CSMA 2017
13ème Colloque National en Calcul des Structures
15-19 Mai 2017, Presqu'île de Giens (Var)

Tester la qualité numérique des codes de calcul avec Verrou

F. Févotte[1], B. Lathuilière[1]

[1] *EDF R&D, département PERICLES, {francois.fevotte, bruno.lathuiliere}@edf.fr*

Résumé — L'outil VERROU vise à faciliter le diagnostic et la correction des erreurs de calcul dans les outils de simulation industriels. Ces erreurs, dues aux propriétés de l'arithmétique flottante, peuvent être détectées et quantifiées grâce à l'Arithmétique en Arrondi Aléatoire (AAA). VERROU utilise cette arithmétique pour instrumenter les codes de calcul sans avoir besoin de les recompiler. Des fonctionnalités plus avancées de VERROU permettent aussi de localiser dans le code source l'origine des erreurs, facilitant ainsi le déboguage numérique.
Mots clés — qualité numérique, arithmétique flottante, arithmétique stochastique.

1 Introduction

Comme de nombreux autres industriels, EDF s'appuie fortement sur la simulation numérique pour conduire ses activités, qu'il s'agisse par exemple d'assurer la sûreté de son parc de centrales nucléaires ou d'optimiser l'utilisation des moyens de production. Il est donc important que tous – non seulement EDF, mais aussi des tiers comme l'Autorité de Sûreté Nucléaire – puissent avoir confiance dans les résultats de simulation produits par les Outils de Calcul Scientifique (OCS). À cette fin, les OCS sont soumis à un processus de Vérification et Validation (V&V), durant lequel plusieurs sources d'erreurs sont évaluées :

- les erreurs de modèle, *i.e.* les différences entre le "monde réel" et les objets mathématiques utilisés pour le représenter ;
- les approximations mathématiques utilisées pour simplifier les modèles, au premier rang desquelles on trouve par exemple les erreurs de discrétisation ou la résolution approchée des systèmes linéaires en résultant ;
- les erreurs de calcul, dues aux différences entre le comportement idéal des nombres réels, et le calcul réalisé en pratique par le CPU, qui utilise typiquement des nombres en virgule flottante selon la norme IEEE-754.

Les deux premières familles d'erreurs mentionnées ci-dessus ont pendant longtemps constitué les termes dominants, et ont par conséquent fait l'objet d'études poussées. Cependant, les progrès continuels de la puissance de calcul au cours des dernières décennies, ainsi que les avancées des méthodes numériques, ont permis d'accroître considérablement la complexité des modèles simulés (réduisant ainsi les écarts de modèles) tout en améliorant leur résolution (réduisant ainsi les approximations, par exemple de discrétisation *via* des maillages de plus en plus raffinés). De plus le calcul parallèle, dont l'utilisation est de plus en plus fréquente, provoque une non-reproductibilité de l'ordre des calculs, qui entraîne à son tour la non-reproductibilité des résultats en arithmétique flottante. L'impact des erreurs de calcul sur la qualité des résultats devient donc non négligeable, et l'analyse de l'arithmétique en virgule flottante constitue maintenant un sujet d'intérêt pour l'industrie. Pourtant, l'introduction de méthodologies appropriées dans les processus de V&V industriels n'est encore que marginale et reste assez largement à faire.

L'une des causes permettant d'expliquer ce constat est la quasi-absence d'outils permettant d'évaluer la qualité numérique de grands codes de calcul. Nous présentons dans ce document l'outil VERROU, qui vise à faire bénéficier le monde industriel des techniques d'analyse des erreurs d'arrondis connues du monde académique.

2 Vérification numérique avec VERROU

Parmi les différentes techniques permettant d'évaluer les instabilités numériques et les erreurs de calcul, la famille des méthodes fondées sur l'arithmétique de Monte-Carlo (*Monte-Carlo Arithmetic*, MCA) [1] semble être la plus prometteuse pour les applications industrielles. Par exemple, l'Arithmétique Stochastique Discrète (ASD), implémentée dans la bibliothèque CADNA [2, 3], a déjà été utilisée avec succès sur des codes industriels [4]. Cependant, son utilisation nécessite l'instrumentation complète des sources du programme à analyser, ce qui en fait une solution trop chère en pratique pour être mise en place largement dans l'industrie. Même des outils moins contraignants, comme par exemple Verificarlo [5] qui demande simplement de recompiler le code de calcul (ainsi que de ses dépendances à analyser) avec un compilateur spécifique, restent souvent d'une utilisation trop peu pratique pour être introduite de manière routinière dans un processus de V&V industriel.

