Implantation de lois de comportement mécanique à l'aide de MFront: simplicité, efficacité, robustesse et portabilité

Thomas Helfer, Jean-Michel Proix, Olivier Fandeur

CSMA 2015
12e Colloque National en Calcul des Structures
18 22 Mai 2015, Presqu'île de Giens (Var)

Implantation de lois de comportement mécanique à l'aide de MFront : simplicité, efficacité, robustesse et portabilité

T. Helfer[1], J.M. Proix[2], O. Fandeur[3,4]

[1] *CEA, DEN/DEC, Département d'Études des Combustibles, thomas.helfer@cea.fr*
[2] *EDF R&D, Département Analyses Mécaniques et Acoustique, jean michel.proix@edf.fr*
[3] *CEA, DEN/DM2S, Département de Modélisation des Systèmes et des Structures, olivier.fandeur@cea.fr*
[4] *IMSIA, UMR 8193, CNRS EDF CEA ENSTA*

Résumé — Ce papier est dédié au générateur de code open-source nommé MFront [6]. Plusieurs langages dédiés (domain specific languages), basés sur le C++ [23], permettent d'écrire de manière simple, compacte, efficace et robuste, l'implantation de lois de comportement mécanique. Après une vue d'ensemble, nous décrivons certaines des techniques de programmation utilisées. Nous présentons ensuite divers algorithmes permettant d'accroître la robustesse des schémas d'intégration implicites. Enfin, nous discutons certaines questions liées à la portabilité des implantations entre différents codes.

1 Introduction

Le projet PLEIADES vise à fournir une plate-forme unifiée de développement des applications de simulation des éléments combustible nucléaires du CEA et d'EDF [19]. Au sein de ce projet, le générateur de code MFront permet de traiter les aspects liés à l'écriture et la capitalisation des connaissances matériau dans un cadre d'assurance qualité strict. MFront s'est ensuite révélé utile hors du domaine combustible, notamment pour les utilisateurs des codes Cast3M [5], Code Aster [8] et AMITEX FTT [4]. Une interface pour le code ZeBuLoN [2] est également disponible. MFront a été mis en open-source en octobre 2014 dans l'espoir d'être utile aux ingénieurs et chercheurs de la communauté des matériaux et/ou du calcul de structure.

L'écriture de lois de comportement mécanique, par nature complexe, a un impact fort sur les temps de calcul et fait donc l'objet d'une attention particulière. MFront propose plusieurs langages dédiés aux lois de comportement mécanique (appelés domain specific languages) sur la base du C++ : certains sont dédiés à des formalismes spécifiques (lois de viscoplasticité ou plasticité isotropes), d'autres aux schémas d'intégration en vitesse et les derniers aux schémas d'intégration implicites.

La première section de ce document propose une vue d'ensemble de MFront.

La seconde met en avant une partie des techniques utilisées pour atteindre des performances numériques élevées. Nous concluons cette section par donner quelques éléments issus d'un benchmark comparant les lois générées via MFront aux lois natives du Code Aster.

La troisième section de ce document est dédiée à l'augmentation de la robustesse des algorithmes implicites. Nous y décrivons différents algorithmes introduits dans MFront.

La quatrième section de ce document décrit certaines questions liées à la portabilité de l'implantation d'une loi de comportement. Nous considérerons les codes Cast3M, Code Aster et ZeBuLoN [18]. Nous traiterons différents points liés aux stratégies de convergence utilisées par ces codes (choix de l'opérateur tangent) ou la gestion des contraintes planes.

2 Vue d'ensemble de MFront

Cette section donne une vue d'ensemble de MFront. Nous y décrivons un exemple très simple, puis nous présentons les différents algorithmes numériques disponibles avant d'aborder la notion d'interface.

```
@Parser IsotropicPlasticMisesFlow; //< domain specific language
@Behaviour Plasticity;             //< name of the behaviour
@Parameter H    22e9;              //< hardening slope
@Parameter s0   200e6;             //< elasticity limit
@FlowRule{                         //< flow rule
  f        seq-H*p-s0;
  df_dseq  1;
  df_dp    -H;
}
```

FIGURE 1 – Implantation d'une loi de comportement plastique avec un écrouissage isotrope.

2.1 Un premier exemple

La figure 1 décrit un fichier d'entrée utilisé par `MFront` pour implanter une loi de comportement plastique avec un écrouissage isotrope. Il s'agit d'un exemple d'un langage dédié à un type de loi particulier qui souligne qu'un des buts de `MFront` est de permettre une implantation de loi de comportement sous la forme la plus simple et la plus compacte possible.

