# New functionalities of Versions 4.1 and 4.2 of the TFEL/MFront project and Version 2.0, 2.1 of the MGIS project

Thomas Helfer, Maxence Wangermez, Salem Khellal, Yushan Wang, Raphaël Prat, Lionel Gelebart, Guillaume Latu

CSMA 2024
16ème Colloque National en Calcul des Structures
13-17 Mai 2024, Presqu'île de Giens (Var)

# New functionalities of Versions 4.1 and 4.2 of the TFEL/MFront project and Version 2.0, 2.1 of the MGIS project

Thomas Helfer[(1)], Maxence Wangermez [(1)], Salem Khellal [(2)], Yushan Wang [(2)], Raphaël Prat [(3)], Lionel Gélébart [(4)], Guillaume Latu [(3)]

[(1)] *CEA, DES/IRESNE/DEC/SESC/LMCP, Département d'Études des Combustibles, Cadarache, France*
[(2)] *CEA, DRF//UGP-MS, Saclay, France*
[(3)] *CEA, DES/ISAS/DRMP/SRMA/LC2M, Saclay, France*
[(4)] *CEA, DES/IRESNE/DEC/SESC/LDOP, Département d'Études des Combustibles, Cadarache, France*

**Abstract** — `MFront` is an open-source tool which allows easy implementation of arbitrarly complex mechanical behaviours in an efficient way. Those implementations are portable between various finite element solvers and FFT solvers. `MFront` is part of the open-source `TFEL` project.

The purpose of this paper is to highlight a selected set of features introduced in Versions 4.1 and 4.2 the `TFEL` project. The paper also covers the `MFrontGenericInterfaceSupport` project which allows to integrate `MFront` behaviours in existing open-source or commercial solvers.

## Contents



## Introduction

The projects `TFEL/MFront` and `MGIS` have been actively developped since the talk given at (1) where the version 4.0 of TFEL/MFront and Version 1.2 of MGIS were presented. Indeed, some

developpments have been made by a large community of academic and industrial users as demonstrated by the various talks at the MFront user days [1] and the list of publications using `MFront` [2].

`TFEL/MFront` is available on a large number of platforms, including `LinuX`, `macOS`, `Windows`, `FreeBSD` with various `C++` compilers (`gcc`, `clang` and `intel`). `TFEL/MFront` can be installed in several popular package managers, including `spack`, `conda` and `homebrew`. Moreover, it is distributed with the `Cast3M` and `code_aster` solvers.

This paper is devoted to highlight a selected set of features introduced in versions 4.1 and 4.2 of the `TFEL/MFront` project and versions 2.0 and 2.1 of the `MGIS` project. The interested reader may refer to the release notes of those versions for a comprehensive and detailled description:

— https://thelfer.github.io/tfel/web/release-notes-4.1.html
— https://thelfer.github.io/tfel/web/release-notes-4.2.html
— https://thelfer.github.io/tfel/mgis/release-notes-2.0.html
— https://thelfer.github.io/tfel/mgis/release-notes-2.1.html

This paper is organized a follows:

— Section 1 provides an overview of `TFEL`, `MFront`, `MTest` and `MGIS`
— Section 2 describes the main improvements to `TFEL/MFront` since Version 4.0.
— Section 3 describes some early results about porting `TFEL/MFront` to Graphical Processing Units (GPUs).

# 1 Overview of `TFEL`, `MFront`, `MTest` and `MGIS`

The `TFEL` project provides mathematical libraries which are the basis of the `MFront` code generator and the `MTest` solver (2). It is an open-source collaborative development of the French Alternative Energies and Atomic Energy Commission (CEA) and Électricité de France (EDF) in the framework of the `PLEIADES` plateform (3).

## 1.1 The `TFEL/Math` library

The `TFEL/Math` library provides:

— A linear algebra engine with mathematical objects (tensors of arbitrary orders) and operations on those objets required to express the constitutive equations in an efficient and natural manner, i.e. as close as possible to the mathematical expressions common in the engineering sciences. These mathematical objects can have units allowing the compiler to perform dimensional analysis at compile-time.
— A framework to build non linear solvers for small sized problems. Efficient and robust implementations of several classical non linear algorithms (Newton-Raphson, Broyden, Levenberg-Marquart, etc.) are provided.

