RECHERCHE REPRODUCTIBLE

Benjamin A. Antunes, David R.C. Hill

RECHERCHE REPRODUCTIBLE

Comment les outils informatiques et le calcul scientifique impactent bien des disciplines

Benjamin ANTUNES
Doctorant au Laboratoire d'Informatique de Modélisation et d'Optimisation des Système

David R.C. HILL
Professeur à l'Université Clermont Auvergne
LIMOS – UMR CNRS 6158
Clermont-Auvergne INP
Mines St. Etienne

Table des matières

Avant-propos

Cet ouvrage propose un état de l'art de la recherche reproductible. Il se concentre sur l'utilisation de l'informatique en science, et son impact sur la reproductibilité dans bien des domaines scientifiques. Nous étudions les divers problèmes et solutions existants, sur l'usage de l'informatique en général, et également plus particulièrement les défis techniques associés au calcul à hautes performances. Ce travail de recherche cible un public averti, ayant conscience de certaines difficultés techniques inhérentes à l'utilisation de l'informatique lors de nos recherches. Cet ouvrage s'adresse donc principalement aux chercheurs qui souhaiteraient faire un tour d'horizon de la situation concernant la recherche reproductible. Néanmoins, tout public avec quelques notions en informatique peut bénéficier de la lecture de ce livre afin de se sensibiliser à la notion de reproductibilité, ce qui peut être intéressant, surtout pour notre science « jeune » qu'est l'informatique, où ces notions sont rarement abordées durant les cursus universitaires.

Introduction

La reproductibilité est reconnue comme l'un des piliers de la science. En 1660, la Royal Society adopta la devise « Nullius in verba », qui se traduit par « ne croire personne sur parole » (Royal Society 1660). Les philosophes des sciences s'accordent à dire que la reproductibilité est l'un des critères permettant de différencier la science de la pseudoscience. Depuis les années 2010, on constate une augmentation significative de l'intérêt international des scientifiques pour la recherche reproductible. Fanelli (2018) a montré une augmentation exponentielle du nombre de publications faisant référence au thème de la crise de la reproductibilité. Nous observons un nombre croissant de revues et de conférences préoccupées par la reproductibilité de leurs articles publiés (Drummond 2018; Bajpai, Bonaventure, *et al.* 2019). Nous pouvons également remarquer la création de journaux dédiés à la reproduction d'articles (Rougier *et al.* 2017). Le domaine de la recherche reproductible est très actif et en rapide évolution. Nous avons trouvé une étude fournissant un état de l'art de la reproductibilité dans le calcul scientifique (Ivie et Thain 2018), et plusieurs livres tentant de le faire (Desquilbet *et al.* 2019; National Academies of Sciences 2019; Randall et Welser 2018). Sans minimiser la qualité des travaux précédents, nous pensons que, comme cette thématique évolue, il est pertinent de mettre à jour nos connaissances et de fournir un état de l'art actualisé sur les définitions et les technologies utilisées dans la recherche reproductible car elles ont évolué récemment. Nous voulons offrir un point de vue supplémentaire, axé sur le calcul à hautes performances (que nous noterons HPC pour « High Performance Computing » en anglais). Alors que les travaux précédemment cités

mettent l'accent sur la reproductibilité computationnelle et sont plus axés sur la méthode scientifique globale ou sur les workflows comme Ivie et Thain (2018), avec le HPC, nous sommes en première ligne où des problèmes de reproductibilité spécifiques peuvent apparaitre.

Compte tenu de l'importance croissante de la reproductibilité dans la recherche scientifique, nous essaierons de répondre à cette question : comment la recherche reproductible peut-elle être effectivement mise en place dans le domaine du calcul haute performance ? En examinant les définitions et les nuances de la recherche reproductible, nous explorons la crise multifacette de la reproductibilité qui s'étend à divers domaines scientifiques, mettant en évidence les défis spécifiques et les opportunités présentées par le HPC. Nous analysons les facteurs contribuant à la perte de reproductibilité, de la science ouverte et de la documentation à l'ingénierie logicielle et à la complexité des workflows. De plus, nous discutons des problèmes de reproductibilité uniques au HPC : les problèmes liés au calcul parallèle, la génération de nombres aléatoires dans les simulations de Monte Carlo parallèles, l'optimisation, l'hétérogénéité matérielle, et les domaines émergents du calcul quantique et de l'apprentissage automatique. L'ouvrage explore ensuite une gamme de solutions conçues pour améliorer la reproductibilité, telles que le suivi de versions, la programmation lettrée et la gestion avancée des workflows, tout en abordant également les défis spécifiques au HPC tels que la reproductibilité des calculs avec des nombres à virgule flottante et la gestion des erreurs. À travers cette étude complète, nous visons à actualiser notre compréhension de la recherche reproductible dans le HPC et à contribuer au discours actuel sur le maintien de l'intégrité du calcul scientifique à une époque où la reproductibilité de la recherche est primordiale.

Nous présenterons dans un premier temps l'importance de la recherche reproductible et les définitions des différents termes utilisés dans ce domaine. Ensuite, nous montrerons comment la crise de la reproductibilité se produit dans de nombreux domaines scientifiques. Nous discuterons également du mouvement contre la montée de la recherche reproductible. Ensuite, nous discuterons des raisons pour lesquelles nous pourrions avoir une perte de reproductibilité, en mettant l'accent spécifiquement sur le calcul à hautes performances qui est le premier impacté. Enfin, nous présenterons les solutions actuelles existantes pour résoudre ces différents problèmes. Avant de conclure, nous parlerons de plusieurs problèmes ouverts dans le domaine de la recherche reproductible pour les applications de HPC.

Chapitre 1

Définitions de la recherche reproductible

1.1. Evolution des terminologies

Trois termes principaux sont employés dans le domaine de la recherche reproductible : Reproductibilité, réplicabilité et répétabilité. Bien que définir la reproductibilité semble quelque chose d'évident, la réalité est qu'une définition consensuelle parmi les chercheurs et les différents domaines de recherche n'a été atteinte que très récemment.

Les anciennes définitions

Voici un peu d'histoire récente 2020, « l'Association for Computing Machinery » (ACM) définissait ces termes comme suit :

- **Répétabilité** : même équipe, même protocole expérimental.
- **Reproductibilité** : équipe différente, protocole expérimental différent.
- **Réplicabilité** : équipe différente, même protocole expérimental.