Ce fichier est converti en sources `C++` formant un total de plus de 1500 lignes qui gèrent :

- l'algorithme d'intégration, tel que décrit par SIMO et HUGUES (réduction à une équation scalaire) [22] ;
- le calcul de l'opérateur tangent cohérent [21] ;
- dans le cadre d'une adhérence au code `Cast3M` (voir la notion d'interface ci-dessous), le traitement des contraintes planes et/ou le passage en grandes transformations suivant différents formalismes dont les déformations logarithmiques [15]. En effet, ces points doivent actuellement être gérés au niveau de la loi de comportement dans `Cast3M` alors que les codes `Code Aster` et `ZeBuLoN` proposent des solutions génériques en amont et/ou en aval de la loi de comportement. Ce point est discuté en section 5.

Les sources générées sont essentiellement des classes `template` paramétrées par l'hypothèse de modélisation et le type numérique utilisé (simple, double ou quadruple précision). Les techniques d'optimisation présentées plus loin (section 3) permettent d'obtenir, pour chaque hypothèse de modélisation, un code optimisable par le compilateur : il n'y a, par exemple, aucune allocation dynamique ni aucune boucle dans les opérations tensorielles.

2.2 Algorithmes numériques disponibles

`MFront` dispose de nombreux algorithmes d'intégration. On distingue deux classes d'algorithmes [3] :
- ceux basés sur un schéma d'intégration explicite de type `Runge Kutta` ;
- ceux basés sur un schéma d'intégration implicite.

La question de la robustesse des algorithmes implicites est traitée en section 4.

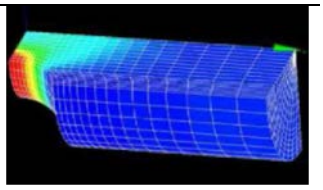
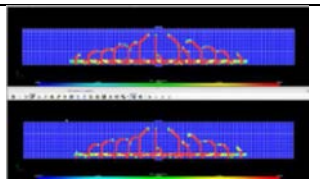
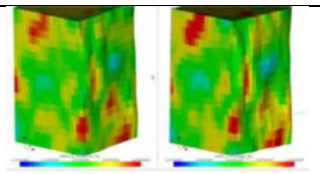
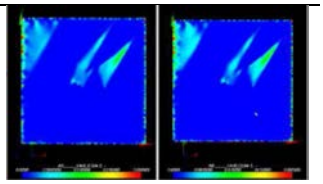
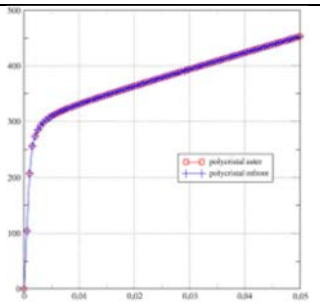
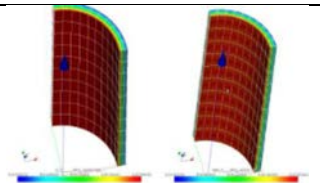
2.3 Interfaces

La notion d'interface dans `MFront` permet de générer un code spécifique au code cible. Il existe aujourd'hui des interfaces pour les codes aux éléments finis `Cast3M` [5], `Code Aster` [8] et `ZeBuLoN` [18] et l'application combustible `Cyrano3` [24]. L'interface au code aux éléments finis `Cast3M` a été reprise par le solveur FFT massivement parallèle `AMITEX FTT` [4].

3 Efficacité du code généré grâce aux techniques de programmation utilisées

Cette section s'intéresse à la question de l'efficacité du code généré. Un benchmark permet de montrer que malgré le caractère générique et multi-code, les performances obtenues sont satisfaisantes. La seconde partie décrit une partie des techniques utilisées pour obtenir ce résultat.

TABLE 1: Quelques exemples issus du benchmark comparant, dans `Code Aster`, les performances du code généré par `MFront` aux lois natives de ce code.

Description	Algorithme	Temps CPU total (`Aster` vs `MFront`)	Illustration
Visco-plastic and damaging for steel [10, 17]	`Implicit`	$17mn43s$ vs $7mn58s$	
Damaging for concrete ([12, 13])	`Default`	$45mn$ vs $63mn$	
Generic Single crystal viscoplasticity ([9, 14])	`Implicit`	$28mn$ vs $24mn$	
FCC single crystal viscoplasticity ([9, 16])	`Ìmplicit`	$33m54s$ vs $29m30s$	
FCC homogenized polycrystals 30 grains ([1, 9])	`Runge Kutta 4/5`	$9s67$ vs $8s22$	
Anisotropic creep with phase transformation ([11])	`Implicit`	$180s$ vs $171s$	

3.1 Benchmark

Le tableau 1 présente un extrait d'un benchmark réalisé par l'équipe de `Code Aster` comparant les performances du code généré par `MFront` aux lois natives de ce code.