## 1.2 The `TFEL/Material` library

The `TFEL/Material` library provides implementations of:

— Various utility functions frequently required in constitutive modelling (computation of the Lamé coefficients, computation of the Hill tensors).
— Various stress criteria and their derivatives with respect to the stress tensor (Hill 1948, Hosfort 1978, Gurson 1977, Gurson-Tvergaard-Needlman 1984, Barlat 2004, Mohr-Coulomb, etc..).

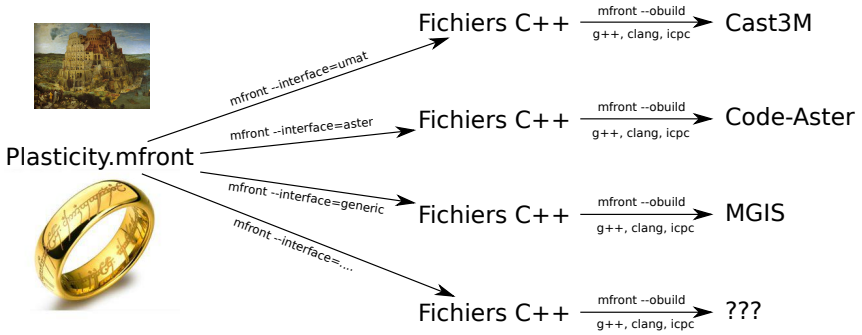


Figure 1 – Principle of `MFront`

### 1.3 The `MFront` code generator

`MFront` translates a set of closely related domain specific languages into plain C++ on top of the `TFEL` libraries.

Those languages are meant to be easy to use and learn by researchers and engineers. They cover three kinds of material knowledge: material properties (Young's modulus, thermal conductivity, etc.), mechanical behaviours [3] and simple point-wise models (such as material swelling under irradiation used in fuel performance codes). Concerning behaviours, the following kinds of behaviours are supported:

— small and finite strain mechanical behaviours.
— cohesive zone models.
— generalized behaviours, such as the one encountered in higher order theories, Cosserat media, heat transfer, strongly coupled thermomecanical analysis, damage gradient models, etc.

Authors of `MFront` pay particular attention to the robustness, reliability and numerical efficiency of the generated code, in particular for mechanical behaviours. Various benchmarks show that `MFront` implementations are competitive with native implementations available in the `Cast3M`, `code_aster`, `Europlexus`, `Abaqus/Standard`, `Abaqus/Explicit` (8) solvers and in the fuel performance codes `Cyrano3` (9) and `Galileo` (10).

Portability is also a very important issue: a behaviour written in `MFront` shall be usable in any solver for which an interface exists. In addition to the aforementioned solvers, interfaces exist for: `Ansys`, `ZMat`, `CalculiX`, `DianaFEA`.

### 1.4 The `generic` interface and the `MFrontGenericInterfaceSupport` project

To limit the number of interfaces supported by `MFront`, an interface called `generic` has been introduced, along with the `MFrontGenericInterfaceSupport` project (MGIS) which provides to solver developpers tools (functions, classes, bindings, etc…) to handle behaviours generated by this `generic` interface.

The `MFrontGenericInterfaceSupport` project has already been integrated or tested in many solvers (11), including `code_aster`, `Manta` (12), `MFEM-MGIS`, `mgis.fenics`, `OpenGeoSys` (13), `MFEF++`, `XPer` (14), `MOOSE`, `MoFEM`, `Disk++`, `Kratos Multiphysics`, `OOFEM`, `JuliaFEM`, `NSPFEM2D` (15), `esys.escript`, `DUNE`, etc.

1. https://github.com/thelfer/tfel-doc/tree/master/MFrontUserDays
2. https://thelfer.github.io/tfel/web/publications.html
3. Among the many projects aiming at easing the implementation of mechanical behaviours (see for example (4–6)), `MFront` can be compared to the `ZebFront` code generator which is part of `ZMat` library. A comprehensive comparison between these two solutions has been presented at the ZSet User Meeting (7).