Dans ces définitions, comme l'a déclaré Drummond (2009), « Reproducibility requires change, replicability avoids it », qui peut se traduire par « la reproductibilité exige des changements, la réplicabilité les évite ». La reproductibilité implique qu'une équipe différente, appliquant une méthode ou une expérience différente pour la même question scientifique, obtienne les mêmes conclusions scientifiques, renforçant ainsi la découverte. D'autre part, la réplicabilité vise à ce qu'une équipe différente atteigne les mêmes résultats avec une certaine précision, en utilisant les artefacts de l'article de la première équipe. Dans la littérature,

les auteurs utilisaient parfois le mot « reproductibilité » pour se référer à la « réplicabilité », et vice versa. Conseillée par la NISO (National Information Standards Organization), l'ACM a changé ses définitions après 2020, échangeant les définitions entre reproductibilité et réplicabilité. La principale raison en était le besoin d'une meilleure correspondance avec la pratique des autres domaines de recherche.

Les définitions actuelles

Le fait que l'informatique était l'un des seuls domaines à utiliser ces définitions était un argument pour suivre les conseils de standardisation de la NISO; l'ACM a donc changé ses définitions de reproductibilité et de réplicabilité. Voici les définitions de 2020, et actuelles, pour les deux termes :

- Reproductibilité (Équipe différente, même expérience) : la mesure peut être obtenue avec la précision déclarée par une équipe différente utilisant la même procédure de mesure, le même système de mesure, dans les mêmes conditions de fonctionnement, au même endroit ou dans un endroit différent lors de plusieurs essais. Pour les expériences informatiques, cela signifie qu'un groupe indépendant peut obtenir le même résultat en utilisant les propres artefacts de l'auteur.
- Réplicabilité (Équipe différente, expérience différente) : la mesure peut être obtenue avec la précision déclarée par une équipe différente, un système de mesure différent, dans un endroit différent lors de plusieurs essais. Pour les expériences informatiques, cela signifie qu'un groupe indépendant peut obtenir le même résultat en utilisant des artefacts qu'ils développent complètement indépendamment.

- Répétabilité (Même équipe, même expérience) : la mesure peut être obtenue avec la précision déclarée par la même équipe en utilisant la même procédure de mesure, le même système de mesure, dans les mêmes conditions de fonctionnement, au même endroit lors de plusieurs essais. Pour les expériences informatiques, cela signifie qu'un chercheur peut reproduire de manière fiable son propre calcul.

Pour illustrer ce problème de non-consensus des définitions, nous pouvons examiner la littérature. Nous nommons D1 le jeu de définitions proposées par l'ACM avant 2020, et D2 le jeu de définitions adoptées par l'ACM après 2020. La plupart des articles ne reprennent pas l'ensemble des trois termes : reproductibilité, répétabilité et réplicabilité.

Dans un article discutant de la reproductibilité en informatique (Hinsen 2014), Konrad Hinsen se rapproche des définitions D1. Pour l'article décrivant la création de leur journal ReScience, Rougier *et al.* (2017) utilisent des définitions en phase avec le jeu D2. Dans une présentation en 2017 (Hinsen 2017), Konrad Hinsen modifie sa vision des termes présentée en 2014 pour passer au sens proposé par le jeu D2. Dans leur article, Stanisic *et al.* (2015) se rapporte aux définitions D1. L'article de Cohen-boulakia *et al.* (2017) sur le thème des sciences de la vie est dans la catégorie D1. Drummond (2009) plébiscite le jeu de définitions D1. Le papier étudiant la reproductibilité de Collberg et Proebsting (2016) applique les définitions D1. Gundersen et Kjensmo (2018) utilise le jeu D1, il en est de même pour Bajpai *et al.* (2019) dans le domaine des réseaux. La National Academies of Sciences Engineering, and Medicine (2019) utilise le jeu de définitions D2 avec un fort impact (plus de 600 citations à ce jour). L'état de l'art le plus récent sur le thème de la reproductibilité des

workflows en calcul scientifique (Ivie et Thain 2018) utilise également le jeu de définitions D1. Stodden a utilisé en 2011 (Stodden 2011) le jeu D1, puis est passée au jeu D2 en 2014 (Stodden *et al.* 2014). Plusieurs articles utilisent les trois termes reproductibilité, répétabilité et réplicabilité, mais la plupart utilisent la notion de « recherche reproductible » sans apporter explicitement de nuances aux différentes notions. C'était le cas dans notre article de 2013 (Hill *et al.* 2013), nous utilisions sans distinction le terme de reproductibilité au sens large de la philosophie des Sciences. Avec plus d'expérience dans le domaine, nous avons utilisé le jeu de définitions D1, dans (Hill 2015) et (Hill *et al.* 2017), où la reproductibilité fait référence à l'obtention de la même conclusion scientifique en réalisant potentiellement une autre expérience et la répétabilité le fait d'avoir les même résultats bits à bits pour pouvoir déboguer. Enfin dans (Hill 2022), nous utilisons le jeu de définitions D2 en conformité avec les avancées de l'ACM et de la « National Academies of Sciences, Engineering, and Medicine ».

Dans notre petit échantillon, qui comprend principalement des articles publiés par des auteurs actifs du domaine et bien cités, nous avons identifié huit articles qui utilisent les définitions fournies par l'ACM avant 2020 (D1) et qui ont été publiés entre 2009 et 2019. Quatre articles utilisent les nouvelles définitions de l'ACM (D2) dans un intervalle de temps similaire. De notre point de vue en tant qu'informaticiens, il semble que la majorité des travaux publiés que nous pouvons voir s'appuient sur les définitions obsolètes de l'ACM (avant 2020, D1). Une étude très intéressante de Barba (2018) a étudié quel champ scientifique utilisait plutôt une définition ou l'autre. Avec cette étude, nous réalisons que l'informatique était l'un des rares domaines à utiliser les définitions données au début de cette section (D1).