Nous décrivons ici VERROU [6, 7], un outil de diagnostic des erreurs de calcul en arithmétique flottante, *open-source* [1] et développé par EDF R&D. VERROU a été conçu avec l'objectif d'en permettre l'application à de grands codes industriels, en garantissant que l'essentiel des fonctionnalités de diagnostic soient disponibles sans nécessiter de recompiler le code de calcul ou d'avoir accès à l'intégralité de ses sources (y compris les bibliothèques sur lesquelles il s'appuie). VERROU s'appuie pour cela sur la plate-forme Valgrind [8], qui est déjà largement utilisé dans les communautés de developpeurs pour aider au déboguage numérique. VERROU est par conséquent compatible avec la plupart des codes de calcul, et bénéficie de la même souplesse d'utilisation que Valgrind. Analyser un calcul ne demande qu'une légère modification de la ligne de commande :

```
valgrind --tool=verrou --rounding-mode=random PROGRAM [ARGS]
```

Lorsqu'il est appelé de cette manière, VERROU s'appuie sur les fonctionnalités d'analyse dynamique du binaire fournies par Valgrind pour instrumenter le code de calcul. Chaque instruction flottante est modifiée de sorte à renvoyer un résultat arrondi aléatoirement vers le haut ou le bas (au lieu d'un arrondi au plus près, comme cela est normalement le cas). Les résultats globaux du calcul sont affichés normalement, si ce n'est qu'ils sont affectés par l'accumulation des arrondis aléatoires dans tous les résultats intermédiaires. De ce fait, VERROU implémente une Arithmétique en Arrondi Aléatoire (AAA), qui peut être vue comme une forme spécifique de MCA, ou une méthode CESTAC asynchrone [9].

L'impact de VERROU sur le temps d'exécution du code s'élève en général à un facteur compris entre $\times 9$ et $\times 15$, même si des ralentissements d'un facteur supérieur à 30 ont été observés dans de très rares cas. Même s'il s'agit là de ralentissements importants, ceci reste compétitif par rapport aux outils d'analyse existants (Verificarlo : $\times 100$, CADNA : $\times 15$ en Fortran et $\times 5$ en C++).

Cette simplicité d'utilisation rend possible l'utilisation de VERROU dans un processus de V&V industriel. Les OCS disposent en effet généralement une d'une suite de cas-tests de non-régression pour laquelle les résultats sont comparés à une référence afin de vérifier leur validité. Un ensemble d'outils est souvent disponible pour produire une synthèse des résultats afin que les développeurs puissent déterminer d'un coup d'oeil si un changement dans le code introduit un changement trop important des résultats. L'introduction de VERROU dans un tel processus permet d'en perturber les résultats à l'aide de l'AAA. Ces résultats perturbés sont ensuite confrontés aux références selon le processus habituel afin d'évaluer la qualité numérique du code de calcul analysé.

VERROU propose aussi des fonctionnalités plus avancées, comme la possibilité de restreindre l'instrumentation à une portion du code (un ensemble de fonctions ou même un ensemble de lignes si le code a été compilé avec les bonnes options, comme `gcc -g`).

3 Localisation de l'origine des erreurs de calcul

VERROU propose deux manières de localiser l'origine des erreurs de calcul dans le code source. La première repose sur le mécanisme, évoqué ci-dessus, de restriction de la portée de l'instrumentation : en ne perturbant qu'une partie du programme, on identifie si cette partie produit des erreurs de calculs. Ceci permet de réaliser une recherche par bisection (utilisant l'algorithme de Delta-Debug [10]) afin de

1. `http://github.com/edf-hpc/verrou`

déterminer quelles fonctions (ou quelles lignes de code) sont à l'origine des plus grandes instabilités numériques. Un outil `verrou_dd` permet d'automatiser ce genre d'analyse.

