Reproductibilité et répétabilité numérique

Constats, conseils et bonnes pratiques pour le cas des simulations stochastiques parallèles et distribuées

David R.C. Hill[1], **Van Toan Dao**[1,2,3], **Claude Mazel**[1], **Vincent Breton**[3]

1. *LIMOS/ISIMA UMR CNRS 6158, Université Blaise-Pascal, Université Clermont-Auvergne, France david.hill@uca.fr, mazel@isima.fr*

2. *Labo, MSI/IFI UMI UMMISCO 209 IRD UPMC, Université nationale du Viêt-nam, Hanoi, Viêt-nam dvtoan@ifi.edu.vn*

3. *LPC CNRS/IN2P3, Université Clermont-Auvergne, BP10448, France vincent.breton@clermont.in2p3.fr*

RÉSUMÉ. *L'évolution récente des processeurs et accélérateurs de calcul, des compilateurs, des options de compilation et de bien d'autres éléments du contexte d'exécution d'une simulation, produisent une absence de répétabilité des expériences computationnelles. Ce sujet commence à prendre une place significative dans bien des domaines scientifiques. La recherche de reproductibilité numérique est essentielle à toute expérimentation scientifique, mais elle est bien des fois négligée. Dans cet article, nous proposons une analyse des enjeux de la reproductibilité numérique. Nous identifions ensuite les principaux obstacles à sa mise en œuvre. Pour pallier ces obstacles, nous proposons, dans le cas des simulations stochastiques parallèles, un ensemble de recommandations et une méthode qui aident à maintenir la répétabilité et la reproductibilité numérique des simulations stochastiques parallèles.*

ABSTRACT. *The recent evolution of processors and hardware accelerators, compilers, compiler options and other elements of the execution context of a simulation, produce a lack of repeatability of computational experiments. This subject is gaining attention in many scientific fields. The search for numerical reproducibility is essential for all scientific experimentations, but it is many times overlooked. In this article we propose an analysis of the challenges of numerical reproducibility. Then we identify the main obstacles to its implementation. To overcome these obstacles, we propose, in the case of parallel stochastic simulations, a set of recommendations and a method which helps maintaining repeatability and reproducibility of parallel stochastic simulations.*

MOTS-CLÉS : *répétabilité, reproductibilité numérique, simulation stochastique, parallélisme.*

KEYWORDS: *repeatability, numerical reproducibility, stochastic simulation, parallelism.*

DOI: 10.3166/tsi.2017.00008 © 2017 Lavoisier

Technique et science informatiques – n° 3-4/2017, 243-272

1. Introduction

Tous les chercheurs en sciences expérimentales sont habitués à remplir des cahiers de laboratoire avec un niveau important de détails (réglages de l'appareil, résultats bruts, techniques de traitement et procédures associées, méthodes statistiques utilisées, observations...). Toutes ces informations sont collectées afin que nos collègues soient en mesure de reproduire leurs expériences. Cette « reproductibilité » est un élément-clé de la méthode scientifique : avec les détails des expériences, d'autres scientifiques seront en mesure d'obtenir des résultats similaires. C'est sur cette reproductibilité que nous construisons, en confiance, de nouvelles avancées scientifiques. En science, la reproductibilité ne suppose pas que les personnes aient utilisé les mêmes techniques, approches ou méthodes. Au contraire, le fait que des collègues obtiennent les mêmes résultats scientifiques avec des approches différentes est un élément essentiel de validation. La reproductibilité scientifique implique divers processus : examen par les pairs, réplication des expériences, confirmation des résultats par d'autres équipes de recherche, vérification des travaux (c'est d'ailleurs par l'absence répétée de reproductibilité que débutent des soupçons de fraude). Aller plus loin dans bien des domaines scientifiques supposerait que nous ayons une recherche « ouverte » où l'ensemble des éléments permettant une découverte serait public. Dans la pratique, il n'est pas si facile pour d'autres scientifiques de retrouver les résultats disponibles dans la littérature, et ce pour de nombreuses raisons que nous discuterons dans le contexte des simulations informatiques stochastiques.

Dans le domaine de l'informatique, et plus particulièrement du calcul scientifique, toutes les expériences computationnelles devraient suivre une démarche scientifique similaire à celle de nos collègues d'autres disciplines. Il s'avère que nous ne faisons que commencer à être sensibilisés à ces préoccupations. Des pionniers issus de domaines applicatifs variés ont défriché le sujet. Nous le constatons *via* le travail important réalisé par Claerbout et ses collègues de Stanford (Schwab *et al.*, 2000 ; Fomel et Claerbout, 2009) et plus récemment par les travaux de Stodden au MIT, puis à Stanford également (Stodden, 2009, 2010).

Dans un premier temps, nous avons besoin de définir la notion de répétabilité qui est un cas particulier de la reproductibilité : il s'agit du fait d'être capable de « reproduire » exactement les résultats (Drumond, 2009). Obtenir les mêmes résultats avec les mêmes données d'entrée et le même algorithme est essentiel pour tester, vérifier et trouver des erreurs, et donc s'assurer d'un premier niveau de reproductibilité des expériences de calcul. Cela pourrait sembler trivial car nous sommes sur des machines de Turing déterministes, mais nous verrons que cela n'est pas si simple. Le premier écueil vient du fait que nous (les informaticiens) n'avons pas été sensibilisés dans nos formations à cette culture du cahier de laboratoire. C'est un fait, les expériences de calcul ne sont pas documentées aussi soigneusement que le serait une expérience de physique, par exemple. Nous n'avons pas forcément le réflexe d'enregistrer aussi soigneusement que nos collègues d'autres domaines de recherche, les éléments qui sont essentiels pour une répétabilité de nos expériences numériques. De plus, la pile logicielle dont nous nous servons fait que nous ignorons souvent bien des détails sensibles qui seraient importants

pour produire des logiciels scientifiques « reproductibles » (Hinsen, 2015a). Même si les sources des logiciels, les données et les résultats étaient disponibles gratuitement (ce qui est rare), nous serions loin du compte. Dans bien des cas sensibles du calcul scientifique, pour obtenir une répétabilité exacte, nous devrions enregistrer bien d'autres détails, comme : le type de matériel informatique, la configuration des logiciels, des bibliothèques et leurs versions, les options de compilation, les réglages des paramètres, l'architecture logicielle, sa conception, le processus de traitement (le *scientific workflow*), et parfois jusqu'à l'ordre d'invocation des fonctions et même l'ordre des opérations en virgules flottantes... Il faudrait parfois aussi archiver les « bonnes » versions des logiciels (face aux évolutions constantes) et garder trace des dépendances des bibliothèques, etc. Nous ne parlons pas du cas de prototypes, mais de logiciels scientifiques qui peuvent être utilisés en production. Pour les meilleurs développements, applications de production en physique ou pour la météorologie par exemple, tous ces paramètres sont documentés. Pour les développements de logiciels communs dans bien des laboratoires ou entreprises, le coût humain pour appliquer toutes les meilleures pratiques est prohibitif. En sciences, on parle de dette technique, comme on le fait dans l'industrie du logiciel (Hinsen, 2015b). De fait, un suivi rigoureux ne peut pas être appliqué dans la plupart des développements scientifiques, et l'habitude n'est absolument pas prise dans le développement de prototypes, ce qui conduit à un manque de répétabilité. Les pratiques courantes observées pour le développement de prototypes informatiques ne suivent que rarement les standards modernes du génie logiciel et cela affecte la productivité dans les laboratoires mais, surtout, la qualité et la crédibilité de leur travail. Cela participe à ce que Krishnamurthi et Vitek (2015) ont appelé, l'an passé, la vraie crise du logiciel. Récemment, Collberg *et al*. (2014) ont analysé plus de 600 articles de recherche en informatique publiés dans des journaux ou conférences de très bon niveau (issus de l'*Association for Computer Machinery* [ACM]). Pour ces articles, les auteurs ont essayé d'obtenir les codes sources et de reconstruire les logiciels (ce qui suppose que les auteurs partagent déjà de nombreuses informations). Cette étude a montré une faible disponibilité de ces sources (un peu plus de 30 % des codes ont été reconstruits) et, de ce fait, une faible répétabilité des résultats publiés. D'autres disciplines ne sont pas forcément plus à l'honneur : en 2015, l'Académie des sciences médicales britannique, qui est une référence en matière d'excellence scientifique, et les financeurs de cette recherche biomédicale ont évalué la reproductibilité des travaux financés (AcadMedSci, 2015). Le résultat final a fait beaucoup de bruit, y compris dans la presse grand public, fin 2015, car, *in fine* seulement un peu plus de 10 % des travaux étaient reproductibles.

Dans notre contexte de calcul informatique, outre la capacité à répéter les expériences à l'identique, la recherche de techniques informatiques permettant une reproductibilité numérique satisfaisante prend une importance croissante dans la science du calcul en général. En France, des pionniers organisent depuis plusieurs années des Rencontres de Réflexion autour de la Recherche Reproductible (R4) particulièrement animées et orientées autour du calcul scientifique. Les travaux de Stanisic *et al*. (2015) montrent par exemple comment mettre en œuvre un cahier de laboratoire numérique permettant de suivre précisément le flot des tâches informatiques en vue d'une recherche

reproductible. Ceux de Langlois *et al*. (2015) montrent comment obtenir des simulations numériquement reproductibles, notamment dans le cas de simulations hydrodynamiques (avec des éléments finis) et, ce qui est particulièrement remarquable, ils montrent comment obtenir des résultats séquentiels et parallèles comparables et identiques. On trouve dans l'article de Jézéquel *et al*. (2014), de nombreuses considérations sur les possibilités de reproductibilité numérique mettant en avant les travaux des équipes de Revol, Jézéquel et Langlois qui sont à la pointe de ces sujets. Les travaux de Revol se concrétisent également au sein d'un standard IEEE (1788) pour l'arithmétique d'intervalles et, plus récemment, nous remarquons des avancées et un regain d'intérêt pour l'arithmétique stochastique et le calcul à hautes performances (Eberhart *et al*., 2015).

Dans cet article, nous présentons les besoins en termes de recherche reproductible pour des simulations stochastiques parallèles utilisant des systèmes de calcul à hautes performances. Après une revue des concepts et des travaux dans ce domaine, nous notons les apports essentiels de la reproductibilité en section 2, notamment dans le contexte des simulations stochastiques parallèles. Nous présentons ensuite les causes majeures de non-reproductibilité (section 3). Sur la base de ces analyses, nous proposons des recommandations pour améliorer la reproductibilité des expériences et des résultats obtenus par une simulation stochastique parallèle (section 4). Ensuite, nous exposons une méthode de conception de simulations stochastiques parallèles reproductibles dans le cas de calculs indépendants (section 5) qui repose sur un usage pertinent de générateurs de nombres pseudo-aléatoires de qualité et des techniques de parallélisation adaptées à ces générateurs. Enfin, nous discutons de quelques approches utilisées pour les systèmes de calcul à hautes performances (section 6) avant de conclure.