	Définitions ACM avant 2020 (anciennes)	Définitions ACM après 2020 (actuelles)
Reproductibilité	Equipe différente, expérience différente	Equipe différente, même expérience
Replicabilité	Equipe différente, même expérience	Equipe différente, expérience différente
Répétabilité	Même équipe, même expérience	Même équipe, même expérience
Classification des définitions utilisées par des auteurs du monde de la recherche reproductible dans leurs articles	(Hinsen 2014), (Stanisic *et al.* 2015), (Cohen-Boulakia *et al.* 2017), (Drummond 2009), (Collberg and Proebsting, 2016), (Gundersen and Kjensmo, 2018), (Bajpai, Brunstrom, *et al.* 2019), (Ivie and Thain, 2018), (Stodden 2011), (Hill *et al.* 2013), (Hill 2015), (Hill 2019), (Hill *et al.* 2017)	(Rougier *et al.* 2017), (Hinsen 2017), (National Academies of Sciences 2019), (Stodden *et al.* 2014) (Hill 2022)

Table 1: Tour d'horizons des définitions au sein de la recherche reproductible

Les informations précédentes sont résumées dans la Table 1. Néanmoins, pour continuer définir précisément un concept, il est impératif de considérer des perspectives au-delà de son domaine immédiat (pour nous, l'informatique). Certaines définitions de la recherche reproductible sont spécifiques à chaque domaine, ce que nous voulons éviter dans le contexte de la reproductibilité, puisque nous voulons

standardiser autant que possible entre les disciplines. Nous observons une tendance à adopter les nouvelles définitions parmi les auteurs contribuant activement au domaine de la recherche reproductible, tel que Konrad Hinsen ou Victoria Stodden, dans cet objectif de standardisation.

Les nouvelles définitions de l'ACM correspondent avec les définitions de l'Académie Nationale des Sciences, de l'Ingénierie et de la Médecine, qui définissent « La reproductibilité est l'obtention de résultats cohérents en utilisant les mêmes données d'entrée; étapes de calcul, méthodes et code; et conditions d'analyse », et « La réplicabilité est l'obtention de résultats cohérents à travers des études visant à répondre à la même question scientifique, chacune ayant obtenu ses propres données » (National Academies of Sciences 2019). Les articles édités avec le soutien de l'ACM peuvent obtenir des « badges », suivant le niveau de disponibilité des artefacts et de reproductibilité de l'article (Badges ACM).

Le dernier terme à discuter est la répétabilité. Il a suscité moins de controverse que les deux autres. Cependant, dans les articles de recherche en informatique, nous trouvons parfois une confusion entre répétabilité et reproductibilité. Nous discuterons de son importance plus tard.

Cependant, d'un point de vue philosophique, la reproductibilité est le seul terme qui peut englober toutes les différentes notions. Karl Popper, le célèbre philosophe des sciences, a discuté de la logique derrière la découverte scientifique au début du 20ème siècle, d'abord en allemand (1934/35), puis en anglais en 1959. Nous disposons maintenant d'une réédition récente (Popper 2005). Popper a défini la reproductibilité comme un critère permettant de distinguer la science et la pseudo-science (Hill 2019). L'ajout d'autres termes tel que réplicabilité ou répétabilité est toujours utile, car cela permet de décrire avec une granularité plus fine ce dont nous parlons dans nos articles. Plus centré sur le terme de

reproductibilité, Goodman *et al.* (2016) ont proposé trois autres définitions qui pourraient convenir et englober la définition standard :

- La reproductibilité des méthodes : se réfère à la capacité de reproduire fidèlement les procédures expérimentales et computationnelles, en utilisant les mêmes données et outils, afin d'obtenir des résultats cohérents. Cela s'aligne avec la définition révisée de la reproductibilité de l'ACM, ou peut-être de la répétabilité.
- La reproductibilité des résultats : concerne la génération de résultats cohérents dans une nouvelle étude grâce à l'utilisation des mêmes méthodes expérimentales. Cela correspondrait également à la nouvelle définition de la reproductibilité de l'ACM. Cependant, dans la définition de la reproductibilité, nous utilisons exactement les mêmes matériaux (artefacts) pour générer les mêmes résultats.
- La reproductibilité inférentielle : implique d'atteindre des conclusions qualitativement similaires soit par une réplication indépendante d'une étude, soit par la ré-analyse des résultats de l'étude originale. Cela correspondrait à la nouvelle définition de la réplicabilité de l'ACM.

Avec ces définitions, nous conservons le terme principal de reproductibilité, mais en ajoutant un contexte à son utilisation. Cependant, nous considérons que les définitions de l'ACM post 2020 sont désormais la norme à laquelle les auteurs devraient se conformer, dans un objectif de standardisation. On constate néanmoins que de nombreuses définitions ont encore émergé pour essayer de définir ce qu'est la reproductibilité.

1.2. A propos de l'importance de la recherche reproductible

L'une des origines du mouvement de la recherche reproductible en science computationnelle a été l'initiative de Claerbout en 1992 (Claerbout et Karrenbach 1992). Buckheit et Donoho, en 1995, l'ont synthétisé avec la célèbre citation : « *Un article en informatique dans une publication scientifique n'est pas en soi la recherche, c'est simplement une publicité pour la recherche. La véritable recherche est l'environnement complet de développement logiciel et l'ensemble complet des instructions qui ont généré les figures* ». Marwick, en 2015, a souligné le défi posé par nos programmes informatiques : ils agissent comme des boîtes noires. Aujourd'hui, presque toute recherche en science implique l'utilisation d'un ordinateur. Dans un passé pas si lointain, il était nécessaire de faire soi-même tous les calculs, toutes les transformations, l'utilisation des données, etc., et décrire précisément la procédure dans un article de recherche que l'on allait publier. Cela permettait de reproduire les articles assez simplement. Aujourd'hui, des couches logicielles plus ou moins complexes cachent de nombreux éléments. Nous ne savons souvent pas exactement ce qui se passe, et il devient beaucoup plus compliqué de reproduire des résultats publiés par d'autres, si nous n'avons pas la même machine avec la même pile logicielle disponible. L'ordinateur devient un instrument de recherche à part entière, et bien plus encore... Par conséquent, il devrait subir le même contrôle de qualité (un travail méticuleux de métrologie) que les biologistes ou physiciens appliquent à leurs instruments.