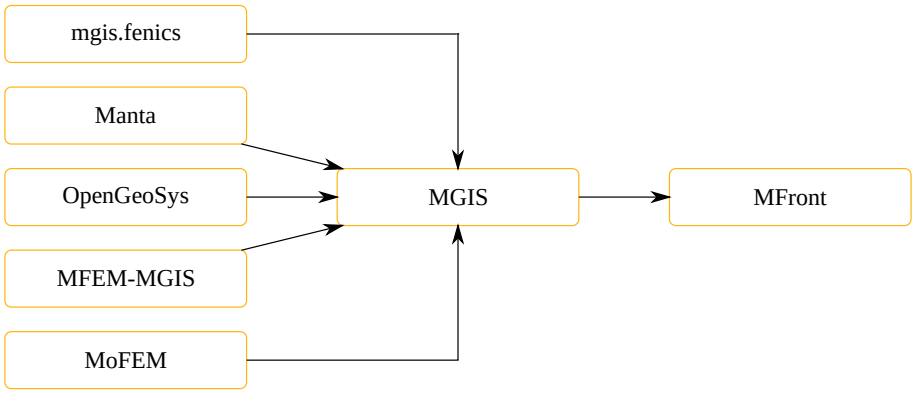


Figure 2 – Principle of `MGIS`

# 2 Improvements introduced in `TFEL/MFront` since Version 4.0

## 2.1 Extensions of the `StandardElastoViscoPlasticity` brick

The `StandardElastoViscoPlasticity` brick is a very high-level way of describing a large class of mechanical behaviours. This brick has been improved to:

- Let the user define an isotropic hardening rule by an arbitrary function of the equivalent plastic strain.
- Let the user define an isotropic hardening rule by a set of experimental points.
- Let the user define the viscoplastic strain rate by an abitrary analytical function of the selected norm of the effective stress and the equivalent viscoplastic strain.
- Introduce the Delobelle-Robinet-Schaffler kinematic hardening rule (16, 17).

## 2.2 Miscellaneous improvements

### 2.2.1 Initialize functions for behaviours

The `@InitializeFunction` keyword introduces a code block that can be used to initialize internal state variables at the very beginning of the computation. Initalize functions may have user so-called *initialize funtion variables*.

**Example of usage** The following code defines a function which initializes the elastic strain from the value of stress:

```
@InitializeFunction ElasticStrainFromInitialStress{
  const auto K = 2 / (3 * (1 - 2 * nu));
  const auto pr = trace(sig) / 3;
  const auto s = deviator(sig);
  eel = eval((pr / K) * Stensor::Id() + s / mu);
}
```

### 2.2.2 Post-processings of behaviours

The `@PostProcessing` keyword introduces a code block that can be used to perform computations independently of the behaviour integration. The outputs of post-processings are stored in so-called *post-processing variables*.

Post-processings are typically meant to be called at the end of a time step, when the equilibrium has been reached.

**Example of usage** The following code defines a post-processing computing the principal strain at the end of the time step:

```
//! principal strains
@PostProcessingVariable tvector<3u,strain> εᵖ;
εᵖ.setEntryName("PrincipalStrain");
//! compute the principal strain
@PostProcessing PrincipalStrain {
  εᵖ = eto.computeEigenValues();
}
```

### 2.2.3 Better control of code generation

Several options were added to modify at compile-time the code generated by `MFront`. Some options are related to quality assurance, such as specifying a build identifier, disabling the initialization of parameters from text file, specifying the default out of bounds policy, disabling the ability to change the out of bounds policy at runtime time or treating parameters as static variables (in this case, the parameters can't be changed at runtime).

Other options are related to performances. In particular, one can specify that some variables (material properties or external state variables) shall be overriden by parameters with known default values. If parameters are treated as static variables, this allows many optimisations of the generated code and reduces data transfered to the behaviour.

Such optimisations however requires to have access to information specific to the simulation considered and is thus only useful if the behaviours are compiled "Just-In-Time."

