiniteElementOrder", order}, {"UnknownsSize", dim},
     {"Hypothesis", "PlaneStrain"}, {"Parallel", parallel}});
// associating names to element and boundary attributes (could be automated for some mesh formats)
problem.setMaterialsNames({{1, "NotchedBeam"}});
problem.setBoundariesNames({{3, "LowerBoundary"}, {4, "SymmetryAxis"}, {2, "UpperBoundary"}});
// declaring behaviour integrators
problem.addBehaviourIntegrator("Mechanics", "NotchedBeam", library, behaviour);
// setting the initial state of the materials
auto& m1 = problem.getMaterial("NotchedBeam");
mgis::behaviour::setExternalStateVariable(m1.s0, "Temperature", 293.15);
...
// defining boundary conditions, postprocessings and solver parameters
problem.addUniformDirichletBoundaryCondition({{"Boundary", "LowerBoundary"}, {"Component", 1}});
problem.addPostProcessing("ParaviewExportResults", {{"OutputFileName", "ssna303-displacements"}});
auto& solver = problem.getSolver();
...
// loop over time step
for (mfem_mgis::size_type i = 0; i != nsteps; ++i) {
  // updating the boundary values and resolution
  ...
  problem.solve(dt);
  problem.update();
  t += dt;
}
```

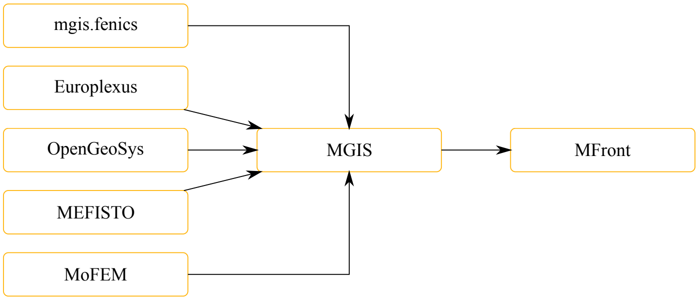
- Instantiating `NonLinearEvolutionProblem` class is the main entry point.
- **Behaviour integrator** is the main new concept of `MFEM/MGIS` .

Design and features of the `MFEM-MGIS` project

MFront goals



- `MFront` is a code generation tool dedicated to material knowledge (material properties, mechanical behaviours, point-wise models):
  - Support for small and finite strain behaviours, cohesive zone models, **generalised behaviours** (non local and or multiphysics).
- Main goals:
  - Numerical efficiency (see various benchmarks on the website).
  - Portability and coupling capabilities (`Cast3M`, `Cyrano`, `code_aster`, `Europlexus`, `TMFTT`, `AMITEX_FFTP`, `Abaqus`, `CalculiX`, `MTest`).
  - **Ease of use**: *Longum iter est per praecepta, breve et efficax per exempla* (It's a long way by the rules, but short and efficient with examples).



- The `MGIS` project provides classes on the solver side to retrieve **metadata** from an `MFront` behaviour and call the behaviour integration over a time step.
- Written in `C++`. Bindings exists for `C`, `Fortran2003`, `python`, `Julia`.
- Used/tested in `mgis.fenics`, `OpenGeoSys`, `Manta`, `XPer`, `MoFEM`, `Disk++`, `Kratos Multiphysics`, `JuliaFEM`, `NairmMPM`, `esys.escript`, `DUNE`, `OOFEM`, and more...

The added value of `MFEM-MGIS`

- Statement:
  - *Poor* support of multiple materials in `MFEM` for solid mechanics.
  - *No* support for functions on integration points for a given material (element attribute).
- Proposed improvements:
  - Add support for non linear behaviour integrators based on `MFront`.
  - Add functions on integration points that depends on material identifier (element attribute).
  - Simplified High-level API for end users (mechanics)

The added value of `MFEM-MGIS`

- Statement:
  - *Poor* support of multiple materials in `MFEM` for solid mechanics.