Comme le dit Popper, la reproductibilité est obligatoire pour l'avancement de la science. Nous devrions être capables d'utiliser les artefacts (code, données, etc.) d'un papier publié pour rejouer l'expérience et obtenir les mêmes résultats. Nous avons besoin des artefacts du papier de recherche pour prévenir les erreurs ou, beaucoup moins fréquemment, les fraudes. De plus, la reproductibilité implique qu'une équipe de

recherche indépendante mène une expérience uniquement basée sur les informations fournies par l'équipe de recherche originale. La capacité à faciliter la reproduction d'un article que l'on publie augmentera la confiance dans les résultats publiés. Notamment, d'autres chercheurs peuvent maintenir un niveau d'objectivité plus élevé, car ils n'ont aucun intérêt à exagérer la performance d'une méthode qu'ils n'ont pas développée eux-mêmes. En outre, ces chercheurs peuvent ne pas partager les mêmes préconceptions et connaissances tacites de l'équipe initiale qui a effectué la recherche. De plus, les variations dans les configurations matérielles et logicielles parmi différents chercheurs aident également à contrôler les variables de bruit associées au matériel et aux logiciels auxiliaires, ainsi qu'aux connaissances implicites et préconceptions. Mais ce dernier point n'est vrai que si l'on considère que la reproductibilité est atteinte avec une précision imparfaite, et non avec des résultats identiques au bit près (dans le cadre de l'informatique), qui pourraient être l'objectif de certains auteurs et aussi une exigence pour le débogage.

Concernant la répétabilité, la définition peut encore varier parmi les domaines scientifiques. En effet, en informatique, nos machines sont conçues pour être déterministes (à l'exception des machines quantique, qui sont par « essence » stochastiques). En informatique, nous voulons obtenir des résultats identiques au bit près d'une exécution à l'autre lorsque nous sommes sur la même machine pour le même programme dans le même environnement logiciel avec les mêmes jeux de données. Ce point est considéré comme acquis par de nombreux scientifiques, et malheureusement ce n'est pas toujours le cas, en particulier lorsqu'il s'agit de calcul haute performance. Cependant, la répétabilité bit à bit reste essentielle pour le débogage et pour la confiance dans l'utilisation de nos ordinateurs déterministes. La répétabilité est une véritable préoccupation

pour les chercheurs qui sont encore conscients du débogage car ils continuent à faire du développement informatique, mais cette quête peut être particulièrement difficile avec le calcul parallèle, à tel point qu'on la compare parfois à une quête du « Graal ». Assurer un débogage parallèle fiable nécessite une répétabilité avec des résultats identiques au bit près. Et en ce sens, nous ne sommes pas entièrement d'accord avec la définition de l'ACM qui ajoute que les résultats sont identiques avec une marge de précision. Cela est dû au fait que la définition de la répétabilité de l'ACM provient du vocabulaire international de métrologie. À notre avis, cette définition est parfaitement correcte pour l'informatique quantique, mais pas pour l'informatique déterministe classique où nous avons vraiment besoin de résultats identiques au bit près pour déboguer correctement. Dans le cas de l'informatique déterministe classique, la définition est valide en précisant que la marge de précision est nulle afin de n'autoriser aucune différence.

Enfin, la réplicabilité est également obligatoire pour la science. En effet, l'idée que plus une hypothèse scientifique est répliquée dans le monde entier (avec différentes équipes de recherche et différentes méthodes ou expériences), plus l'hypothèse devient forte, et plus elle sera acceptée par la majorité les scientifiques.

Nous pouvons voir les trois termes des définitions de l'ACM comme se situant à différents niveaux. La répétabilité se situe au niveau de l'auteur, qui doit déboguer ou refaire sa propre expérience. La reproductibilité se situe au niveau de l'article : d'autres chercheurs pourraient vouloir s'appuyer sur ce papier pour construire leur propre recherche, et poursuivre l'objectif d'améliorer les connaissances en évitant de « réinventer la roue ». Atteindre cet objectif implique que les articles soient publiés avec tous leurs artefacts, et cela améliorera la confiance dans les

résultats publiés. Enfin, la réplicabilité se situe au niveau de la Science : différentes équipes de recherche, réalisant différentes expériences, mais obtenant la même conclusion scientifique. Ceci est obligatoire pour valider une hypothèse scientifique.

Nous pouvons voir que toutes ces notions sont cruciales pour la communauté scientifique. Il y a une montée d'intérêt pour la recherche reproductible qui peut désormais être observée dans les conférences et les revues scientifiques. La création des journaux dédiés à la reproduction des résultats des articles publiés est également un grand pas pour favoriser la recherche reproductible. Dans son article (Drummond 2018), Drummond, même s'il ne l'approuve pas, constate le fait que de nombreuses conférences (AAAS, AMP, ENAR, NSF, SIAM-CSE, SIAM-Geo) et journaux, dans le domaine de l'apprentissage artificiel, se préoccupent de plus en plus de la recherche reproductible. Stodden et ses collègues évaluent l'efficacité d'une politique de journal exigeant des auteurs qu'ils rendent disponibles les données et le code après publication (Stodden *et al.* 2018). Sur un échantillon aléatoire de 204 articles scientifiques publiés dans la revue à fort impact (Science), après la mise en œuvre de cette politique, les artefacts n'ont été obtenus que pour 44 % de l'échantillon. Ensuite, les résultats ont été reproduits avec succès pour 26 %. Cette politique des journaux apporte une amélioration par rapport à l'absence de celle-ci, mais elle reste cependant insuffisante pour garantir la reproductibilité. Stodden et ses collègues estiment que les incitations des conférences et des revues améliorent la reproductibilité, et qu'il faut continuer de travailler dans cette direction.

Chapitre 2

Crise de la reproductibilité : quelques exemples

La crise de la reproductibilité en science est désormais un phénomène mondial et largement transdisciplinaire qui contribue à la méfiance de la société envers le monde de la recherche. En 2016, Baker (2016) a publié une enquête auprès de 1576 scientifiques pour connaître leur opinion sur le fait que nous sommes actuellement confrontés à une crise de reproductibilité. Au total, 90 % des répondants pensent qu'il existe une crise de reproductibilité significative ou légère. Seuls 3 % sont convaincus qu'il n'y a pas de crise du tout. Cette étude souligne le consensus au sein de la communauté scientifique qu'une crise de reproductibilité est bien présente, couvrant diverses disciplines.