2. La reproductibilité numérique : un besoin pour la simulation stochastique

La simulation est devenue un outil essentiel pour de nombreux domaines scientifiques. Elle est utilisée pour améliorer nos connaissances sur des systèmes, pour améliorer des méthodes, pour optimiser, tester et valider des modèles proposés pour des systèmes complexes qu'il serait difficile d'étudier autrement que par simulation.

Parmi les techniques de simulation qui se sont particulièrement répandues, nous trouvons les simulations stochastiques dérivées de la méthode de Monte Carlo. Ces simulations font appel au tirage de nombres pseudo-aléatoires. Dans cette classe de simulations, le générateur de nombres pseudo-aléatoires (Pseudo Random Number Generator [PRNG]) est un module fondamental pour simuler des variables aléatoires. De nombreux générateurs de qualité sont disponibles (L'Ecuyer *et al*., 2016). Ils sont censés être reproductibles et devraient être infaillibles sur cet aspect, mais nous avons montré que ce n'était pas toujours le cas, même pour les meilleurs d'entre eux. En effet, suivant les environnements (processeurs, systèmes d'exploitation, machines virtuelles, compilateurs…), la portabilité et la reproductibilité numérique font parfois défaut.

Une confiance aveugle n'est pas de mise et le lecteur intéressé trouvera les détails dans (Dao *et al*., 2014).

Si les simulations stochastiques permettent d'aborder de façon algorithmique bien des classes de problèmes, elles ont l'inconvénient d'être lentes. Cependant, dans la majorité des cas, et spécifiquement dans le cas des simulations de type Monte Carlo, les calculs sont indépendants. De plus, les simulations stochastiques font appel à des plans d'expériences qui, eux aussi, génèrent de nouveaux calculs indépendants. Cette caractéristique fait que ces simulations sont considérées comme faciles à paralléliser dans le but d'améliorer très significativement leurs performances par rapport à une exécution séquentielle. Cette facilité présente néanmoins un coût caché : en effet, bien des simulations stochastiques parallèles sont implémentées avec un manque de rigueur dans leur conception informatique. Hellekalek (1998) nous avertissait à ce sujet il y a près de 20 ans. Les domaines de recherche tels que la sûreté civile face aux catastrophes naturelles, le nucléaire ou la médecine nucléaire ne peuvent pas se contenter d'une qualité approximative des résultats finaux en raison de mauvaises techniques de distribution des nombres pseudo-aléatoires parallèles, par exemple. Dans ces domaines pointus, la méthode scientifique est une préoccupation, et la reproductibilité des résultats est un passage obligé. Notre expérience provient plus particulièrement de la réalisation de logiciels parallèles pour la médecine nucléaire et pour la physique des particules. Dans ces contextes, nous avons pu mettre en évidence de nombreux problèmes dans des simulations en médecine nucléaire qui nécessitent bien souvent plus de 10^{20} nombres pseudo-aléatoires et qui se déploient sur des milliers de processeurs (Maigne *et al*., 2004 ; Lazaro *et al*., 2005 ; El Bitar *et al*., 2005 ; Reuillon *et al*., 2008). Dans la majorité des domaines, la reproductibilité scientifique est bien moins sensible, mais elle n'en reste pas moins importante.

Pour développer avec rigueur une simulation stochastique parallèle, il est nécessaire de disposer d'un grand nombre de flux stochastiques indépendants. Le flux (*stochastic stream* dans la terminologie utilisée par les spécialistes) correspond au vecteur de tous les nombres pseudo-aléatoires utilisés par une exécution d'une simulation. Les multiples expériences (réplications) doivent disposer de vecteurs « indépendants ». Ceci suppose l'usage de techniques de parallélisation et de distribution de multiples flux de nombres pseudo-aléatoires que nous aborderons.

Une simulation stochastique parallèle est exécutée sur un système multiprocesseur. Les systèmes de calcul à hautes performances sont bien évidemment le meilleur choix pour améliorer le temps d'exécution. Cependant, la reproductibilité numérique de ces simulations pose plus de problèmes sur ces systèmes. Pour de nombreuses simulations, la pertinence des résultats obtenus est remise en cause (Taufer *et al*., 2010 ; Diethelm, 2012 ; Hill, 2015).

3. Apports de la reproductibilité pour les simulations stochastiques parallèles

Dans cette section, nous allons observer que, comme dans de nombreux domaines, il existe, dans le cas des simulations stochastiques, une variabilité dans la définition du mot

« reproductibilité ». Vandewalle *et al*. (2009) ont proposé comme définition de la reproductibilité la capacité pour un chercheur indépendant disposant des données d'une expérience, de reproduire les résultats de celle-ci. Bien avant, Srinivasan *et al*. (1999) restreignaient le concept de reproductibilité pour un logiciel à sa capacité à être porté correctement d'une machine à une autre. Récemment, Jimenez *et al*. (2014) ont considéré que la reproductibilité consiste en la capacité à refaire exactement une simulation et à obtenir les mêmes résultats, ce qui correspond en fait à la notion de répétabilité (Drumond, 2009) et, dans une moindre mesure, à la notion de reproductibilité numérique.

La reproductibilité est bien plus générale que la répétabilité. Stodden a introduit différents niveaux possibles de reproductibilité au sens large. Les différents niveaux proposés sont les suivants : l'examen par les pairs, la confirmation des résultats, la vérification, la validation et la recherche ouverte (Stodden *et al*., 2013). Feitelson (2015) a également présenté plusieurs définitions pour la reproductibilité ainsi que pour d'autres termes proches, ainsi que les relations entre ces termes. Au-delà de la notion de reproductibilité, il a également introduit, sa définition du concept de corroboration.

Selon Demmel et Nguyen (2013), la reproductibilité numérique consiste à obtenir, d'une exécution à une autre, des résultats identiques jusqu'au niveau du bit (*bitwise*). Par rapport à cette définition, Revol et Théveny (2014) soulèvent la difficulté de définir le résultat et de définir son exactitude. Ils proposent donc une autre définition de la reproductibilité, à savoir l'obtention d'un même résultat lorsque le calcul scientifique est exécuté plusieurs fois, soit sur la même machine, soit sur des machines différentes, pour différents nombres d'unités de traitement, différents environnements d'exécution et différentes charges de calcul.

Dans la suite de cet article, nous retenons que la répétabilité consiste à obtenir des résultats strictement identiques et que la reproductibilité numérique vise à obtenir des résultats très proches que l'on peut considérer équivalents. Quant à la reproductibilité au sens large, il s'agit éventuellement de travaux bien différents mais menant aux mêmes conclusions scientifiques.

À partir de ces définitions, nous identifions trois apports essentiels de la reproductibilité numérique pour des simulations stochastiques parallèles.

3.1. Une approche de test et de mise au point

La reproductibilité numérique sert à déboguer et à tester une simulation sur différents environnements d'exécution. Si l'on a la même donnée d'entrée, la même simulation et les mêmes conditions de tests, alors on doit obtenir des résultats identiques. La vérification de cette propriété incombe à l'auteur de la simulation.

3.2. Une méthode de vérification et validation

La reproductibilité numérique impose une méthode et constitue un standard pour juger de la pertinence d'une expérience numérique publiée et donc des conclusions qui

en découlent. Il appartient aux auteurs de l'expérience de fournir les réponses aux questions suivantes :

– Est-ce que les résultats des simulations sont reproductibles ?

– Peut-on répéter les résultats de ces simulations ?

– Comment reproduire l'expérience de simulation ?

– Est-ce que les résultats des simulations sont exacts et crédibles ?

– Comment évalue-t-on la qualité de résultats de recherche obtenus par le calcul ?

Si l'on peut répondre de façon satisfaisante à ces questions, d'autres scientifiques peuvent refaire les expériences et retrouver des résultats identiques à ceux qui sont présentés dans les publications.

3.3. La promotion du developpement scientifique

La reproductibilité numérique est également une façon de promouvoir le développement scientifique. Dans le contexte de la simulation stochastique parallèle, la reproductibilité numérique doit signifier que la simulation peut être refaite avec différents langages de programmation utilisant les mêmes normes de calcul scientifique (notamment IEEE 754, DEC64 ou toute autre norme pour le calcul scientifique), différents types d'exécution (séquentielle ou parallèle) et éventuellement différents modèles de programmation parallèle, etc. C'est un défi très difficile à relever, dès que la simulation revêt une structure complexe.

4. Pourquoi observons-nous, depuis quelques années, une non-reproductibilité des expériences computationnelles ?

4.1. Quelques pistes expliquant la non-répétabilité des calculs

Nous avons présenté dans la section précédente les concepts de répétabilité et de reproductibilité, leurs variantes et leur importance dans notre contexte de calcul scientifique pour des simulations parallèles. Faire une expérience numérique reproductible n'est pas une chose simple.

Selon Jimenez *et al*. (2014), la non-reproductibilité d'une expérience computationnelle peut provenir du matériel (processeur, système d'exploitation, configurations ou bibliothèques) ; de l'utilisateur (scripts et bibliothèques de tierces parties) ; des paramètres/données d'entrée ; des services externes (données, système de fichiers parallèles, etc.).

Dans le cas de la simulation stochastique parallèle, d'autres sources de non-reproductibilité doivent être prises en compte. Nous les avons classées en cinq catégories (figure 1) : trois concernent tous les types de simulations (la culture de publication scientifique, le calcul en virgule flottante et le type de matériel) tandis les

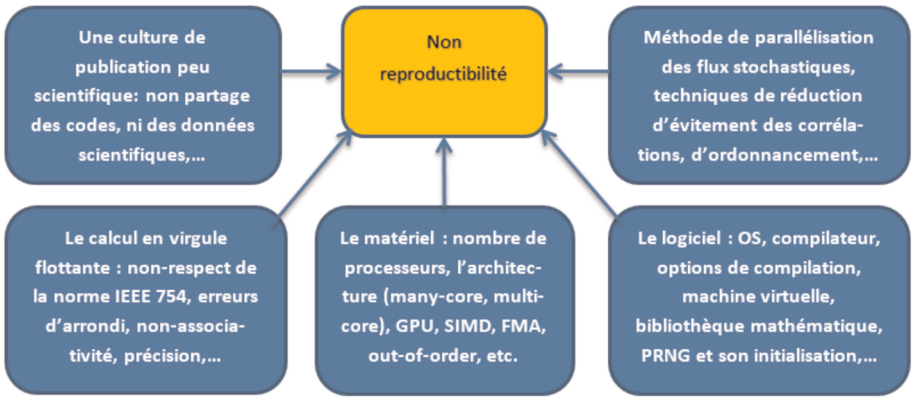

Figure 1. Les principales sources de non-reproductibilité numérique dans le contexte des simulations stochastiques parallèles et distribuées

deux autres sont plus spécifiques à la simulation stochastique parallèle à haute performance (générateur de nombres pseudo-aléatoires, techniques de distribution et de parallélisation des flux stochastiques, technique de conception adaptée aux critères stochastiques). Nous allons maintenant analyser ces catégories plus en détail.