On remarque que les lois générées par `MFront` sont en général compétitives avec les implantations natives [1]. L'un des intérêts forts de `MFront` est le temps de développement de ces lois de l'ordre de la journée.

1. Parmi les cas présentés, nous voyons que l'utilisation de la loi native de Mazars est nettement plus efficace. Ceci est en partie dû au fait qu'il s'agit d'une loi explicite : le temps passé dans la loi est inférieur au temps nécessaire au branchement de la loi (l'appel à la loi native est plus « direct »). Un travail est en cours pour réduire ce temps de branchement.

3.2 Template metaprogramming, expression templates et concepts

L'utilisation d'un générateur de code permet de masquer l'emploi de techniques de programmation avancées proposées par le langage `C++`, qui sont d'un accès difficile même pour des développeurs aguerris. Elles sont cependant une des clés pour obtenir des implantations numériquement efficaces.

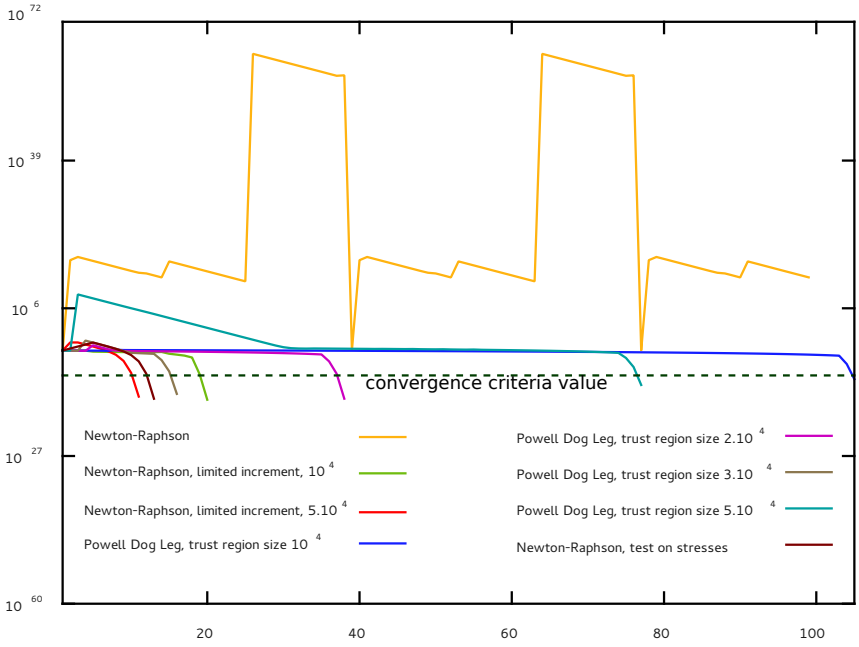


FIGURE 2 – Évolution du résidu en fonction du nombre d'itérations : échec de l'algorithme de NEWTON et emploi de stratégies alternatives.

4 Robustesse de l'intégration implicite

Le schéma d'intégration implicite est privilégié. Cependant, l'algorithme de NEWTON généralement utilisé pour résoudre le système non-linéaire qui en résulte peut manquer de robustesse, ce que nous avons illustré sur la figure 2 dans le cas d'une loi de comportement cristalline en grandes déformations.

L'utilisation de diverses modifications et stratégies alternatives à l'algorithme de NEWTON, notamment en réduisant la taille des corrections faites à chaque itération de l'algorithme, ou en considérant les algorithmes « en patte de chien » de POWELL ou l'algorithme de LEVENBERG-MARQUARDT [7, 20], est illustrée.

5 Sur l'écriture portable et efficace d'une loi comportement mécanique

Bien que la notion d'interface évoquée plus haut permette de se brancher à différents codes, ceux-ci présentent des particularités notamment liées aux algorithmes d'équilibre disponibles.

Dans le cadre de ce papier, nous proposons un survol à (très) grosses mailles de certaines de ces particularités dans le cas des codes `Cast3M`, `Code Aster` et `ZeBuLoN` : nous décrivons le traitement des contraintes planes et le calcul de l'opérateur tangent, si le code le nécessite.

Remerciements

Le développement de `MFront` a été soutenu par le projet `PLEIADES`, co-développé par le CEA, EDF et AREVA, et le projet `Simu Meca2015` d'EDF R&D. Les auteurs remercient Jacques BESSON et Stéphane QUILICI pour le temps qu'ils nous ont consacré lors de l'écriture de l'interface `zmat` dédiée au code aux éléments finis `ZeBuLoN`.