La littérature existante offre une myriade d'exemples, tant théoriques qu'empiriques, qui mettent en lumière la crise de la reproductibilité. En médecine, nous avons le célèbre titre provocateur de Ioannidis, un éminent épidémiologiste. L'un de ses articles majeurs dans ce domaine était intitulé : « *Pourquoi la plupart des résultats de recherche publiés sont faux* » (Ioannidis 2005). Dans cet article très cité, il discute des failles statistiques qui pourraient affecter les résultats publiés. Errington *et al.* (2021) présentent les résultats de reproductibilité d'études sur le cancer, montrant un taux de succès de seulement 46 % sur 112 tentatives. Begley et Ellis (2012) ont eux aussi lancé une alerte sur la reproductibilité dans la recherche sur le cancer. Ce taux de 46% peut sembler relativement optimiste si l'on considère (Ioannidis 2015), qui affirme que 85 % des financements de la recherche sont gaspillés. Eklund *et al.* (2016) ont trouvé

un problème significatif avec les études par IRM, indiquant que de nombreux articles pourraient avoir rapporté de faux résultats. Open Science Collaboration (2015) a révélé un problème similaire dans les articles de psychologie, avec des taux de reproductibilité allant de 30 % à 50 %. Dans le domaine des neurosciences, Topalidou *et al.* (2015) n'ont pas pu reproduire un modèle et ont dû passer trois mois à le ré-implémenter.

Nous avons vu que le domaine médical est fortement impacté par cette crise de la reproductibilité, y compris des domaines tels que le domaine pharmaceutique, les dispositifs médicaux (IRM), la psychologie, la recherche sur le cancer et les neurosciences. Cependant, comme mentionné précédemment, aucun domaine n'est exempt de ce problème. Nous aurions pu considérer l'informatique comme une science exacte, grâce à ses machines déterministes, et ne faisant pas face à ce problème. Néanmoins, l'informatique n'est pas à l'abri de la crise de reproductibilité; en réalité, elle en est en partie l'instigatrice. Une étude approfondie de Collberg et Proebsting (2016) sur la reproductibilité des articles en informatique a donné des résultats plutôt mauvais, avec seulement environ 30 % des articles de recherche reproductibles sur 601 examinés. Manninen *et al.* (2017) ont tenté de reproduire quatre modèles de « *L'excitabilité calcique dans les astrocytes* », et 3 sur 4 modèles manquaient d'informations essentielles, avec 2 sur 4 modèles ayant des équations incorrectes. Même après avoir corrigé les différents modèles, ils n'ont pas produit des résultats cohérents entre eux. Mesnard et Barba (2017) ont travaillé sur la mécanique des fluides, et il leur a fallu trois ans pour reproduire les résultats obtenus à partir de différentes versions leurs propres codes et outils. L'intelligence artificielle n'est également pas exempte de ce problème, comme l'ont démontré Gundersen et Kjensmo (2018). Dans le

domaine des réseaux, Kurkowski *et al.* (2005) ont trouvé que moins de 15 % des articles sur les simulations de réseaux MANET étaient reproductibles. En traitement d'image, Kovacevic (2007) a étudié 15 articles publiés dans son domaine, et a constaté qu'aucun algorithme présenté n'était soutenu par un code, et que seulement 33 % des données étaient disponibles. Vandewalle *et al.* (2009) ont examiné 134 articles également dans le domaine du traitement d'image, constatant que 9 % des articles avaient un code disponible, et 33 % pour les données.

Cela souligne que l'informatique, à travers ses différents sous-domaines, n'est pas à l'abri des défis de la reproductibilité. Ces points soulignent l'importance d'établir une recherche reproductible. Mettre l'accent sur la nécessité d'une recherche reproductible peut prévenir les fraudes et les scandales potentiels. Par exemple, Reinhart & Rogoff, spécialistes mondiaux de l'économie, ont affirmé en 2010 qu'augmenter la dette d'un pays au-delà de 90 % du Produit Intérieur Brut (PIB) d'un pays arrêterait sa croissance économique. Cette affirmation a conduit de nombreux pays occidentaux, comme les États-Unis, à adopter des politiques d'austérité. Cependant, Herndon *et al.* (2014) ont ensuite prouvé que l'étude était erronée. Reinhart et Rogoff avaient exclu des données qui contredisaient leurs conclusions et commis des erreurs de calcul. Puisqu'ils ont partagé le code et les données (un fichier Excel), il a été possible de vérifier leur travail. Cela plaide fortement en faveur d'une recherche reproductible et d'une science ouverte (partage de toutes les données liées à un article). La seule façon de démontrer qu'un article est faux est d'avoir accès à ses artefacts. Un autre exemple décrit dans (Miller 2006) concerne Geoffrey Chang, qui a eu un impact significatif sur le domaine de la structure protéique des bactéries résistantes aux antibiotiques. Cependant, plusieurs années plus tard, ses résultats se sont avérés faux en raison de la

découverte d'une erreur de programmation dans les outils internes de Chang.

Récemment, la crise du Covid19 a fortement augmenté la sensibilisation des citoyens non scientifiques à l'importance de la recherche reproductible. Nous avons été témoins de scandales dans le contexte du Covid-19, tels que la rétractation de deux articles du Lancet et du New England Journal of Medicine (Piller, Servick 2020). Les deux études ont influencé la politique internationale sur l'utilisation, ou non, de certains médicaments dans la cadre de la pandémie et ont dû être rapidement rétractées. Un autre cas était le modèle Covid de Neil Ferguson (Ferguson *et al.* 2020), pour lequel Pouzat (2022) a écrit un article humoristique, soulignant le scandale causé par la non-publication du code initial de Ferguson. Ce modèle avait déjà influencé les politiques internationales sur les mesures de confinement, en particulier en Angleterre. Sous la pression internationale, principalement des États-Unis, Neil Ferguson a publié une version révisée de son code et il a ensuite été rapporté comme gravement défectueux (Zolfagharifard et Boland 2020) menant à une pétition sur GitHub (Holmes 2020) pour retirer le code afin d'éviter son utilisation comme base pour d'autres modèles épidémiologiques. La pression financière, en particulier en médecine à l'échelle mondiale (Abbasi 2020), peut fortement diminuer la qualité scientifique, car sans reproductibilité, les entreprises se rapprochent étroitement de la pseudoscience, selon les critères énoncés par Karl Popper. Le financement et les conflits d'intérêts gangrènent ces situations, dans un cadre où nous avions besoin d'une collaboration publique internationale rapide. Iqbal *et al.* (2016) ont déclaré que « *Les articles publiés dans des revues dans la catégorie de la médecine clinique par rapport à d'autres domaines étaient presque deux fois plus susceptibles de ne pas inclure d'informations sur le*

financement et d'avoir un financement privé ». De plus, il est reconnu que les articles avec un financement industriel ou des auteurs industriels sont moins susceptibles de partager des codes et des données (Collberg *et al.* 2015), ce qui conduit au fait que nous ne pouvons pas accorder toute notre confiance aux articles industriels, si nous n'avons pas accès aux artefacts des publications proposées. Une initiative Française du Ministère des Solidarités et de la Santé est à remarquer, la mise en ligne des informations concernant les conventions, les rémunérations et les avantages liant des entreprises et des acteurs du secteur de la santé.