4.2. La culture liée aux publications scientifiques

Les articles publiés présentent les résultats de travaux de recherche mais, souvent, ils ne précisent pas suffisamment les détails pratiques, notamment les données et les codes sources. Dans l'étude de Collberg que nous avons citée, seulement 102 publications sont reproductibles. Quelques chiffres de cette étude parlent d'eux-mêmes : 105 auteurs ont été impossibles à contacter, 179 n'ont pas répondu au courrier et 129 tentatives de recompilation ont échoué (Collberg *et al.*, 2014). Stodden (2010) a aussi étudié les conséquences du non-partage des données et des codes sources, mettant en évidence l'impact des mauvaises habitudes de nos communautés scientifiques. Il arrive parfois qu'en travaillant avec des collègues d'un autre domaine que l'informatique (biologie, physique…), le souhait exprimé par « l'informatique » de mettre les codes publiquement à disposition ne soit pas compris et que le « protectionnisme » prévale. Dans ce cas (que nous avons vécu), il est possible d'isoler la contribution générique de l'informatique et de mettre à disposition librement une sous-partie du code, en tant que cadriciel (*software framework*) pour un domaine. Ce n'est pas satisfaisant en termes de reproductibilité scientifique, mais cela permet au moins de continuer à faire progresser une recherche libre et ouverte.

4.3. Les raisons liées à l'arithmétique flottante

La majorité des résultats obtenus par simulation ne sont pas reproductibles à cause des limites de l'arithmétique flottante et d'une méconnaissance de ces limites. La norme

IEEE 754 pour l'arithmétique binaire en virgule flottante est la plus courante, elle permet une approximation d'un nombre réel sur un ordinateur avec un significande (une mantisse), une base, un exposant et un signe. Pour la majorité des applications une représentation en double précision est suffisante et chaque système fournit des bibliothèques mathématiques normalisées. Les ordinateurs classiques et les super-calculateurs réalisent les calculs en virgule flottante grâce à leurs unités arithmétiques et logiques appelées unités à virgule flottante (*Floating Point Units* [FPU]). Il est cependant important de connaître les limites du calcul à virgules flottantes. Le lecteur intéressé se référera utilement à l'article de Goldberg (1991). Il est en effet pertinent de se rappeler que l'ordre des opérations utilisant l'arithmétique flottante est important. Si dans l'espace mathématique des nombres réels l'associativité est de mise, ce n'est plus le cas pour le calcul informatique utilisant des virgules flottantes. Les approximations faites altèrent les résultats d'une manière différente en fonction de l'ordre d'exécution des opérations. Pour réaliser l'impact sur des expressions simples, prenons le cas de la non-équivalence des expressions suivantes (1) et (2) qui sont différentes en arithmétique flottante mais équivalentes dans l'espace des nombres réels.

$$a + (b + c) \tag{1}$$

$$(a + b) + c \tag{2}$$

Plaçons-nous dans un cas concret qui donne respectivement la « bonne » puis la mauvaise réponse, du fait des approximations faites par des opérations utilisant la double précision de la norme IEEE 754. Considérons l'expression $(2^{60} - 2^{60}) + 1$: elle équivaut à $0 + 1$, qui vaut finalement 1. Si l'ordre des opérations est changé et que l'on fait le calcul $(2^{60} + 1) - 2^{60}$, le résultat sera faux et donnera 0. En effet, avec la représentation en double précision, seuls 53 bits sont disponibles pour la mantisse et l'expression $2^{60} + 1$ s'arrondira à 2^{60}. Ainsi, on obtient $2^{60} - 2^{60}$ ce qui vaut 0 et donne un résultat faux. Pour le cas précédent, nous recommandons bien sûr l'utilisation de précisions étendues (par exemple les « long double ») ou bien l'utilisation de bibliothèques de calcul, certes plus lentes, mais adaptées au calcul nécessitant un grand nombre de chiffres significatifs (cas de nombreuses simulations en cosmologie).

Par défaut, certains langages utilisent les 10 octets disponibles au sein des unités de calcul flottant de marque Intel pour les calculs de précision, mais ils ignorent bien des propriétés d'arrondi de la norme IEEE 754 (le calcul en double précision travaille sur 8 octets). Ces choix des langages ont des impacts significatifs sur les résultats. Entre 1998 et 2004, des conférences de Kahan et Darcy (Berkley, Stanford) signalaient déjà les problèmes de ce type notamment avec le langage Java « *How Java Floating Point hurts everyone everywhere* » (http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf).

Dans la dernière version de la norme IEEE 754, en 2008, les limitations clairement identifiées au niveau de l'associativité (concernant l'ordre des opérations) et de la gestion des erreurs d'arrondis sur l'addition et la multiplication ont été à nouveau mises en évidence pour l'ensemble de la communauté scientifique. De nombreux travaux signalent ces limites dans un contexte de reproductibilité (Demmel et Nguyen, 2013 ;

Revol et Théveny, 2014) et rappellent les fondamentaux encore méconnus que Goldberg enseignait dès 1991.

4.4. Les raisons liées au matériel utilisé

4.4.1. Les accélérateurs de calcul

Une simulation stochastique parallèle peut être spécifique à certains matériels (accélérateurs de calculs de types, GP-GPU, FPGA, Intel Xeon Phi…), limitant de fait la portabilité de l'application scientifique. Lorsqu'une simulation stochastique parallèle fonctionne sur plusieurs types de matériels mais que les résultats obtenus ne sont pas identiques, alors on peut dire que la reproductibilité numérique n'est pas atteinte. Le matériel utilisé dans le domaine du calcul à haute performance est une source importante de non-reproductibilité. Depuis quelques années, nous constatons une inflation de la diversité des causes matérielles de non-reproductibilité numérique. Il est fréquent que les accélérateurs proposent des FPU qui ne respectent plus la norme IEEE 754. Ceci a été le cas de nombreux many-cœurs et multi-cœurs et plus encore de certains accélérateurs de type GP-GPU, mais aussi de certaines unités d'*Hyper-Threading*. On note aussi des divergences entre les processeurs d'architectures 32 bits et ceux en 64 bits. À un niveau encore plus bas, le parallélisme au niveau des instructions (ILP, pour *Instruction Level Parallelism* en anglais) et les techniques d'optimisation des codes au sein des microprocesseurs ne respectent plus forcément l'ordre des instructions spécifié par les programmes. Nous discuterons plus loin de l'impact du calcul en arithmétique flottante et des processeurs dont l'exécution est dite *out of order*. L'architecture actuelle des accélérateurs de type Xeon Phi est : *in order* ce qui facilite la répétabilité des expériences, alors que l'architecture suivante qui est annoncée pour les accélérateurs Intel est *out of order*. Nous y reviendrons dans la suite.

Tous les points mentionnés ci-dessus ont un impact sur la reproductibilité numérique. Diethelm (2012) a également étudié l'impact d'un nombre différent de processeurs. Taufer *et al*. (2010) ont indiqué que la reproductibilité et la stabilité des résultats de simulations de dynamiques moléculaires ne peuvent pas être garanties dans les simulations parallèles à grande échelle exécutées sur GP-GPU et sur des CPU multi-cœurs.

4.4.2. La variabilité des fréquences de calcul

Un autre point, lié au matériel, impacte la reproductibilité de l'évaluation des performances : il s'agit de la fonctionnalité permettant une fréquence d'exécution variable dans les microprocesseurs modernes. D'une exécution à l'autre du même code, avec le même contexte d'exécution, on observe des disparités significatives sur les temps d'exécution. Les temps propres à un processus sont bien distingués des temps d'*overhead* du système d'exploitation (liés par exemple à la charge d'un serveur). La principale raison vient de la fonctionnalité dite de *Turbo Boost* des microprocesseurs récents qui adaptent leur fréquence aux conditions de calcul, à la température, etc. Pour une meilleure métrologie informatique (des prises de mesures fiables et des

performances reproductibles), il convient de désactiver cette fonctionnalité au niveau des configurations BIOS (*Basic Input Output System*) des serveurs de calcul.

4.4.3. L'impact croissant des soft errors

D'autres obstacles matériels à l'obtention de résultats reproductibles se profilent pour les super-calculateurs. Le premier obstacle est lié à la latence du réseau d'interconnexion de ces machines qui est hors du contrôle des programmeurs. Le second obstacle est lié à la corruption silencieuse des données et des unités de calcul (nommée *soft errors* en anglais, même si cela n'a rien à voir avec le logiciel). Les interactions rares entre les circuits et des radiations ionisantes (ex. : particules alpha) deviennent fréquentes si l'on considère la taille et la densité de ces super-machines. L'introduction des mémoires ECC (*Error Checking and Correcting*), d'abord sur les serveurs, puis sur les micro-ordinateurs, vise à gérer précisément ce type de problème même si ces événements restent rares pour les ordinateurs de bureau et pour de petites fermes de calcul informatique (*commodity cluster*). Par contre, sur les machines de taille imposante, il est un fait qu'avec les technologies actuelles, le temps moyen entre deux pannes (*Mean Time Between Failure* [MTBF]) se mesure maintenant en heures sur les plus grosses machines du classement mondial Top500.

4.4.4. Les processeurs à mode d'exécution dans le désordre

Les compilateurs traduisent les instructions d'un code source et génèrent un code qui reflète l'ordre des instructions préconisé par le développeur. Ce qui est encore vrai au niveau des compilateurs n'est plus vrai au niveau des microprocesseurs récents. En effet, parmi les techniques récentes d'optimisation des ressources d'un microprocesseur, l'exécution dans le désordre (ou *out of order execution* en anglais) permet au microprocesseur de réorganiser l'ordre des instructions qui vont s'exécuter. De fait, ces instructions ne sont alors plus forcément exécutées dans l'ordre qui était spécifié par le développeur !

Cette approche est utilisée par les fondeurs de processeurs car elle permet, d'une part, des gains en performance et, d'autre part, des gains en consommation d'énergie en simplifiant la conception du processeur. Tout d'abord utilisée pour des processeurs à basse consommation comme les ARM et l'Intel Atom, cette option a été également choisie pour le développement de processeurs de serveurs Intel ou IBM.