La deuxième méthode d'analyse repose sur l'utilisation combinée de VERROU et d'outils d'analyse de couverture de code. Lorsqu'un programme est instrumenté pour réaliser une analyse de couverture (par exemple à l'aide de `gcc -fprofile-arcs -ftest-coverage`), son exécution produit un code source annoté permettant d'indiquer le nombre de passages dans chaque ligne de code. La comparaison de ces annotations pour deux exécutions perturbées par l'AAA permet de déterminer quels branchements dans le code source sont instables (*i.e.* quels tests "`if`" conduisent à prendre une branche différente s'ils sont perturbés par l'AAA).

4 Conclusion

Nous avons présenté dans ce document l'outil VERROU, qui vise à faciliter l'analyse de la qualité numérique des Outils de Calcul Scientifique industriels. Une telle analyse est rendue de plus en plus nécessaire par le fait que l'arithmétique flottante est responsable d'une part croissante des erreurs affectant les résultats de calcul. Dans ce contexte, VERROU vise à permettre l'introduction de la vérification numérique dans les processus de V&V industriels, en fournissant les principales fonctionnalités suivantes :

- compatibilité avec une large gamme d'outils de calcul (mêmes contraintes que Valgrind : pas besoin de recompiler, pas besoin d'un accès au code source…), multi-langage (C/C++, Fortran, Python, assembleur…) ;
- évaluation des erreurs de calculs accumulées, à l'aide de l'Arithmétique en Arrondi Aléatoire ;
- localisation dans le code source des fonctions ou lignes dans lesquelles s'accumulent les erreurs d'arrondis affectant le plus les résultats, ainsi que des tests instables.

VERROU a déjà été utilisé avec succès sur un certain nombres d'outils de calculs industriels, couvrant une vaste gamme d'applications et de méthodes numériques : Athena 2D (contrôle non-destructif par ultra-sons), code_aster (mécanique des milieux continus), MFront (lois de comportement), Apogene (optimisation de la production), Micado (neutronique), *etc.* Par ailleurs, l'outil étant *open source*, il peut constituer une base sérieuse pour quiconque souhaite vérifier la qualité numérique de ses outils de calcul.

Références

[1] D. Stott Parker. Monte Carlo arithmetic: exploiting randomness in floating-point arithmetic. Technical Report CSD-970002, University of California, Los Angeles, 1997.

[2] Fabienne Jézéquel, Jean-Marie Chesneaux, and Jean-Luc Lamotte. A new version of the CADNA library for estimating round-off error propagation in Fortran programs. *Computer Physics Communications*, 181(11):1927–1928, 2010.

[3] Jean-Luc Lamotte, Jean-Marie Chesneaux, and Fabienne Jézéquel. CADNA_C: A version of CADNA for use with C or C++ programs. *Computer Physics Communications*, 181(11):1925–1926, 2010.

[4] Sethy Montan. *Sur la validation numérique des codes de calcul industriels*. PhD thesis, Université Pierre et Marie Curie (Paris 6), France, 2013. in French.

[5] Christophe Denis, Pablo de Oliveira Castro, and Eric Petit. Verificarlo: checking floating point accuracy through Monte Carlo Arithmetic. In *IEEE Internatinal Symposium on Computer Arithmetic (ARITH)*, 2016.

[6] François Févotte and Bruno Lathuilière. VERROU: Assessing Floating-Point Accuracy Without Recompiling. https://hal.archives-ouvertes.fr/hal-01383417, October 2016.

[7] François Févotte and Bruno Lathuilière. VERROU: a CESTAC evaluation without recompilation. In *International Symposium on Scientific Computing, Computer Arithmetics and Verified Numerics (SCAN)*, Uppsala, Sweden, September 2016.

[8] Nicholas Nethercote and Julian Seward. Valgrind: A framework for heavyweight dynamic binary instrumentation. In *ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI)*, 2007.

[9] Jean Vignes. A stochastic arithmetic for reliable scientific computation. *Mathematics and Computers in Simulation*, 35:233–261, 1993.

[10] Andreas Zeller. *Why Programs Fail*. Morgan Kaufmann, Boston, second edition, 2009.