Chapitre 3

Des oppositions au mouvement de la recherche reproductible

Cependant, tous les auteurs ne sont pas en accord avec le mouvement de la recherche reproductible. Dans son article de 2018 (Drummond 2018), Drummond affirme que le partage du code source utilisé pour produire un article n'est pas nécessaire. Il pense que les chercheurs sont forcés de le faire pour éviter d'être mal vus, mais que cela n'a aucun intérêt pour la Science. Pour lui, le mouvement de la recherche reproductible n'est pas basé sur des faits mais seulement sur des sentiments. Il ajoute que l'obligation de fournir le code source conduira à l'acceptation, par les journaux ou les conférences, d'articles sur la base de critères techniquement faibles et que, pour lui, la fraude a toujours existé et n'a jamais posé de problème significatif. Cependant, Drummond soutient toujours le concept de science ouverte. Nous ne sommes pas d'accord avec les déclarations de Drummond : partager le code et les données est maintenant devenu simple avec la multitude d'outils à notre disposition. De plus, pourquoi devrions-nous automatiquement faire confiance à un article publié ? Nous devrions pouvoir vérifier que ce qui a été publié n'est pas entaché d'erreurs. Un cas dont nous avons précédemment discuté a montré comment une erreur sur Excel (au minimum) dans un article de recherche invité publié par des scientifiques de renom, a impacté les politiques économiques de nombreux pays (Herndon *et al.* 2014). S'il est heureusement très rare d'observer des fraudes, les humains peuvent toujours commettre des erreurs. Il existe par ailleurs un potentiel pour que les auteurs sélectionnent volontairement les données qui les arrangent, ou

commettent des erreurs involontaires. En effet, un nombre accru de relecture et de reproduction ne peut qu'améliorer la détection de telles erreurs. Un code sans bug est un code qui n'a pas été suffisamment testé. Enfin, concernant l'affirmation selon laquelle les fraudes n'ont pas posé de problèmes significatifs, nous pensons que cela est sujet à débat. De plus, en ce qui concerne la recherche financée par des fonds publics, vraisemblablement financée par les contribuables, il semble y avoir un impératif éthique pour assurer une accessibilité complète pour les citoyens aux résultats. Dans un deuxième article, Drummond (2019) critique fortement la priorisation des reproductions d'articles par rapport à la nouveauté. Cela va à l'encontre du sentiment prédominant, et par conséquent, il s'oppose aux politiques actuelles des journaux et des conférences qui tendent vers la volonté de publier des articles plus reproductibles. Il y a aussi Fanelli (2018) qui affirme que la crise de la reproductibilité est largement exagérée, voire fausse, et que pour lui, ce récit est nuisible car il démotive les jeunes chercheurs. Pourtant, être rigoureux fait partie de notre travail, et nous ne voyons pas pourquoi cela dissuaderait les jeunes chercheurs.

Un grand nombre de chercheurs sont fortement en désaccord avec les voix s'opposant au mouvement de la recherche reproductible, bien qu'utiles pour remettre en question la pertinence de telle ou telle approche, que nous avons citées ci-dessus (Fanelli et Drummond en 2018 et 2019). Stodden *et al.* (2013), Bajpai *et al.* (2017), Pouzat (2022) dans son dialogue humoristique, Rougier, Hinsen et Barba, pour ne citer qu'eux, via la création de leur journal dédié à la reproduction d'articles (Rougier *et al.* 2017), National Academies of Sciences (2019), Ten Hagen (2016), et d'autres, défendent l'idée que les journaux devraient encourager la recherche reproductible. Barba (2018) est beaucoup moins indulgente

envers Drummond, déclarant que l'écart entre les définitions de la reproductibilité et de la réplicabilité du domaine de l'informatique par rapport aux autres domaines scientifique était dû à Drummond, après son article de 2009 : « *Ils ont [les chercheurs en informatique], à leur tour, basé leurs définitions sur le travail emphatique mais essentiellement erroné de Drummond (2009).* ». Nous croyons que le scepticisme est une marque de fabrique d'un scientifique compétent. Promouvoir une culture du doute et de scepticisme est crucial, mais il est nécessaire de faire preuve de prudence, car l'utilisation abusive du doute « scientifique » a également été employée pour entraver la reconnaissance de découvertes révolutionnaires, comme ce fut le cas avec le lien entre les cigarettes et le cancer du poumon.

Chapitre 4

Pourquoi pouvons-nous perdre la reproductibilité ?

Dans notre exploration des défis liés à la reproductibilité dans la recherche informatique, nous constatons que les raisons de pertes de reproductibilité sont multifactorielles, telles qu'illustrées dans la Figure 1. La perte de reproductibilité peut provenir de divers facteurs qui sont catégorisés sous les rubriques de l'informatique scientifique et du calcul à hautes performances, chacun influencé par le contexte scientifique et le chercheur en lui-même. Dans l'informatique scientifique, les problèmes de reproductibilité surviennent principalement, dans un contexte scientifique « indépendant » du chercheur, en raison de différences dans les environnements logiciels, les workflows, une culture scientifique encourageant ou non certaines bonnes pratiques et le degré d'ouverture de la science. Ces facteurs sont complétés par des problèmes pouvant venir du chercheur en lui-même tel que son niveau technique en génie logiciel, la qualité de la documentation et des statistiques. De l'autre côté, le calcul à hautes performances fait face à son propre ensemble de défis techniques, indépendamment de l'engagement des chercheurs, nous pensons aux erreurs silencieuses (particules cosmiques etc.), au bruit actuel sur les ordinateurs quantiques, mais également aux compétences techniques que doivent acquérir les chercheurs pour maîtriser l'utilisation parallèle de générateurs de nombres pseudo-aléatoires (Hill *et al.* 2013) ou encore les techniques d'optimisation et les subtilités du calcul parallèle liées à l'évolution des architectures matérielles. Ces éléments soulignent collectivement la nature complexe et stratifiée de la perte de

reproductibilité, qui peut à la fois dépendre du niveau technique du chercheur, de sa volonté à mettre en place certaines bonnes pratiques favorisant la recherche reproductible, mais également un contexte scientifique facilitant ou non la mise en place d'une recherche reproductible, certains facteurs étant extérieurs au chercheur.