  - *No* support for functions on integration points for a given material (element attribute).
- Proposed improvements:
  - Add support for non linear behaviour integrators based on `MFront`.
  - Add functions on integration points that depends on material identifier (element attribute).
  - Simplified High-level API for end users (mechanics)
- In practice:
  - `MFEM-MGIS` is a library made up of
    1. Classes that inherit from `MFEM`:
       loops on elements, linear & non-linear solvers, mesh management...
    2. Classes that inherit from `MGIS`: materials management, internal variables, fill matrix entries...
    3. Procedures for the end user to interact with MFEM & MGIS

The added value of MFEM-MGIS

- Statement:
  - *Poor* support of multiple materials in MFEM for solid mechanics.
  - *No* support for functions on integration points for a given material (element attribute).
- Proposed improvements:
  - Add support for non linear behaviour integrators based on MFront.
  - Add functions on integration points that depends on material identifier (element attribute).
  - Simplified High-level API for end users (mechanics)
- In practice:
  - MFEM-MGIS is a library made up of
    1. Classes that inherit from MFEM:
       loops on elements, linear & non-linear solvers, mesh management. . .
    2. Classes that inherit from MGIS: materials management, internal variables, fill matrix entries. . .
    3. Procedures for the end user to interact with MFEM & MGIS
- MFEM-MGIS is a C++17 opensource library:
  - https://github.com/thelfer/mfem-mgis
  - https://github.com/latug0/mfem-mgis-examples

The role of behaviour integrators

Computation of the inner forces (MFEM/MGIS)

Computation of the strain $\underline{\varepsilon}^{\mathrm{to}}_{g} = B_{g}\,U_{e}$

$\downarrow$

Behaviour integration - (MGIS/MFront)

$\downarrow$

Element inner forces $F_{e} = \sum_{g} {}^{T}B_{g}\,\underline{\sigma}_{g}\,\mathrm{w}_{\mathrm{g}}$

Computation of the stifffness matrix (MFEM/MGIS)

Element stiffness matrix $K_{e} = \sum_{g} {}^{T}B_{g} \left(\dfrac{\mathrm{d}\underline{\sigma}}{\mathrm{d}\Delta\,\underline{\varepsilon}^{\mathrm{to}}}\right)_{g} B_{g}\,\mathrm{w}_{\mathrm{g}}$

- Behaviour integrators are associated with a material identifier (element attribute).
- Behaviour integrators are called in the assembly loop over the elements for:
  - the residual (contribution of the inner forces)
  - the jacobian matrix (i.e. the tangent stiffness matrix)

Behaviour integrators

- Behaviour integrators are meant to:
  - Compute the gradients from unknowns (using the B matrix).
  - Handle the state variables.
  - Call the behaviour integration.
  - Compute the inner forces from the thermodynamic forces.
  - Compute the stiffness matrix from the consistent tangent operator blocks.
- All those steps depends on:
  - Kind of problem described (mechanics, heat transfer).
  - Symmetry of the material (isotropic or orthotropic).
  - Modelling hypotheses: $3D$, plane strain, plane stress, axisymmetry ...
- The writting of behaviour integrators is tedious and error-prone and shall be done with care.
  - However, specific behaviour integrators gives access to pieces of physics.
  - Code generation to the rescue !

The `behaviour-integrator` code generator

- Aim of this code-generator
  - Generate behaviour integrators from definition of the gradients.
  - Automated code factorisation/optimisation (no sparse matrix multiply).
  - Avoid coding errors due to tedious formula.
- Machinery
  - Based on the `GiNaC` for symbolic computations in `C++`
  - Current scope: isotropic and orthotropic, small and finite strain behaviours in plane strain, plane stress and tridimensional hypotheses.
- Extensions (to come)
  - Support of axisymmetry
  - Non linear heat transfer, non linear diffusion.
  - Can be extended to other non-linear models, wide range of phenomena.