Examinons de plus près les mécanismes en jeu. Les instructions sont dépendantes des données, et, pour respecter cette dépendance des données, il est essentiel que l'ordre d'exécution des lectures et des écritures ne change pas. Dans le cas contraire, les résultats des programmes ne seraient plus les mêmes. Cependant, tous les processeurs modernes sont de types super-scalaires, capables d'exécuter plusieurs instructions simultanément. Plusieurs sous-unités de calcul sont organisées en pipeline et ces processeurs sont capables de détecter l'absence de dépendances entre instructions. Avec un pipeline, il arrive que l'ordre des lectures et écritures change lorsque par exemple des lectures et écritures dans les registres se font à des étages différents du pipeline. Dans ce

cas, pour éviter qu'une instruction ne lise une donnée non encore écrite, le processeur insère ce que l'on appelle des bulles dans le pipeline. Ces bulles ont pour effet de s'assurer que l'ordre des lectures et écritures est conforme à celui spécifié par le programmeur. Les approches les plus simples d'exécution *out of order* cherchent à respecter les dépendances des données et détectent la présence de bulles dans les pipelines pour permettre des optimisations. Une optimisation simple consiste à remplacer les bulles par les instructions qui suivent celle qui est bloquée en attente de données. Par conséquent, l'ordre d'exécution des instructions et l'ordre dans lequel les résultats sont issus des sous-unités de traitement ne correspondent plus à l'ordre spécifié initialement. Dans le cadre de la répétabilité numérique, les problèmes viennent du fait que les contextes d'exécution des microprocesseurs multi-cœurs modernes varient d'une exécution à l'autre. Ces microprocesseurs mutualisent des unités entre plusieurs flots d'exécution et sont capables de détecter l'absence de dépendances entre instructions, mais le respect de l'associativité sur les calculs flottants ne se trouve plus garanti.

Lors de la conférence Supercomputing 2015 (à Austin, au Texas), nous étions invités à un atelier pour traiter de la reproductibilité numérique dans la perspective des machines exaflopiques (Hill *et al*., 2015). Le constat suivant a été fait : la répétabilité numérique était aisée avec les anciens processeurs mono-cœurs, mais l'évolution du matériel au cours des dernières décennies l'a rendue progressivement de plus en plus difficile. Elle peut encore être au rendez-vous mais souvent au prix de pertes de performances. Lors de ce workshop, il a été noté que cette perte de reproductibilité a commencé avec les processeurs utilisant les optimisations basées sur les techniques d'exécution *out of order*, mais pas uniquement. En effet, les récentes tendances architecturales proposent de plus en plus d'unités de traitement du type des accélérateurs de calcul à base de GPU (*Graphical Processing Units*) ou de FPGA (*Field Programmable Gate Arrays*). Cette hétérogénéité des architectures a accéléré un décrochage en termes de reproductibilité numérique. Si l'actuel Xeon Phi (Knight corner) proposant 240 ou 244 cœurs logiques possède des unités d'exécution *in order*, le prochain processeur Intel manycores est annoncé avec un mode d'exécution *out of order*.

4.5. Le logiciel sous-jacent

La contribution du logiciel à la qualité de la reproductibilité numérique revêt plusieurs aspects. Parmi ceux-ci, nous recensons principalement les points suivants : l'implémentation de la bibliothèque mathématique utilisée pour la précision des calculs ; les implémentations des langages de programmation ; les compilateurs, les options et les versions de ces compilateurs ; les bibliothèques associées et supplémentaires ; le type de machine virtuelle ; et même le type de système d'exploitation.

Glatard *et al*. (2015) ont illustré le problème de la non-reproductibilité dans leur étude des performances de trois logiciels d'analyses d'images cérébrales en utilisant le

compilateur gcc et des versions différentes de la bibliothèque C (glibc) embarquée dans l'exécutable, sur des systèmes d'exploitation Linux contenant une même version de la bibliothèque mathématique libmath.

Dans le cas des simulations stochastiques, la reproductibilité numérique est aussi liée au choix du générateur de nombres pseudo-aléatoires, à ses statuts d'initialisation et également, dans le cas des simulations parallèles, à la méthode de parallélisation des flux stochastiques. Pour du calcul stochastique parallèle, Reuillon *et al*. (2008) ont montré plusieurs limites concernant l'utilisation de générateurs de nombres pseudo-aléatoires. Heinrich (2004) avait auparavant détecté une liste des générateurs présentant des insuffisances dans la bibliothèque CLHEP tels que random, lrand48, randu, rndm, ranmar, ranlux, ranecu, ranarray. Ces générateurs et cette bibliothèque existent encore dans de nombreuses applications scientifiques. De même, 45 des 58 générateurs de nombres pseudo-aléatoires disponibles dans la *GNU Scientific Library* présentent des insuffisances dans le choix du schéma d'initialisation (Matsumoto *et al*., 2007). Plus récemment, nous avons étudié la non-reproductibilité des générateurs modernes de nombres pseudo-aléatoires, que ce soit sur le plan logiciel ou matériel, et nous avons mis en évidence plusieurs défaillances (Dao *et al*., 2014). Dans la section 5, nous proposerons une *short list* de quelques générateurs de qualité, dont certains éprouvés depuis deux décennies dans le calcul parallèle.

4.6. La technique de distribution et de parallélisation des flux stochastiques

Pour un calcul stochastique parallèle rigoureux, il est nécessaire de disposer de nombreux flux « indépendants », mais il faut pouvoir lire ces flux de manière parallèle. Si elle est mal réalisée, cette parallélisation est aussi une source de non-reproductibilité. Comme, avec des threads exécutés en parallèle, l'ordre des tirages aléatoires peut être différent d'une exécution à l'autre, la « parallélisation » d'un générateur de nombres pseudo-aléatoires et l'assignation de différents flux aux processeurs jouent un rôle important dans la reproductibilité des résultats. Dans Hill *et al*. (2013), nous avons présenté les limites de certaines techniques et l'inadéquation de la technique du serveur central pour atteindre la reproductibilité. Nous avons également discuté des problèmes liés au recouvrement de sous-flux qui introduisent des biais dans les résultats. Il y a près de 20 ans, Hellekalek montrait le risque potentiel de « corrélation » dans les méthodes de distribution des flux stochastiques sur différents processeurs. Le titre d'un de ses articles à la conférence PADS de 1998 « *Don't trust parallel Monte Carlo* » est toujours d'actualité (Hellekalek, 1998). Nous présenterons dans la section 5 plusieurs bonnes approches pour réaliser une distribution de flux stochastiques indépendants.

5. Quelques bonnes pratiques

Afin de reproduire complètement une simulation stochastique parallèle utilisant un système de calcul à hautes performances, nous devons nous assurer de la répétabilité de l'expérience et de la reproductibilité numérique. La première devrait permettre de

réaliser exactement l'expérience publiée par l'auteur. Si l'auteur partage sa recherche, en particulier ses codes sources, ses paramètres d'entrée et les données utilisées, une telle répétabilité peut être envisagée.

Nous proposons quatre recommandations.

5.1. Recommandation 1

Pour être en mesure de répéter une expérience de simulation stochastique parallèle, outre la disponibilité de matériel similaire, nous devons disposer des données fournies par l'auteur, notamment :

– des codes sources de la version finale qui a été utilisée pour générer les résultats publiés ;

– des données utilisées (si elles existent) ;

– des paramètres d'entrée ;

– des scripts supplémentaires ;

– d'une description des ressources utilisées (le matériel, le nombre de processeurs, le logiciel, le système d'exploitation, le compilateur, la bibliothèque de programmation, la machine virtuelle, les paquets supplémentaires) ;

– d'un guide qui explique étape par étape l'expérience, la façon de l'exécuter, la correction des bogues et la version du code ;

– des contraintes et dépendances associées à la technique informatique utilisée, des enregistrements des traitements effectués sur les données (l'outil de calcul, les tableaux de données obtenus, les outils pour créer des images ou des graphiques s'ils existent) ;

– du modèle de simulation (implémenté en version séquentielle et parallèle), des entrées et sorties du modèle, de la structure des classes, des algorithmes, etc. ;

– d'un descriptif du problème de simulation traité et également de sa mise en œuvre.

Ces données sont nécessaires à une bonne publication scientifique. En d'autres mots, la reproductibilité des expériences numériques doit permettre à une personne du domaine de télécharger toutes les données disponibles en ligne, ainsi que les informations et documents complets concernant l'expérience computationnelle. Dans l'état actuel de la recherche en informatique, cela suppose un changement dans la culture de communication des résultats scientifiques.

Avant la distribution des données et des informations, l'auteur devrait appliquer une norme de recherche reproductible – telle que Reproducible Research Standard (RRS de Stodden, 2009) – pour garantir son droit d'auteur et aussi faciliter la reproductibilité, la réutilisation et la redistribution par des collègues. Clairement, RRS n'englobe pas seulement les logiciels, mais aussi les procédures et la description des matériels nécessaires pour la réplication des expériences computationnelles. Elle assure

également que la reproduction soit effectuée sans faute par les pairs. Il est possible de partager les données en ligne avec des sites internet tels que github.com, bitbucket.org, RunMyCode.org, sodata.org ou d'autres sites similaires. RRS préconise aussi la réplication d'une expérience avec des outils du type de CDE ou ReproZip (Stodden *et al.*, 2013).

Après avoir répété une expérience, si l'on obtient les mêmes résultats que l'auteur, les résultats publiés sont considérés comme significativement plus fiables.

5.2. Recommandation 2

Cette recommandation porte sur les techniques nécessaires pour reproduire exactement les résultats d'une simulation stochastique. Cette recommandation concerne les auteurs qui doivent toujours penser à la reproductibilité lors du développement de simulations.

Les points de vigilance sont :

– la conception du programme, pour pouvoir comparer les résultats et tester la reproductibilité entre les calculs séquentiels et les calculs parallélisés, comme nous le verrons en section 5 ;

– le choix de bonnes bibliothèques et types de données mathématiques qui supportent une haute précision des calculs en conformité avec la norme IEEE 754 (Robey, 2013) ;

– l'examen attentif de l'ordre d'exécution des calculs en virgule flottante ;

– l'utilisation de techniques pour augmenter la reproductibilité de sommation en virgule flottante comme celle proposée par Kahn avec la précision composite (Taufer *et al.*, 2015) ;

– l'utilisation des meilleurs générateurs de nombres pseudo-aléatoires ;

– l'utilisation de bonnes méthodes d'initialisation de ces générateurs en séquentiel et aussi en parallèle (PRAND, clRNG, SPRNG, etc.) (L'Ecuyer *et al.*, 2016) ;

– une assignation rigoureuse des flux de nombres pseudo-aléatoires aux éléments de calcul ;

– la conception d'un programme séquentiel en pensant au parallélisme et à l'indépendance des flux stochastiques pour assurer la comparaison des résultats entre exécutions séquentielle et parallèle. Cette approche assure une reproductibilité numérique même en cas de changement du nombre de processeurs.

5.3. Recommandation 3

Avant la publication de résultats, il faut être vigilant sur l'environnement d'exécution d'une simulation. Il faut donc lancer son application, si possible, sur au minimum deux environnements d'exécution différents pour évaluer l'impact de :

– la différence dans le matériel (GP-GPU, MPPA, Xeon Phi Manycore, etc.) ;

– la différence de compilateurs et de versions ;

– la différence de systèmes d'exploitation et de machines virtuelles ;

– la différence dans le nombre de cœurs d'une exécution à l'autre.