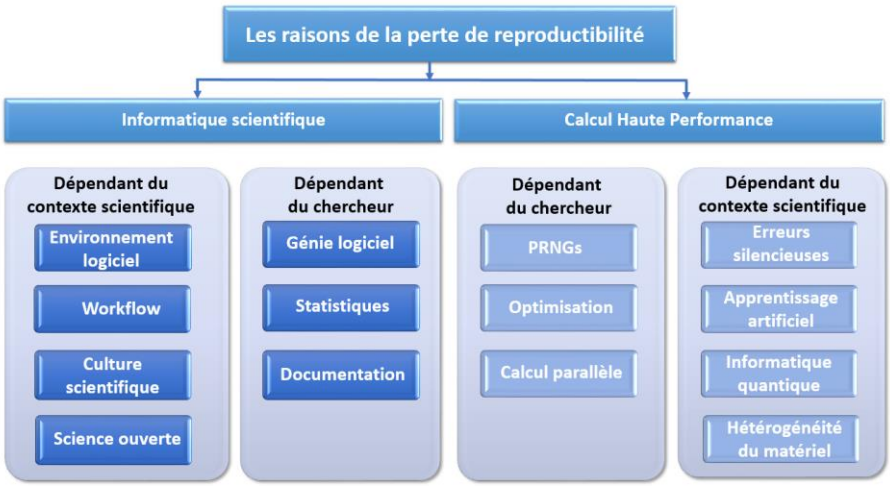

Figure 1. *Les différentes raisons de pertes de la reproductibilité pour les sciences ayant recours à l'informatique*

4.1. La science ouverte

Une des raisons principales sous-jacente à l'incapacité de reproduire les articles scientifiques est la réticence ou l'échec des auteurs à partager leurs artefacts (codes, scripts, données, etc.). Sans partage de codes et de données, la reproductibilité d'un travail scientifique utilisant un ordinateur pour produire une partie de ses résultats est impossible (soit une grande partie de la recherche, que ce soit en informatique, mais également en biologie, et physique, et autres). L'essor de l'informatique parmi tous les domaines scientifiques a contribué à la crise de la reproductibilité, comme

l'a bien décrit Ben Marwick (Marwick 2015). La plupart des auteurs qui plaident en faveur de la recherche reproductible soulignent l'importance du partage des codes et des données. En effet, de nombreuses études citées précédemment ont indiqué que les problèmes de reproductibilité proviennent principalement de l'absence de partage de ces éléments.

L'utilisation de logiciels propriétaires peut également entraîner des problèmes de reproductibilité. Dans (Nüst *et al.* 2020), les auteurs rappellent que nous avons besoin de « l'open source » pour pouvoir avoir une science reproductible. L'Académie Nationale des Sciences, de l'Ingénierie et de la Médecine (National Academies of Sciences 2019) déclare que les agences de financement devraient financer la science ouverte. Un autre aspect a son importance : la propriété intellectuelle du code publié. Sans licence explicite, les droits de propriété intellectuelle par défaut s'appliquent, comme décrit dans (Halchenko et Hanke 2015), ce qui peut freiner la possibilité de reproduire certains articles.

Enfin, d'autres raisons sont mentionnées de manière informelle et personnelle par les auteurs eux-mêmes dans le sondage mené par Collberg et Proebsting (2016). Par exemple, certains auteurs hésitent à publier leur code car ils estiment qu'il devrait être nettoyé ou modifié. Ils considèrent que leur code n'est pas assez propre pour être partagé, ce qui est regrettable. Parfois, le code n'était pas initialement destiné à être partagé, et le rendre facilement accessible nécessiterait trop d'effort de la part de l'auteur qui n'est pas toujours à jour avec les meilleures pratiques en génie logiciel. Les problèmes de licence peuvent également empêcher les auteurs de publier leur code, en particulier lorsqu'il s'agit de logiciels non open source. De plus, une différence dans l'accessibilité du code a été observée entre le domaine public et le domaine privé. Fait intéressant, certains auteurs ne peuvent plus utiliser leur propre code parce que la seule

personne qui savait l'utiliser a quitté le laboratoire ! Cela souligne l'importance de mettre en œuvre de bonnes pratiques de génie logiciel pour éviter de tels écueils. De surcroît, plus conventionnellement, le code peut être perdu. Cela peut être dû à des plantages ou à de mauvaises politiques de sauvegarde, parfois le langage de programmation ou le style peuvent être trop obscurs ou dépassés pour être utilisés par quelqu'un d'autre. La non-divulgation peut aussi être intentionnelle pour atténuer les risques tels que la découverte d'une vulnérabilité de sécurité, et d'empêcher son exploitation par quiconque. Les grands modèles de langages peuvent scruter les codes et aider des « hackers » mal intentionnés.