Feed-backs on some issues using `MFEM`

Issues related to the behaviour integration step

- The integration of the mechanical behaviour is a **complex** kernel solving a system of ordinary differential equations which can **fail**.

Issues related to the behaviour integration step

- The integration of the mechanical behaviour is a **complex** kernel solving a system of ordinary differential equations which can **fail**.
- Problem: the `NonlinearFormIntegrator::AssembleElementVector` and `NonlinearFormIntegrator::AssembleElementGrad` does not handle failures.

Issues related to the behaviour integration step

- The integration of the mechanical behaviour is a **complex** kernel solving a system of ordinary differential equations which can **fail**.
- Problem: the `NonlinearFormIntegrator::AssembleElementVector` and `NonlinearFormIntegrator::AssembleElementGrad` does not handle failures.
- As a work-around, we derived our own Newton algorithm which separates the behaviour integration step from residual and stiffness assemblies.

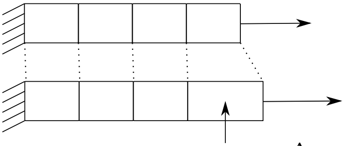
Issues related to the behaviour integration step

- The integration of the mechanical behaviour is a **complex** kernel solving a system of ordinary differential equations which can **fail**.
- Problem: the `NonlinearFormIntegrator::AssembleElementVector` and `NonlinearFormIntegrator::AssembleElementGrad` does not handle failures.
- As a work-around, we derived our own Newton algorithm which separates the behaviour integration step from residual and stiffness assemblies.
  - This solution seems compatible with partial assembly (untested).
  - **This work-around does not work with** `PETSc`.

Issues related to the behaviour integration step

- The integration of the mechanical behaviour is a **complex** kernel solving a system of ordinary differential equations which can **fail**.
- Problem: the `NonlinearFormIntegrator::AssembleElementVector` and `NonlinearFormIntegrator::AssembleElementGrad` does not handle failures.
- As a work-around, we derived our own Newton algorithm which separates the behaviour integration step from residual and stiffness assemblies.
  - This solution seems compatible with partial assembly (untested).
  - **This work-around does not work with** `PETSc`.
- More generally, we started a discussion on how to improve the robustness of operators in `MFEM`:
  - `https://github.com/mfem/mfem/issues/2139`

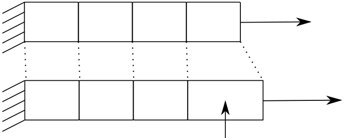
Dirichlet boundary conditions



$$\Delta\,\varepsilon = \frac{\Delta u}{h}$$

- Imposing the boundary Dirichlet in `MFEM` leads to unphysically strain increments on elements near the boundary at the first iteration:
  - The problem becomes more and more import as the mesh is refined.
  - In finite strain, this leads to severe divergence of the Newton algorithm du to the geometric stiffness matrix.

Dirichlet boundary conditions



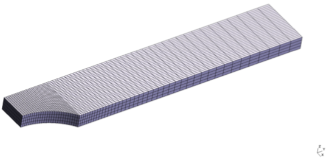
- Imposing the boundary Dirichlet in `MFEM` leads to unphysically strain increments on elements near the boundary at the first iteration:
  - The problem becomes more and more import as the mesh is refined.
  - In finite strain, this leads to severe divergence of the Newton algorithm du to the geometric stiffness matrix.
- In most **mechanical** solvers, a **prediction** of the solution based on the tangent problem is performed.
  - We were not able to implement this prediction so far.