Il faut aussi examiner l'influence des options de compilation. On se doit d'utiliser les options du compilateur garantissant le maximum de reproductibilité numérique. Même avec un compilateur Intel et un processeur de marque identique, l'utilisation des options standard de compilation peut fréquemment conduire à des résultats non reproductibles.

Enfin, il faut contrôler autant que faire se peut, les possibilités et l'impact des calculs *out of order*. Il est parfois possible de contrôler cet aspect au niveau du jeu d'instruction en assembleur. Par exemple sur les architectures IBM de type « Power PC », il existe l'instruction machine EIOIO (*Enforce In Order Input Output*) qui permet de veiller à ce que les accès et stockage cache-inhibés soient effectués dans la mémoire principale dans l'ordre spécifié par le programme.

5.4. Recommandation 4

Éviter d'utiliser trop de ressources matérielles spécifiques.

Dans le cas où le développement d'une application parallèle nécessite des matériels spécifiques, un dernier point de progression concerne l'aspect de corroboration introduit précédemment. Cela peut supposer :

– de comparer les résultats pour différentes exécutions parallèles et/ou vectorielles par rapport à une référence séquentielle ;

– d'utiliser quand c'est possible des matériels similaires de différents fournisseurs (architectures de super-calculateurs vectoriels ou à base de GP-GPU de fournisseurs différents) ;

– de comparer, quand c'est possible, les résultats obtenus par des architectures matérielles différentes, ce qui suppose d'utiliser des codes logiciels adaptés à chaque architecture mais qui, du coup, deviennent significativement différents ;

– d'utiliser différents modèles de programmation ;

– d'utiliser différents langages de programmation.

En résumé, la première recommandation suggère un changement de culture scientifique. La deuxième recommandation donne un certain nombre de conseils pour concevoir une application informatique reproductible. La troisième recommandation est de tester les applications sur au moins deux environnements d'exécution différents et de comparer les résultats obtenus. La quatrième recommandation donne quelques conseils concernant l'hétérogénéité des ressources matérielles.

6. La conception de simulations stochastiques parallèles reproductibles

6.1. Les bases « déterministes » des simulations « stochastiques »

On lit parfois dans des magazines de calcul à hautes performances qu'il est normal de ne pas avoir des résultats identiques pour des simulations stochastiques parallèles alors que les simulations parallèles stochastiques se placent dans le cadre de programmes déterministes faisant usage de générateurs simulant le hasard, mais eux-mêmes également déterministes lorsqu'ils utilisent des générateurs pseudo-aléatoires. Dans le cadre des processeurs dits *in order* (qui se font rares), le matériel à notre disposition peut être considéré comme déterministe. Les environnements de développement et les compilateurs sont encore eux aussi déterministes.

Dans ce contexte, l'approche scientifique suppose que l'on soit capable d'obtenir des résultats identiques d'une simulation à l'autre lorsque les données en entrée sont les mêmes. Ne pas être en mesure de reproduire ses résultats d'une exécution à l'autre met en évidence soit un problème matériel ou logiciel, soit un manque de rigueur que l'on aurait raison de critiquer.

6.2. Quelques bons générateurs modernes et parallélisables

6.2.1. Une short list de qualité

Les algorithmes que nous avons retenus ci-dessous proposent tous une technique fiable d'après les meilleures batteries de tests statistiques pour distribuer des nombres aléatoires dans un environnement parallèle (L'Ecuyer *et al*., 2016). Nous avons déjà recensé ces techniques dans Hill *et al*. (2013), mais seules certaines d'entre elles peuvent être appliquées sur une plate-forme de type GPU. Les PRNG issus de cette étape de sélection sont alors des candidats idéaux pour être portés sur GPU, si ce n'est pas déjà le cas, et par extension pour être intégrés dans ShoveRand. Voici ci-dessous quelques générateurs actuellement disponibles ou en cours d'intégration dans ShoveRand :

– MLFG6331_64 (Mascagni et Srinivasan, 2004). Il s'agit d'un générateur non linéaire sur 64 bits de type Multiplicative Lagged Fibonacci Generator (MLFG), et c'est d'ailleurs le meilleur générateur de la suite logicielle SPRNG (Scalable Parallel Random Number Generators) qui est très répandue dans le domaine du calcul parallèle stochastique (période 2^{124}) ;

– MRG32k3a (L'Ecuyer, 1999). L'algorithme concis a été prévu pour être implémenté directement sur des nombres réels flottants. Sa structure de données stocke uniquement six valeurs en double précision, ce qui est un atout pour les architectures de type GP-GPU. Sa période est de 2^{191} éléments. Pour des besoins en performance, son implémentation optimisée avec un compilateur C/C++ Intel moderne et pour des processeurs Intel Xéon récents reste malheureusement jusqu'à 15 fois plus lente que celle de Mersenne Twister (cf. ci-dessous). MRG32K3a reste tout à fait adapté à la réalisation de codes de simulations stochastiques parallèles lorsque la proportion de calculs associés aux tirages de nombres pseudo-aléatoires est faible ;

– Mersenne Twister. Matsumoto et Nishimura (1998) ont proposé un générateur qui est rapidement devenu une référence. Outre sa vitesse de génération pour la version la plus répandue, ce générateur propose une période de 2^{19937} qui ne souffrira pas face aux futures machines exaflopiques. Ce générateur, comme tous ceux de la famille Mersenne Twister, n'est pas adapté aux applications cryptographiques. Cette famille de générateurs s'est enrichie avec une version pour GPU (nommée MTGP pour Mersenne Twister for Graphic Processors) (Saito et Matsumoto, 2013), qui utilise les instructions vectorielles de SFMT (Simd-oriented Fast Mersenne twister). Un mini Mersenne Twister (nommé TinyMT) a également vu le jour. Tous ces générateurs peuvent être paramétrisés par la technique dite de Dynamic Creation. Cette technique à base de méta-programmation permet de créer des « petits » Mersenne Twisters (Matsumoto et Nishimura, 2000) ;

– Phylox et Threefry sont des générateurs qui ont fait une entrée remarquée à la Supercomputing Conference de 2011 en recevant le prix du meilleur article (Salmon *et al*., 2011). Salmon et ses collègues proposent trois générateurs d'inspiration cryptographique qui s'appuient sur la technique de paramétrisation pour résoudre le problème de distribution des flux stochastiques au sein des applications parallèles. La publication de Salmon *et al*. annonce qu'ils sont *crush-resistant* (cf. section 5.2.2) mais tout comme Michael Mascagni et l'un de ses doctorants, nous n'avons pas pu reproduire ces tests de manière satisfaisante et, de ce fait, nous ne pouvons conseiller que la prudence face à ces générateurs.

6.2.2. Comment pouvons-nous tester les générateurs et nos applications ?

Dans le cas de simulations séquentielles, il est déjà de bonne pratique de changer de type de générateur pour constater que le modèle ne présente pas une grande variabilité stochastique. Même si avec la qualité des générateurs qui vous sont présentés, la probabilité d'une sensibilité de ce type est faible, ce type de test est toujours intéressant pour des simulations distribuées. La visualisation des résultats de remplissage d'espace révèle parfois des surprises (même en 2 et 3 dimensions) : c'est le test dit spectral qui met en évidence ces écueils.

Les générateurs présentés ci-dessus étant de familles et de structures différentes, c'est une richesse de pouvoir en disposer pour ce type de test. Il est bon de se rappeler que dans le cadre de simulations stochastiques, le système à étudier est un tuple comportant : le générateur, la façon de l'initialiser, la méthode de parallélisation retenue pour ce générateur, mais aussi, un dernier élément de poids, la simulation qui utilise le générateur. Chacun des éléments de ce tuple a son importance dans la qualité des résultats. Si beaucoup d'applications de simulation sont robustes face à la qualité des générateurs et à leur utilisation, d'autres sont bien plus sensibles, en particulier quand elles sont déployées à grande échelle.

Avant de choisir un générateur pour des simulations stochastiques parallèles, il faut déjà que celui-ci satisfasse à la plus exigeante des batteries de tests statistiques : BigCrush de TestU01 (L'Ecuyer et Simard, 2007). Les PRNG qui passent tous ces tests

sans exception sont dits *crush-resistant*. Même si le fait d'être *crush-resistant* n'atteste en aucun cas d'une quelconque perfection aléatoire du flux considéré, c'est une propriété satisfaisante dans l'état actuel des connaissances, propriété que seulement quelques PRNG peuvent prétendre posséder.

Pour des tests parallèles, Srinivasan et Mascagni ont donné des conseils intéressants (Srinivasan *et al*., 2003) sur la base des critères proposés par Mascagni dans sa bibliothèque pour la génération parallèle de nombres pseudo-aléatoires (SPRNG). Coddington et Ko (1998) ont proposé, un peu après les critères de Mascagni (1997), un ensemble de techniques pour des tests parallèles de générateurs de nombres pseudo-aléatoires. Pour éviter les corrélations à longue portée, nous devons utiliser les meilleurs générateurs avec les techniques de distribution qui leur sont adaptées. Le lecteur pourra se reporter aux propositions faites par De Matteis et Pagnutti (1988, 1995).

6.3. Flux stochastiques indépendants pour simulations parallèles

6.3.1. La notion relative « d'indépendance »

Dans le cadre des simulations stochastiques, nous souhaitons (c'est le cas largement majoritaire) réaliser (simuler) des expériences « indépendantes ». Nous savons, du fait que les générateurs pseudo-aléatoires sont déterministes, qu'il n'y aura jamais aucune preuve mathématique de cette « indépendance », mais les meilleurs mathématiciens ont travaillé pour que nous ayons des techniques qui permettent d'approcher au mieux « l'indépendance ». L'indépendance réelle est inaccessible car la dépendance réelle se cache derrière les algorithmes de génération de nombres pseudo-aléatoires et de distribution des nombres générés. Dans la suite de cette section, nous montrons que l'affectation de flux aléatoires pour différents éléments de calcul parallèle peut être faite avec l'une des deux approches suivantes : la première propose de partitionner un flux aléatoire unique, tandis que la deuxième approche considère plusieurs flux « indépendants ». Dans ce cas, les flux « indépendants » sont obtenus par le paramétrage d'une famille de générateurs. L'essentiel de ces techniques de distribution et de parallélisation des PRNG sont décrites dans Hill *et al*. (2013) et L'Ecuyer *et al*. (2016).