**2.2.3.1 Example of usage** The following code generates an optimised version of a behaviour were the temperature is fixed and the parameters are treated as static variables:

```
$ mfront --obuild --interface=generic                                    \
         --behaviour-dsl-option='overriding_parameters:{T: 293.15, dT: 0}' \
         --behaviour-dsl-option='parameters_as_static_variables:true'      \
         Plasticity.mfront
```

# 3 Porting to GPUs

GPUs are now commonly used to build most supercomputers for exascale simulations. Porting mechanical behaviours to GPUs is challenging: the behaviour integration step is a natural candidate to be a kernel as computations on each integration points are independent, but the following concerns need to be tackled:

- The amount of logics used in complex mechanical behaviours and very unbalanced workload (localized damaged or plasticity) is problematic on GPUs devices.
- The amount of data processed can be important, leading to many memory accesses from/to caches and global memory of the device.
- Data transfers between CPU and GPUs can hinder any performance boost gained by using GPUs.
- The number of local variables used may require a huge number of registers which are very limited on GPUs.
- The usual flexibility of mechanical behaviours allowing the use of uniform or spatially variable material properties (Young's modulus for instance) or external state variables (temperature for instance) is not compatible with GPUs.
- Many programming models, including `CUDA`, `HIP`, `SYCL` or `Kokkos` requires to modify the sources to distinguish host (CPU) and device (GPU) code.

### 3.1 GPU support in the `TFEL/Math` and `TFEL/Material` libraries

Being massively based on, generally `constexpr`, `template` functions, porting the `TFEL/Math` and `TFEL/Material` libraries, described in Sections 1.1 and 1.2 was quite direct, although explictely marking every functions as usable on both CPUs and GPUs is a tedious task.

Two major issues are worth noticing:

— The first one is related to the fact that error reporting through exceptions is not supported, or badly supported on GPUs. Removing exceptions usage required some deep refactoring in `MFront` to maintain backward compatibility.
— The second issue is related to the memory access pattern from the global memory on the device: one shall flavor structure of arrays rather than arrays of structures, as usually done one CPUs.
— The third issue is about registers usage. The default strategy of the `TFEL/Math` library is to allocate objects on the stack which is a very efficient strategy on CPUs but tends to saturate registers usage on GPUs.

### 3.2 Early results

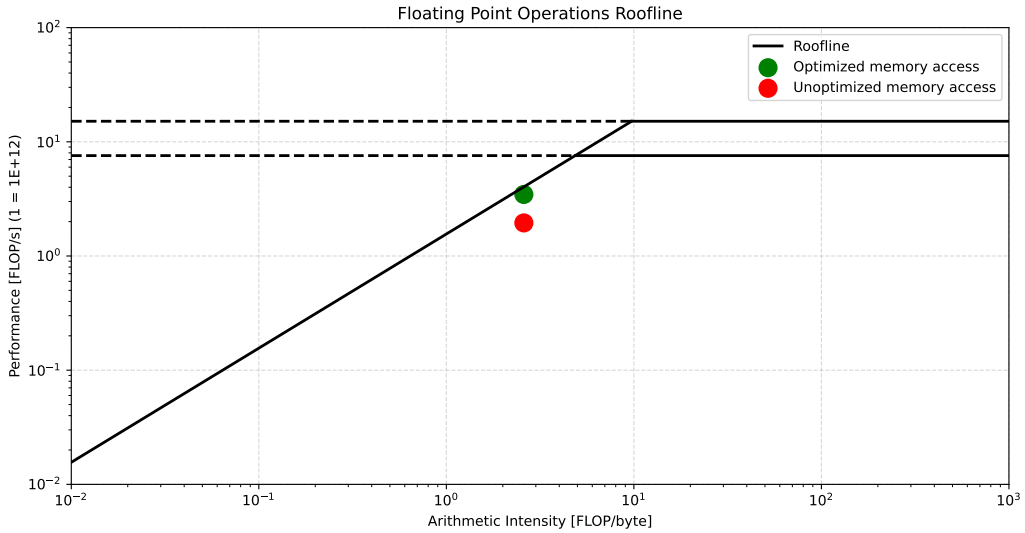


Figure 3 – Roofline obtained for an isotropic plastic behaviour with nonlinear isotropic hardening implemented using the CUDA programming model on a NVIDIA A100 GPU as provided by NVIDIA Nsight Compute.