  - `https://github.com/mfem/mfem/issues/2174`

Examples

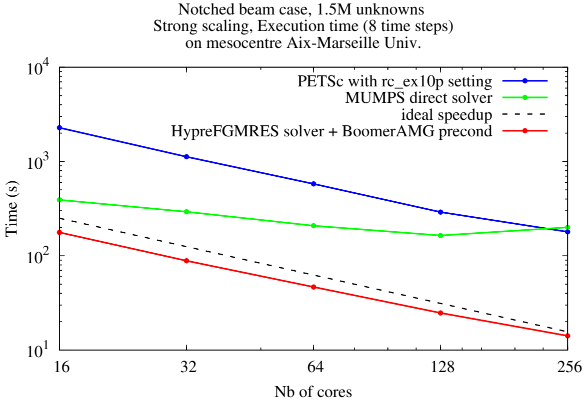
Non linear case
Elastoplastic modelling, finite strain

Case description

- Uniaxial tensile test on a 3D notched beam
- Imposed displacement: right of the beam
- Symmetry conditions at: left position, down position



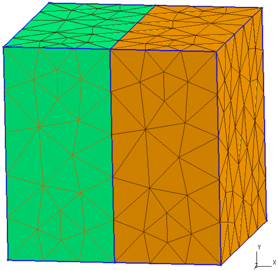
Geometry of the notched beam

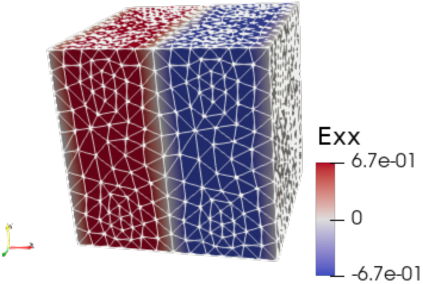


Test case setting - elastic model

- Two isotropic materials, each one filling half of a cube
- Aim : compare against `MFEM-MGIS` results against **analytical** results
  - considering 3 cases with uniaxial strain
  - considering 3 cases with shear test

Mesh and result





Periodic RVE - Numerical results on 32 cores

Properties

- 14 million unknowns, elastic modelling
- Ref. case on *32 cores* - skylake @ mesocentre Aix-Marseille University
- Main linear solver: Conj. Gradient (iterative), no preconditioner used

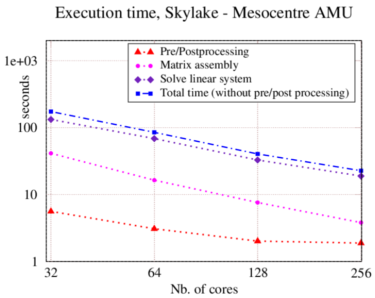
Timings

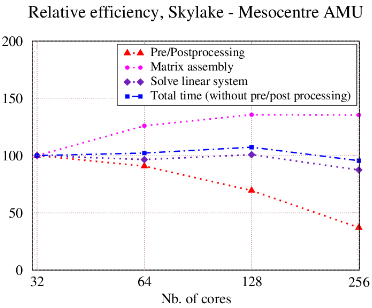
- MFEM linear solve
  - total **166s**, matrix assembly 39s, CGsolver 120s
- MFEM non-linear solve - residual form, a single newton iteration
  - total **179s**, matrix assembly 53s, CGsolver 117s
- MFEM-MGIS non-linear solve - residual form, a single newton iteration
  - total **188s**, matrix assembly 42s, CGsolver 133s
- Overheads due to MFEM-MGIS are low
  - Due to: read/write to memory buffers, function calls for assembly

Periodic REV - Strong Scaling

Strong scaling

- Using MFEM-MGIS non-linear solve
- Strong scaling from 32 to 256 cores
- Scalable parallelism observed



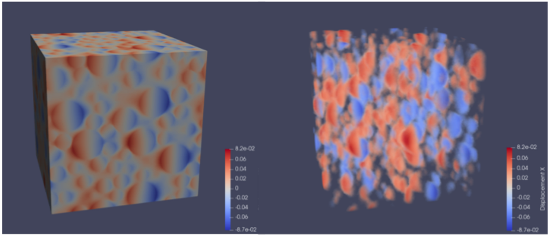


Periodic REV - Modelling inclusions

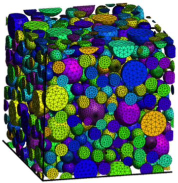
MFEM contributions & coupling

- using as input : 3D periodic `gmsh` meshes (public MFEM contrib)
- read/write MED files (private development)
- in progress : using `netgen` to get REV meshes (`gmsh` format)

Large runs



2000 inclusions (cut spheres), elastic modelling,
displacement in x direction is shown,
linear system with 500M unknowns



Conclusions and perspectives

Conclusions and perspectives

- What was done ?