6.3.2. Techniques de partitionnement d'un flux stochastique unique

L'approche la plus basique pour le partitionnement d'un unique flux de nombres pseudo-aléatoires utilise un serveur central qui gère un générateur de nombres pseudo-aléatoires unique qui offre les nombres à la demande. Nous pouvons également citer les techniques à base d'automates cellulaires booléens, qui ont été considérées pour générer des nombres pseudo-aléatoires parallèles, mais nous n'avons pas trouvé de références concernant leur utilisation dans le cadre d'applications de calcul à hautes performances, ni concernant des tests rigoureux avec les meilleures suites de tests statistiques. Une autre technique est connue sous le nom de *Sequence Splitting* (fractionnement de séquences), parfois nommée *blocking* ou *regular spacing* (espacement régulier). Le principe derrière ces noms est simple : la séquence principale est divisée en N blocs

contigus (donc sans chevauchement de séquences). Jusqu'à récemment (2008), les techniques mathématiques de saut dans la séquence d'un générateur (*jump ahead*), n'étaient pas disponibles pour les générateurs modernes avec des récurrences linéaires modulo 2 disposant de périodes gigantesques (comme celles proposées par les familles MT et WELL). Ce problème a été résolu dans Haramoto *et al*. (2008), mais en dépit de son efficacité relative, la technique mathématique proposée est toujours considérée comme lente par les spécialistes (tout du moins par rapport à ce qui peut être obtenu pour un bon générateur comme MRG32k3a avec une plus petite période) (L'Ecuyer *et al*., 2002). Le temps d'exécution de quelques millisecondes ne pose aucun problème pour une seule initialisation au début d'une section de code parallèle.

La technique dite des séquences indexées (*indexed sequence*), encore nommée technique d'espacement au hasard (*random spacing*), est une autre technique qui consiste à initialiser le générateur avec différents statuts aléatoires. Elle n'est pas sans risque pour les générateurs disposant d'une période modeste et pour les générateurs sensibles à leur état initial (cas des générateurs sensibles au *zero excess initial states*).

La dernière technique connue pour le partitionnement d'un flux unique est référencée comme technique du *Leap Frog* (saut de grenouille). Avec cette méthode, les nombres aléatoires sont distribués aux éléments de calculs comme lorsqu'on distribue un jeu de cartes à différents joueurs. Si le générateur considéré ne dispose pas de technique de sauts dans la séquence (*jump ahead*), on retombe sur une technique de type serveur central avec ses inconvénients.

6.3.3. Production de différents générateurs parallèles

Des flux stochastiques indépendants peuvent être obtenus en produisant différents générateurs indépendants, issus de la même famille. Ils sont créés par « paramétrisation ». Il faut se rappeler qu'il n'existe aucune technique mathématique prouvant l'indépendance entre deux flux. Pour la famille des Mersenne Twisters (Matsumoto et Nishimura, 1998), nous avons précédemment mentionné l'existence d'un outil qui permet de créer des générateurs les plus indépendants possibles, dans la limite des connaissances de nos meilleurs mathématiciens. Cet outil, Dynamic Creator (DC) (Matsumoto et Nishimura, 2000) génère par paramétrage des « petits » générateurs Mersenne Twister. Les générateurs à récurrences linéaires modulo 2 définis par des matrices dont les polynômes caractéristiques sont premiers entre eux sont censés être hautement indépendants. Les générateurs TinyMT et MTGP pour GP-GPU bénéficient de la même approche. Même si cette approche est considérée comme sûre d'après la communauté scientifique, nous recommandons de tester les générateurs qui en résultent. En cas d'échec, il est possible d'améliorer ces générateurs en informant les auteurs qui sont très réactifs. Cela a été fait pour MTGP où nous avons constaté que plus de 30 % des générateurs issus de DC étaient de mauvaise qualité (Passerat-Palmbach *et al*., 2010), et ce défaut a été très rapidement pris en compte par les auteurs.

La paramétrisation a également été utilisée par Michael Mascagni pour proposer des générateurs de type MLFG (*Multiplicative Lagged-Fibonacci Generators*). Une

présentation complète de cette étude a été publiée dans la revue *Parallel Computing* (Mascagni et Srinivasan, 2004). L'ensemble de ces approches est présenté en détail dans Hill *et al*. (2013), avec, d'une part, les avantages et les inconvénients de chaque technique et, d'autre part, des recommandations pour les architectures hybrides à base de GP-GPU.

6.4. Une méthode pour obtenir la répétabilité de simulations stochastiques

Il est possible, en suivant une méthode simple et rigoureuse, de réaliser des simulations stochastiques numériquement reproductibles. Ce type de reproductibilité numérique correspond à la répétabilité et c'est, comme nous l'avons dit, un cas particulier du problème de la reproductibilité scientifique en informatique. Il s'agit en effet d'être au moins capable de répliquer ses calculs et d'obtenir de manière fiable des résultats identiques pour des données en entrée identiques. Or, dans le cadre de simulations stochastiques parallèles, obtenir ce niveau de qualité avec différents contextes d'exécution (ne serait-ce qu'un nombre de processeurs différents) est déjà un challenge. Dans Hill (2015), nous proposons une méthode autorisant la répétabilité des résultats issus de simulations stochastiques parallèles de type Monte Carlo. Pour obtenir une répétabilité des résultats, nous recommandons aux concepteurs les préconisations suivantes :

– une approche de simulation par objets (Hill, 1996) ;

– l'utilisation de générateurs modernes et de qualité tels que ceux précités ;

– l'utilisation d'une bonne technique de parallélisation de flux stochastique, celle-ci devant être en adéquation avec le choix du générateur (Hill *et al*., 2013) ;

– une méthode de conception « parallèle » du programme séquentiel qui sert de référence.

Le détail de la méthode que nous utilisons est donné ci-après :

1. Concevoir tout d'abord un programme séquentiel avec son jeu de données d'entrée qui servira de référence. La clé de la reproductibilité se base sur le fait que cette conception du programme séquentiel doit se faire en pensant parallèle dès le départ. Tous les « objets stochastiques » de cette simulation susceptibles d'être exécutés de manière indépendante (et donc en parallèle) doivent être identifiés et avoir leur propre flux de nombres pseudo-aléatoires, avec un statut initial précis (le statut correspond au germe des anciens générateurs).

2. L'ensemble des flux stochastiques « indépendants » doit être produit à l'aide d'une des techniques précédemment évoquées dans cet article (soit ces flux sont produits à partir de nombreux générateurs parallèles conçus pour être « indépendants », soit ils sont issus d'un seul générateur grâce à une technique produisant de nombreux sous-flux). Comme il a été précisé ci-dessus, il convient d'archiver dans l'état initial de la simulation tous les statuts des flux stochastiques utilisés, de façon à pouvoir reproduire la même suite pour chaque objet stochastique simulé.

3. L'exécution en parallèle se basera sur une distribution des seuls objets stochastiques indépendants sur un nombre *N* d'unités de calcul (ou *Processing Elements* [PE]). Tous ces objets, bien identifiés, seront initialisés chacun sur un PE. Mais quel que soit le nombre de PE, les flux stochastiques sont propres à chaque objet et seront initialisés avec leur statut propre, c'est-à-dire avec les mêmes statuts que ceux utilisés dans la simulation séquentielle.

4. À petite échelle, vérifier que les exécutions parallèles sur différents nombres de PE donnent bien les mêmes résultats intermédiaires que l'exécution séquentielle lorsque le même jeu de données en entrée est utilisé.

5. Lorsque les résultats intermédiaires sont assurés, il convient de fixer l'ordre des calculs pour la réduction des résultats obtenus en parallèle. La réduction est une opération séquentielle, mais, comme nous l'avons précisé plus haut dans cet article, l'arithmétique flottante n'est pas associative, et pour l'obtention d'un résultat parallèle identique au résultat séquentiel, il convient de placer dans un tableau les résultats des calculs indépendants qui ont été parallélisés et de réaliser le calcul de réduction final en respectant l'ordre des opérations qui était utilisé par le programme séquentiel (par exemple le calcul d'une moyenne et de son intervalle de confiance).

L'utilisation de cette méthode permet de s'assurer que les résultats des simulations parallèles stochastiques sont reproductibles. Cette propriété est très importante pour de nombreux scientifiques travaillant dans de nombreux domaines sensibles (finance, sûreté nucléaire, médecine…).

Nous avons appliqué cette méthode avec succès dans la thèse de Schweitzer (2015) dans le cadre du Labex ClerVolc et du projet Tomuvol. L'objectif était de pouvoir simuler un phénomène physique de grande ampleur et plus particulièrement la propagation de muons atmosphériques à travers de larges édifices tels que des volcans, des pyramides… Pour réaliser cette simulation, il était nécessaire de disposer de codes parallèles particulièrement efficaces afin que le temps de calcul n'excède pas le temps de recueil des données réelles. Le travail a été réalisé à l'échelle du milliard de threads sur des architectures parallèles de serveurs à base de processeurs Intel Xeon multi-cœurs classiques (de type Ivy Bridge), mais aussi de serveurs hybrides disposant d'accélérateurs de type manycores Xeon Phi (5110P et 7120P). Une attention particulière a été portée à la reproductibilité numérique. Après nous être assurés d'une reproductibilité numérique *run to run* sur chaque type d'architecture, nous nous sommes intéressés à la reproductibilité numérique entre deux architectures matérielles différentes : Xeon classique en x86 et Xeon Phi qui dispose d'une architecture k1om. Nous avons ensuite exploré l'impact des options de compilation et évalué la divergence des résultats lorsque l'on utilise des options de compilation classiques ou agressives (celles qui demandent par exemple un profil *fast math*).

Si les développeurs ne prêtent pas une attention particulière aux options de compilation, on se retrouve avec des différences importantes, au point que deux résultats peuvent totalement diverger dans leur interprétation physique. Nous avons constaté à quel point il est essentiel de prêter une attention particulière aux options de compilation

Tableau 1. Différences relatives des valeurs et bits modifiés entre Xeon Phi et Xeon CPU sur 2 simulations identiques de muographies (Schweitzer, 2015)

Différence ↓ / Valeur →	Pos.X	Pos.Z	Dir.X	Dir.Y	Dir. Z
0 bit : reproductibilité bit-à-bit	4922	4934	4896	4975	4913
1 bit : 1.11E-16 $\leq \Delta <$ 2.22E-16	25	21	14	5	18
2 bits : 2.22E-16 $\leq \Delta <$ 4.44E-16	21	18	52	4	31
3 bits : 4.44E-16 $\leq \Delta <$ 8.88E-16	15	12	23	6	12
4 bits : 8.88E-16 $\leq \Delta <$ 1.78E-15	10	7	5	4	10
5 bits : 1.78E-15 $\leq \Delta <$ 2.25E-11	7	8	10	6	16

qui favorisent la reproductibilité numérique. Pour diminuer les écarts entre les résultats obtenus sur les deux architectures, nous avons été amenés à retenir les options de compilation suivantes (pour un compilateur Intel, nécessaire au calcul sur Xeon Phi): « *-fp-model precise -fp-model source -fimf-precision = high* ». Pour le processeur Intel Xeon Phi, nous avons également désactivé la possibilité de faire des *Fused-Multiply-Add* (FMA) matériels, comme préconisé par Intel, grâce à l'option de compilation « *-no-fma* ». Les options permettant une reproductibilité numérique ont un coût en temps de calcul (dans cette étude le facteur de ralentissement était de 1.2). Mais à quoi bon calculer plus vite pour donner des résultats faux ? En appliquant les pratiques que nous avons décrites, nous obtenons une reproductibilité numérique plus que satisfaisante. Dans le cas de ces 2 architectures Intel différentes, Corden (2014) a signalé la difficulté et même parfois l'impossibilité d'une répétabilité parfaite (ou reproductibilité bit-à-bit – *bit-to-bit reproducibility*). Cet état de fait peut être bloquant dans certaines applications où l'on trace finement des événements.