Figure 3 shows that the GPU kernel obtained for an isotropic plastic behaviour with nonlinear isotropic hardening is fairly decent. This implementation is based on the code generated by `MFront` using the `generic` interface used by `MGIS` with some minor modifications.

This figure shows that the access pattern to global memory plays a crucial role on performance for simple behaviours, up to a factor 3. This access pattern can be transparently handled during the code generation phase.

Work are currently underway to optimize the kernels associated with more complex behaviours, notably by reducing the number of registers used.

## Conclusions and future works

`TFEL 4.2` is the latest minor release of the `4.x` series. This series will be supported for years to come and several bug fix versions will be released.

In parallel, the development of `5.x` series starts with a major overhaul of the `TFEL/Math` an `TFEL/Material` libraries. The aim of this overhaul is to reduce the complexity of the code and increase conformity with the `C++-23` standard.

The `5.x` series will focus on:

— Support of Just In Time compilation for improved performances, notably on GPUs. See Section 2.2.3 for details.
— Increasing the number of specific DSLs with highly optimised integration scheme (18).

*Acknowledgements* The authors are grateful to the many contributors to the `TFEL/MFront` project. This research was conducted in the framework of the PLEIADES project, which was supported financially by the CEA (Commissariat à l'Énergie Atomique et aux Énergies Alternatives), EDF (Électricité de France) and Framatome. Porting MFront to GPUs is supported financially by the CEA PTC-SIMU program through the INCOME project.

## References

1. Helfer, Thomas, Hure, Jérémy, Shokeir, Mohamed, Olivier, Fandeur, Mathieu Jean-Philippe an Raude Simon, Jamond Olivier an, Dominique, Geoffroy, Jérémy, Bleyer, Thomas, Nagel and Guillaume, Latu. New functionalities of Versions 3.3, 3.4 and 4.0 of the TFEL/MFront project and Version 1.0, 1.1 and 1.2 of the MGIS project. In : *Actes du 15e Colloque National en Calcul des Structures*. Giens, France, 2022.

2. Helfer, Thomas, Michel, Bruno, Proix, Jean-Michel, Salvo, Maxime, Sercombe, Jérôme and Casella, Michel. Introducing the open-source mfront code generator: Application to mechanical behaviours and material knowledge management within the PLEIADES fuel element modelling platform. *Computers & Mathematics with Applications*. September 2015. Vol. 70, no. 5, p. 994–1023. DOI 10.1016/j.camwa.2015.06.027. Available from: http://www.sciencedirect.com/science/article/pii/S0898122115003132

3. Marelle, Vincent, Goldbronn, Patrick, Bernaud, Stéphane, Castelier, Étienne, Julien, Jérôme, Nkonga, Katherine, Noirot, Laurence and Ramière, Isabelle. New developments in ALCYONE 2.0 fuel performance code. In : *Top fuel*. Boise, USA, 2016.

4. Stainier, Laurent. Introduction to MatLib: Founding concepts. 2010.

5. Japan Association for Nonlinear CAE. Unified material model driver for plasticity. 2018. Available from: https://www.jancae.org/annex/annexUMMDe/index.html

6. Portillo, David, Pozo, Daniel del, Rodríguez-Galán, Daniel, Segurado, Javier and Romero, Ignacio. MUESLI - a material UnivErSal LIbrary. *Advances in Engineering Software*. March 2017. Vol. 105, p. 1–8. DOI 10.1016/j.advengsoft.2017.01.007. Available from: http://www.sciencedirect.com/science/article/pii/S0965997816301430

7. Helfer, Thomas, Fandeur, Olivier, De Soza, Thomas, Deloison, Dominique and Toulemonde, Charles. Material knowledge management with the MFront code generator. Description of the ZMAT interface. Centre de l'Onéra, Chatillon, 2017. Available from: https://github.com/thelfer/tfel-doc/blob/master/Talks/ZSetUserDays2017/mfront-zset.pdf

8. Deloison, Dominique and Congourdeau, Fabrice. Testing and validation of the MFRONT interface for ABAQUS. EDF Lab Sacaly, 2016. Available from: https://github.com/thelfer/tfel-doc/blob/master/MFrontUserDays/SecondUserDay/deloison-abaqus.pdf

9. Petry, Charles and Helfer, Thomas. Advanced mechanical resolution in CYRANO3 fuel performance code using MFront generation tool. In : *LWR fuel performance meeting/TopFuel/WRFPM*. Zurich, Switzerland, July 2015.