  - High level declarative API suitable for engineering studies and upcoming integration in the `PLEIADES` platform.
    - `MFEM`' API still accessible user a lower level API (not shown here)
  - Multi-material support
  - Ability to handle arbitrary complex small and finite strain behaviours:
    - All behaviours are set at runtime (dynamically loaded libraries).

Conclusions and perspectives

- What was done ?
  - High level declarative API suitable for engineering studies and upcoming integration in the `PLEIADES` platform.
    - `MFEM`' API still accessible user a lower level API (not shown here)
  - Multi-material support
  - Ability to handle arbitrary complex small and finite strain behaviours:
    - All behaviours are set at runtime (dynamically loaded libraries).
- What comes next ?
  - Overall robustness of the code (critical).
  - Prediction and handling of the Dirichlet boundary conditions (critical).
  - Provide more examples (urgent).
  - Extension to other physical phenomea (non linear heat-transfer, diffusion, non local mechanics). *Should be easy* using code generation of behaviour integrators .
  - Adaptative mesh refinements (requires new data structures in MGIS).
  - Additional boundary conditions, including contact with friction.
  - Port to GPUs (requires tremendous work on the MFront and MGIS side).
  - Support for partial assembly (if possible ...), chekpoint/restart.

Thank you for your attention.

Time for discussion !

https://github.com/thelfer/mfem-mgis
https://github.com/thelfer/MFrontGenericInterfaceSupport
https://github.com/thelfer/mfem-mgis
https://tfel.sourceforge.net
https://www.researchgate.net/project/TFEL-MFront
https://twitter.com/TFEL_MFront

tfel-contact@cea.fr

The development of MFront is supported financially by CEA, EDF and Framatome in the framework of the PLEIADES project.

Non linear resolution in solid mechanics

- Mechanical equilibrium: find$\Delta\vec{\mathbb{U}}$ such as:

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right)=\vec{\mathbb{O}}\quad\text{avec}\quad\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right)=\vec{\mathbb{F}}_i\left(\Delta\vec{\mathbb{U}}\right)-\vec{\mathbb{F}}_e$$

- Resolution using the Newton-Raphson algorithm:

$$\Delta\vec{\mathbb{U}}^{n+1}=\Delta\vec{\mathbb{U}}^{n}-\underline{\underline{\mathbb{K}}}^{-1}.\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}^{n}\right)$$

- Element contribution to inner forces:

$$\vec{\mathbb{F}}_i^e=\sum_{i=1}^{N^G}\left(\underline{\sigma}_{t+\Delta t}\left(\Delta\underline{\epsilon}^{to}(\vec{\eta}_i),\Delta t\right):\underline{\mathbf{B}}(\vec{\eta}_i)\right)w_i$$

- Element contribution to the stiffness:

$$\underline{\underline{\mathbb{K}}}^e=\sum_{i=1}^{N^G}{}^t\underline{\mathbf{B}}(\vec{\eta}_i):\frac{\partial\,\Delta\underline{\sigma}}{\partial\,\Delta\underline{\epsilon}^{to}}(\vec{\eta}_i):\underline{\mathbf{B}}(\vec{\eta}_i)w_i$$

$\dfrac{\partial\,\Delta\underline{\sigma}}{\partial\,\Delta\underline{\epsilon}^{to}}$ is the {**consistent tangent operator**}