De fait, même si la répétabilité totale n'est pas au rendez-vous, nous avons cependant obtenu une répétabilité bit-à-bit entre les deux architectures pour une grande majorité des valeurs en double précision, et pour toutes les valeurs si on se restreint, pour la comparaison, à la simple précision (les calculs étant, quant à eux, effectués en double précision). Le tableau 1 ci-dessus donne les différences relatives entre les deux architectures. Le lecteur intéressé par les détails de cette étude pourra se reporter à la thèse de Schweitzer (2015) que nous avons encadrée. Dans cette étude, les écarts de reproductibilité numérique sont quantifiés bit-à-bit.

7. La reproductibilité numérique pour les futurs super-calculateurs

Nous n'aborderons pas dans cette section la reproductibilité des calculs sur des architectures de rupture telles que *The Machine* développée par Hewlett Packard Enterprise (HPE) à base de memristors et d'optronique, sur des puces neuromorphiques,

ou encore sur D-Wave ou tout autre accélérateur quantique mis à disposition à travers le Cloud par IBM. Nous nous focalisons sur des solutions disponibles pour les architectures de Von Neumann qui seront plus probablement celles qui franchiront la barrière de l'Exascale d'ici quelques années.

Les échelles et ordres de grandeur sont un obstacle, pas seulement en termes de coût de consommation électrique ou de problèmes physiques (*soft errors* ou autres). En effet, à l'échelle actuelle des meilleurs systèmes (de l'ordre de 10^7 cœurs de calculs), tous les problèmes mentionnés dans les sections précédentes se rencontrent. Une machine exaflopique proposera environ un milliard de cœurs de calcul, peut-être à l'horizon 2020. Tout comme le mentionnent Ahn *et al*. (2013) dans leurs travaux, nous pouvons comprendre que si nous voulons obtenir une reproductibilité numérique satisfaisante à cette échelle, nous avons besoin d'un ensemble d'outils communs avec des approches unifiées. Cet ensemble d'outils sera particulièrement utile pour contrôler et éliminer de nombreuses sources de non-déterminisme.

En plus des outils logiciels, l'impact des aspects numériques du calcul en virgule flottante doit être de nouveau souligné. Nous avons vu que les résultats numériques dépendent non seulement de la précision arithmétique dont nous disposons en informatique, mais aussi de l'ordre des opérations arithmétiques. L'émergence du calcul hybride (avec des accélérateurs matériels) a un impact sur ces deux aspects. Dans Bailey *et al*. (2002), nous trouvons des propositions pour obtenir une précision arbitraire adaptée au calcul haute performance. De même, dans les travaux de Jézéquel *et al*. (2014) et de Revol et Théveny (2014), nous trouvons des approches essentielles pour une arithmétique informatique qui permettrait d'accroître notre confiance dans les calculs réalisés à très grande échelle.

8. Conclusion

Cet article a pour objectif d'accroître la sensibilité de la communauté scientifique au besoin de reproductibilité dans le domaine de la simulation. Il vise à augmenter la qualité de nos publications et à permettre d'économiser le temps passé à essayer de reproduire les résultats d'études qui ne sont pas assez documentées (avec l'investissement financier que cela représente). Dans le cas des simulations parallèles stochastiques, cette reproductibilité est possible. Elle suppose cependant de traiter avec soin la distribution des nombres pseudo-aléatoires. Nous avons essayé d'identifier les principales causes de non-reproductibilité numérique. Aucune étude ou approche ne peut actuellement régler tous les problèmes rencontrés. Outre cette présentation de l'ensemble des problèmes pouvant conduire à des échecs en termes de reproductibilité numérique, nous avons avancé un ensemble de recommandations et nous avons mis en avant les apports de la reproductibilité numérique pour les études faisant usage des techniques de simulation. Ce papier étant focalisé sur les techniques de simulations stochastiques, nous avons retenu et présenté quelques-uns des meilleurs générateurs actuels pour des architectures classiques et pour des architectures de type GP-GPU. Nous avons aussi exposé rapidement les principales techniques de génération parallèle

de flux stochastiques. Nous avons rappelé dans cet article qu'il n'existe aucune preuve mathématique d'indépendance. Bien au contraire, nous sommes sûrs qu'il existe une dépendance par le choix de l'algorithme de génération et de la technique de génération parallèle. Il s'agit donc d'utiliser nos meilleures connaissances pour simuler au mieux le hasard par la génération uniforme de nombres pseudo-aléatoires et de simuler au mieux la production de flux « indépendants » pour le calcul parallèle. Là encore nous sommes pleinement dans l'activité de simulation stochastique.

Les meilleures techniques de distribution de flux stochastiques restent méconnues, et c'est aussi l'une des raisons pour laquelle nous y avons consacré une part de cet article. Ces techniques sont appelées à être de plus en plus utilisées sur les architectures parallèles les plus puissantes. Nous avons vu, notamment à cause des *soft errors* que les machines les plus performantes sont sujettes à des défaillances techniques. Cependant, la majorité des méthodes de simulation stochastique, grâce à leur approche statistique, peuvent très bien s'accommoder de ces erreurs ou pannes matérielles qui sont malgré tout de plus en plus fréquentes sur les super-calculateurs les plus imposants. En effet, les systèmes de première classe ont un MTBF inférieur à 24 heures ; sur ce type de machine, plutôt que de subir des échecs de façon répétée avec des solveurs parallèles déterministes d'équations aux dérivées partielles, il serait parfois possible et préférable de disposer d'un solveur basé sur une méthode stochastique. Ces derniers peuvent être conçus de façon plus résiliente, capable d'identifier statistiquement des résultats étranges avant de les intégrer, de travailler par étapes pour récupérer des résultats intermédiaires et d'obtenir des résultats finaux de bonne précision malgré quelques pannes.

Enfin, nous avons présenté une méthode simple, permettant de s'assurer de la reproductibilité des simulations stochastiques parallèles. Cette méthode est basée sur l'expérience de nombreuses études réalisées dans des domaines sensibles. Elle s'applique parfaitement à la majorité des simulations de Monte Carlo, et nous l'avons validée à l'échelle du milliard de threads (Schweitzer *et al*., 2015), mais elle n'est pas conçue pour des codes parallèles communicants (de type simulation multi-agents par exemple). Ce dernier cas est un champ de recherche ouvert et passionnant auquel nous nous intéressons en ce moment en ce qui concerne la reproductibilité numérique, dans un premier temps sur processeurs multi- et manycores. La simulation fait également un usage intensif des plans d'expériences qui sont certainement un des domaines les plus gourmands en calcul à hautes performances. Le fait que les expériences d'un plan puissent être reproduites est un point crucial pour l'avancement de la science.

Nous terminerons cette conclusion avec un avertissement lié à l'utilisation de machines hybrides utilisant pour le même calcul des architectures matérielles de types différents (notamment des accélérateurs de calcul). Nous avons abordé ce cas dans la thèse de Schweitzer (2015) avec des processeurs Xeon classiques et manycores (Xeon Phi) ; sans précautions, les écarts sur les résultats peuvent atteindre plusieurs ordres de grandeur. Pour ce qui concerne les GP-GPU, Taufer (2013), qui est l'un des acteurs majeurs du calcul à hautes performances, constate que ces architectures ne permettent pas facilement l'obtention d'une reproductibilité numérique satisfaisante dans son contexte applicatif.

Les nouvelles machines de première classe (projet CORAL d'IBM allant de 100 à 200 PetaFLOPS à l'horizon 2017-2018) vont proposer un espace mémoire unifié pour le calcul entre CPU et accélérateurs de type GP-GPU. Cette fonctionnalité est un atout majeur pour simplifier le développement et la portabilité des codes scientifiques. Elle est déjà présente sur les processeurs IBM Power de 8^{e} génération qui peuvent communiquer directement et de façon très efficace avec des GP-GPU Nvidia (*via* le lien NVLink). Ces nouvelles architectures hybrides sont retenues pour les plus gros systèmes de calcul depuis 2010. Il reste que cette hétérogénéité peut se révéler difficile à maîtriser en termes de reproductibilité numérique. Toujours dans le cadre de perspectives, la reproductibilité scientifique se doit de traiter également le problème de la communication des modèles, et donc de leurs spécifications, qu'elles soient formelles ou non. C'est un sujet qui est abordé dans l'article de Duboz *et al*. (2012) et qui intéresse particulièrement la communauté scientifique autour de DEVS (Discrete Event Specification). Terminons par un fait marquant l'année 2016 sur le sujet de la reproductibilité : pour la première fois dans son histoire, la conférence annuelle de super-calcul a lancé un concours pour les doctorants sur la répétabilité des calculs d'un article de recherche utilisant des clusters de calculs. Le président de cette conférence (Stephen Harrel) précise : « Nous voulons que les doctorants comprennent, dès le début de leur carrière, le rôle important que la reproductibilité numérique joue dans la recherche. »

Remerciements

Nous remercions particulièrement l'Agence universitaire de la francophonie et l'Institut francophone international pour le financement d'une thèse sur cette thématique de la reproductibilité numérique dans le cadre du calcul à haute performance.

Bibliographie

Ahn D.H., Lee G.L., Gopalakrishnan G., Rakamari Z., Schulz M., Laguna I., (2013). Overcoming extreme-scale reproducibility challenges through a unified, targeted, and multilevel toolset. In *ACM Proceedings of the 1st International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*. p. 41-44.

Coddington P.D., Ko S.-H. (1998). Random number generator for parallel computers. In *Proceedings of the 12th international Supercomputing Conference, ACM New York*. p. 282-288.

Dao V.T., Maigne L., Breton V., Nguyen H.Q., Hill D. (2014). Numerical reproducibility, portability and performance of modern pseudo random number generators: preliminary study for parallel stochastic using hybrid Xeon Phi computing processors, In *Proceedings of the 28th European Simulation and Modeling, ESM'2014*, p. 80-87.

De Matteis A.D., Pagnutti S. (1988). Parallelization of random number gemerators and long-range correlations. *Numerische Mathematik*, vol. 53, n^{o} 5. p. 595-608.

De Matteis A.D., Pagnutti S. (1995). Controlling correlations in parallel Monte Carlo. *Parallel Computing*, vol. 21, n° 1. p. 73-84.

Demmel J., Nguyen H.D. (2013). Numerical reproducibility and accuracy at exascale. In *IEEE Symposium on Computer Arithmetic (ARITH)*, p. 235-237.