10. Vioujard, N., Bessiron, V., Garnier, Christophe, Georget, V., Mailhé, P., Barbier, B., Deuble, D., Landskron, H., Bellanger, P. and Arimescu, V. I. GALILEO, AREVA's advanced fuel rod performance code and associated realistic methodology. In : *Proceedings of the TOPFUEL 2012 conference*. Manchester UK, 2010.

11. Helfer, Thomas, Bleyer, Jeremy, Frondelius, Tero, Yashchuk, Ivan, Nagel, Thomas and Naumov, Dmitri. The 'MFrontGenericInterfaceSupport' project. *Journal of Open Source Software*. 2020. Vol. 5, no. 48, p. 2003. DOI 10.21105/joss.02003. Available from: https://doi.org/10.21105/joss.02003

12. Jamond, Olivier, Lelong, Nicolas, Fourmont, Axel, Bluthé, Joffrey, Breuze, Matthieu, Bouda, Pascal, Brooking, Guillaume, Drui, Florence, Epalle, Alexandre, Fandeur, Olivier, Folzan, Gauthier, Helfer, Thomas, Kloss, Francis, Latu, Guillaume, Motte, Antoine, Nahed, Christopher, Picard, Alexis, Prat, Raphael, Ramière, Isabelle, Steins, Morgane and Prabel, Benoit. MANTA : Un code HPC généraliste pour la simulation de problèmes complexes en mécanique. In : *CSMA 2022 15ème colloque national en calcul des structures*. Giens, France, May 2022. Available from: https://hal.archives-ouvertes.fr/hal-03688160

13. Bilke, Lars, Flemisch, Bernd, Kalbacher, Thomas, Kolditz, Olaf, Helmig, Rainer and Nagel, Thomas. Development of open-source porous media simulators: Principles and experiences. *Transport in Porous Media*. 1 October 2019. Vol. 130, no. 1, p. 337–361. DOI 10.1007/s11242-019-01310-1. Available from: https://doi.org/10.1007/s11242-019-01310-1

14. Perales, Frederic, Socie, Adrien, Nkoumbou Kaptchouang, Noé Brice, Dubois, Frederic, Monerie, Yann, Mozul, Remy, Vincent, Pierre Guy and Babik, Fabrice. XPER : Une plateforme pour la simulation numérique distribuée d'interactions multiphysiques entre corps. 2022. Available from: https://hal.science/hal-03704427

15. Guo, N. and Yang, Z. X. NSPFEM2D: A lightweight 2D node-based smoothed particle finite element method code for modeling large deformation. *Computers and Geotechnics*. 1 December 2021. Vol. 140, p. 104484. DOI 10.1016/j.compgeo.2021.104484. Available from: https://www.sciencedirect.com/science/article/pii/S0266352X21004699

16. Delobelle, P., Robinet, P., Geyer, P. and Bouffioux, P. A model to describe the anisotropic viscoplastic behaviour of zircaloy-4 tubes. *Journal of Nuclear Materials*. 1 November 1996. Vol. 238, no. 2, p. 135–162. DOI 10.1016/S0022-3115(96)00450-3. Available from: https://www.sciencedirect.com/science/article/pii/S0022311596004503

17. Schäffler, I., Geyer, P., Bouffioux, P. and Delobelle, P. Thermomechanical behavior and modeling between 350°c and 400°c of zircaloy-4 cladding tubes from an unirradiated state to high fluence. *Journal of Engineering Materials and Technology*. 6 July 1999. Vol. 122, no. 2, p. 168–176. DOI 10.1115/1.482783. Available from: https://doi.org/10.1115/1.482783

18. Chaboche, J. L. and Cailletaud, G. Integration methods for complex plastic constitutive equations. *Computer method in applied mechanics and engineering*. 1996. Vol. 133, p. 125–155.