Diethelm K. (2012). The limits of reproducibility in numerical simulation. *Computing in Science and Engineering*, vol. 14 n° 1, p. 64-72.

Drumond C. (2009). Replicability is not reproducibility: nor is it good science. In *Proceedings of the Evaluation Methods for Machine Learning workshop, 26th International Conference for Machine Learning 2009, Montreal, Quebec, Canada*.

Duboz R., Bonté B., Quesnel G. (2012). Vers une spécification des modèles de simulation de systèmes complexes. *Stud. Inform. Univ.*, vol. 10, n° 1, p. 7-37.

Eberhart P., Brajard J., Fortin P., Jézéquel F. (2015). High performance numerical validation using stochastic arithmetic. *Reliable Computing*, vol. 21, p. 35-52.

El Bitar Z., Lazaro D., Coello C., Breton V., Hill D.R.C., Buvat I. (2005). Fully 3D Monte Carlo image reconstruction in SPECT using functional regions. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 569 n° 2, p. 399-403.

Feitelson D.G. (2015). From repeatability to reproducibility and corroboration. *ACM SIGOPS Operating Systems Review-Special Issue on Repeatability and Sharing of Experimental Artifacts*, vol. 49 n° 1, p. 3-11.

Fomel S., Claerbout J.F. (2009). Reproducible Research. *Computing in Science & Engineering*, vol. 11, n° 5, p. 5-7.

Glatard T., Lewis L.B., Ferreira da Silva R., Adalat R., Beck N., Lepage C., Rioux P., Rousseau M.E., Sherif T., Deelman E., Khalili-Mahani N., Evans A.C. (2015). Reproducibility of neuroimaging analyses across operating systems. *Frontiers in Neuroinformatics*, vol. 9, article 12, p. 1-12.

Goldberg D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, vol. 23, n° 1, p. 5-48.

Haramoto H., Matsumoto M., Nishimura T., Panneton F., L'Ecuyer P. (2008). Efficient jump ahead for F2-linear random number generators. *INFORMS Journal on Computing*, vol. 20 n° 3, p. 385-390.

Hellekalek P. (1998). Don't trust parallel Monte Carlo! In *Proceedings of the twelfth workshop on Parallel and Distributed Simulation*. p. 82-89.

Hill D. (1996). *Object-oriented analysis and simulation*, Addison-Wesley Longman.

Hill D. (2015). Parallel random numbers, simulation, science and reproducibility. *IEEE/AIP Computing in Science and Engineering*, vol. 17, n° 4, p. 66-71.

Hill D., Passerat-Palmbach J., Mazel C., Traore M.K. (2013). Distribution of random streams for simulation practitioners. *Concurrency and Computation: Practice and experience*, vol. 25, n° 10, p. 1427-1442.

Hinsen K. (2015a). Technical debt in computational science. *Computing in Science & Engineering*, vol. 17, n° 6, p. 103-107.

Hinsen K. (2015b). The approximation tower in computational science: why testing scientific software is difficult. *Computing in Science & Engineering*, vol. 17 n° 4, p. 72-77.

Jézéquel F., Langlois P., Revol, N. (2014). First steps towards more numerical reproducibility. *ESAIM: Proceedings and Surveys*, vol. 45, 2014, p. 229-238.

Krishnamurthi S., Vitek J. (2015). The real software crisis: repeatability as a core value. *Communications of the ACM CACM*, vol. 58, n° 3. p. 34-36.

L'Ecuyer, P. (1999). Good parameters and implementations for combined multiple recursive random number generators. *Operations Research*, vol. 47, n° 1, p. 159-164

L'Ecuyer P., Simard R. (2007). TestU01: a C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, vol. 33, n° 4, p. 40

L'Ecuyer P., Munger D., Oreshkin B., Simard R. (2016). Random numbers for parallel computers: requirements and methods, with emphasis on GPUs. *Mathematics and Computers in Simulation*, Open Online Access. http://www.sciencedirect.com/science/article/pii/S0378475416300829.

L'Ecuyer P., Simard R., Chen E.J., Kelton W.D. (2002). An Object-oriented random-number package with many long streams and substreams. *Operations Research*, vol. 50, n° 6, p. 1073-1075.

Langlois P., Nheili R., Denis C. (2015). Numerical reproducibility: feasibility issues. In *NTMS'2015: 7th IFIP International Conference on New Technologies, Mobility and Security (IEEE-IFIP)*. p. 1-5.

Lazaro D., El Bitar Z., Breton V., Hill D., Buvat I. (2005). Fully 3D Monte Carlo reconstruction in SPECT: a feasibility study. *Physics in Medicine and Biology*, vol. 50, p. 3739-3754.

Maigne L., Hill D.R.C., Breton V., Reuillon R., Calvat P., Lazaro D., Legré Y., Donnarieix D. (2004). Parallelization Of Monte Carlo simulations and submission to a grid environment. *Parallel Processing Letters*, vol. 14, n° 2, p. 177-196.

Mascagni M., Srinivasan A. (2004). Parameterizing parallel multiplicative lagged-Fibonacci generators. *Parallel Computing*, vol. 30, p. 899-916.

Matsumoto M., Nishimura T. (1998). Mersenne Twister: A 623-dimensionnally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulations: Special Issue on Uniform Random Number Generation*, vol. 8, n° 1, p. 3-30.

Matsumoto M., Nishimura T. (2000). Dynamic Creation of Pseudorandom Number Generators, in Niederreiter H., Spanier J. (eds), *Monte Carlo and Quasi-Monte Carlo Methods*, Springer, p. 56-69.

Matsumoto M., Wada I., Kuramoto A., Ashihara H. (2007). Common defects in initialization of pseudo random number generators. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 17, n° 4.

Passerat-Palmbach J., Mazel C., Mahul A., Hill D. (2010). Reliable Initialization of GPU-enabled Parallel Stochastic Simulations Using Mersenne Twister for Graphics Processors. In *Proceedings of the European Simulation and Modeling Conference*. p. 187-195.

Reuillon R., Hill D., El Bitar Z., Breton V. (2008). Rigorous Distribution of Stochastic Simulations Using the DistMeToolkit. *IEEE Transactions on Nuclear Science*, vol. 55, n° 1, p. 595-603.

Revol N., Théveny P. (2014). Numerical reproducibility and parallel computations: Issues for interval algorithms. *IEEE Transactions on Computer*, vol. 63, n° 8, p. 1915-1924.

Saito M., Matsumoto M. (2013). A variant of mersenne twister suitable for graphics processors. *ACM Transactions on Mathematical Software*, vol. 39 n° 2, p. 1-20.

Salmon J.K., Moraes M.A., Dror R.O., Shaw D.E. (2011). Parallel random numbers: as easy as 1, 2, 3. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. p. 1-12.

Schwab M., Karrenbach M., Claerbout J. (2000). Making scientific computations reproducible. *Computing in Science & Engineering*, vol. 2, n° 6. p. 61-67.

Schweitzer P. (2015). *Simulations parallèles de Monte Carlo appliquées à la physique des hautes énergies pour plates-formes manycore et multicore : mise au point, optimisation, reproductibilité*. Thèse de doctorat en informatique. Université Blaise-Pascal.

Srinivasan A., Ceperley D.M., Mascagni M. (1999). Random number generators for parallel applications.*Advances in Chemical Physics: Monte Carlo methods in chemical physics*, vol. V 105, p. 13-35.

Srinivasan A., Mascagni M., Ceperley D. (2003). Testing parallel random number generators. *Parallel Computing*, vol. 29 n° 1, p. 69-94.

Stanisic L., Legrand A., Danjean V. (2015). An effective git and org-mode based workflow for reproducible research. *Operating Systems Review, Association for Computing Machinery*, vol. 49, p. 61-70

Stodden V. (2009). The reproducible research standard: Reducing legal barriers to scientific knowledge and innovation. *IEEE Computing in Science & Engineering*, vol. 11, n° 1, p. 35-40.

Stodden V. (2010). *The scientific method in practice: reproducibility in the computational sciences*. MIT Sloan Research. Paper No. 4773-10.

Stodden V., Bailey D.H., Borwein J., LeVeque R.J., Rider W., Stein W. (2013). Setting the default to reproducible: reproducibility in computational and experimental mathematics. In *ICERM Workshop report*.

Taufer M., Pardon O., Saponaro P., Patel S. (2010). Improving numerical reproducibility and stability in large-scale numerical simulations on GPUs. In *IEEE Int'l Symp. Parallel and Distributed Processing*. p. 1-9.

Vandewalle P., Kovacevic J., Vetterli M. (2009). Reproducible research in signal processing – What, why, and how. *IEEE Signal Processing Magazine*, vol. 26, n° 3, p. 37-47.

Web-références (dernières consultation le 14/04/2017)

Academy of Medical Sciences (2015). *Reproducibility and reliability of biomedical research: improving research practice*. https://www.acmedsci.ac.uk/policy/policy-projects /reproducibility-and-reliability-of-biomedical-research/.

Bailey D.H., Yozo H., Li X.S., Thompson B. (2002). *ARPREC: An arbitrary precision computation package*. Lawrence Berkeley National Laboratory. http://escholarship.org/uc/item/2rt1p2vm.

Collberg C., Proebsting T., Moraila G., Shankaran A., Shi Z., Warren A.M. (2014). *Measuring reproducibility in computer systems research*. Technical report. http://reproducibility.cs.arizona.edu/tr.pdf.

Corden M. (2014). Run-to-run reproducibility of floating-point calculations for applications on Intel Xeon Phi coprocessors (and Intel Xeon Processors). *Intel website*.https://software.intel.com/en-us/articles/run-to-run-reproducibility-of-floating-point-calculations-for-applications-on-intel-xeon.

Heinrich J. (2004). *Detecting a bad random number generator*. http://www-cdf.fnal.gov/physics/statistics/notes/cdf6850_badrand.pdf.

Hill D., Congo F., Dao V.T. (2015). *Numerical reproducibility for parallel stochastic simulation "Exascale Ready"*. *Super Computing 2015*. Austin, Texas, Numerical Reproducibility Workshop, ACM, p. 1-3. Slides available at NIST: https:// www.nist.gov/news-events/events/2015/11/numerical-reproducibility-exascale-nre2015.

Jimenez I., Maltzahn C., Moody A., Mohror K. (2014). *Redo: reproducibility at scale*. https:// www.soe.ucsc.edu/research/technical-reports/UCSC-SOE-14-12.

Robey R.W. (2013). Reproducibility for parallel programming. In *Bit-wise reproducibility workshop at SC13*. http://sc13.supercomputing.org/sites/default/files/WorkshopsArchive/pdfs/wp110s1.pdf.

Taufer M., Becchi M., Johnston T., Chapp D. (2015). Numerical reproducibility challenges on extreme scale multi-threading GPUs. In *GTC 2015*. http://on-demand.gputechconf.com/gtc/2015/presentation/S5245-Michela-Taufer.pdf.

Article reçu le : 10/10/2016
Accepté le : 03/05/2017