Le principal moyen pour accroître le mouvement de la science ouverte est d'en parler. De nombreux articles sur la recherche reproductible mettent en effet en lumière l'importance de la science ouverte. Cependant, il existe des exemples concrets d'actions entreprises pour renforcer l'ouverture de la science. L'un des premiers projets de science ouverte était peut-être le Stanford Exploration Project de Jon Claerbout. Plus récemment, le CERN a développé Zenodo, dans le cadre de la science ouverte. L'Inria propose Software Heritage pour archiver l'ensemble des versions de codes sources afin d'archiver la connaissance (Di Cosmo, Zacchiroli 2017). Les journaux réagissent également, comme le montre la politique de PLOS (PLOS one n.d.). Les États, qui ont un pouvoir de financement, doivent également s'impliquer dans la science ouverte, comme l'Institut National de la Santé (Collins et Tabak 2014) ou le gouvernement français (Ministère de l'enseignement supérieur et de la recherche 2021) qui soutient fortement l'initiative l'archive ouverte HAL. Enfin, l'intégration de l'enseignement de la science ouverte et de la reproductibilité dans le cadre éducatif des jeunes chercheurs pourrait permettre de faire avancer cette cause (Janz 2016), c'est la voie qu'ont

choisi Konrad Hinsen, Arnaud Legrand, et Christophe Pouzat avec leur proposition de MOOC sur la recherche reproductible (Pouzat *et al.* 2018) qui se décline maintenant en deux niveaux.

4.2. Documentation

La documentation, dans le contexte de la recherche scientifique, fait référence à la sauvegarde et à la fourniture d'informations détaillées sur le code, les logiciels, les méthodes et les procédures expérimentales utilisées dans une étude. Dans le contexte de la recherche reproductible en informatique, l'importance de la documentation lors du partage du code est primordiale. Une documentation précise et complète est essentielle pour faciliter la réplication des résultats de recherche et promouvoir la transparence et la crédibilité au sein de la communauté scientifique. Comme le souligne l'étude (Boettiger 2015), une documentation incomplète ou imprécise sur la manière d'installer et d'exécuter le code peut être un obstacle significatif à la reproduction, en particulier pour les chercheurs qui ne sont peut-être pas familiers avec les outils spécifiques et les gestionnaires de paquets impliqués. Ce problème est encore souligné dans (Kitzes *et al.* 2018), où les auteurs discutent de la disponibilité des données et des logiciels, y compris l'importance d'une documentation adéquate, des pratiques open-source, des techniques d'ingénierie logicielle et des considérations sur les droits d'auteur.

Les meilleures pratiques, telles que définies dans (Wilson *et al.* 2014), soulignent la nécessité de documenter la conception et l'objectif du code plutôt que de se concentrer uniquement sur ses mécanismes. En réalisant cela, les chercheurs permettent à d'autres de comprendre la fonctionnalité du code et ses objectifs, facilitant ainsi la reproductibilité. L'étude dans (Boettiger 2015) souligne également l'impact d'une documentation

imprécise sur la reproduction des analyses, un nombre significatif d'expériences utilisant des logiciels populaires n'étant pas reproductibles en raison d'une documentation incomplète.

De plus, dans le domaine en évolution rapide de l'intelligence artificielle (IA), les pratiques de documentation sont cruciales pour garantir la reproductibilité, comme le souligne Gundersen et Kjensmo (2018). De nombreuses études de recherche en IA manquent de méthodes et de codes bien documentés, rendant difficile la reproduction des résultats rapportés et freinant les progrès dans le domaine. Cependant, l'article reconnaît également que les pratiques de documentation se sont améliorées au fil du temps, soulignant l'importance de poursuivre les efforts pour promouvoir et maintenir des normes de documentation rigoureuses.

Une documentation efficace des logiciels et du code est un aspect fondamental de la recherche reproductible en informatique. Les chercheurs doivent s'efforcer de fournir une documentation claire et complète, incluant les procédures d'installation, les descriptions des paramètres et la justification de la conception. Une documentation adéquate permet non seulement de reproduire les résultats de recherche, mais favorise également la collaboration, le partage des connaissances et l'avancement de la science.

4.3. Statistiques

En 2005, l'article de Ioannidis précédemment cité (Ioannidis 2005) a fortement impacté le monde de la recherche reproductible. Il affirmait que ses simulations démontrent que les affirmations des publications scientifiques sont plus susceptibles d'être fausses que vraies, soulevant des questions sur la qualité des résultats de recherche et leurs implications pour le développement scientifique. L'importance des statistiques en recherche

est en effet majeure. La statistique joue un rôle crucial dans la détection et l'évitement de résultats faux-positifs, qui sont une préoccupation significative dans la recherche scientifique et dans le domaine médical en particulier. Plusieurs problèmes en statistique peuvent conduire à une perte de reproductibilité dans les articles, ce qui est plus, dans ce cas, une question de réplicabilité (la capacité à obtenir la même conclusion scientifique en réalisant une autre expérience).

Le « P-hacking », défini comme la manipulation des analyses statistiques pour obtenir des résultats statistiquement significatifs, est l'un des principaux facteurs contribuant à la crise de la reproductibilité. Les pratiques de P-hacking, telles que le choix sélectif des données ou la réalisation de multiples tests statistiques jusqu'à l'obtention d'un résultat significatif, peuvent conduire à des conclusions biaisées et peu fiables. Comme décrit dans (Forstmeier *et al.* 2017), ce problème est exacerbé par la pression lié à la culture scientifique de ne publier que des résultats statistiquement significatifs, ce qui peut donc créer un biais dans la recherche scientifique, puisque toutes les études trouvant un résultat négatif ne seront pas publiées, celles trouvant un résultat positif apparaitront comme correcte sans nuance. Ils présentent le fait que la diminution de la taille de l'échantillon (parfois pour des raisons éthiques d de préservation des animaux de laboratoires), la pression impliquant la recherche d'innovation et la réalisation de tests multiples peuvent tous augmenter la probabilité de conclusions positives erronées. Pour aborder ces problèmes, il est essentiel d'adopter des pratiques rigoureuses en recherche. Positionner les observateurs en aveugle pendant la collecte et l'analyse des données, et rapporter tous les résultats, qu'ils soient significatifs ou non, sont quelques-unes des stratégies qui peuvent améliorer l'objectivité de la recherche scientifique. De plus, réallouer les

efforts de la recherche de nouveauté et de découverte vers la reproduction de résultats de recherche importants peut bénéficier à la communauté scientifique. Il est crucial d'évaluer la recherche sur la base de la rigueur scientifique plutôt que de se fier uniquement aux métriques liées aux facteurs d'impact des journaux. La dépendance aux P-values comme mesure de preuve et de véracité est un autre aspect critique à considérer dans le contexte de la recherche reproductible. Les P-values sont souvent utilisées pour déterminer la force des preuves dans les résultats de recherche, mais des études ont remis en question leur fiabilité et leur objectivité (Nuzzo 2014). Les chercheurs ont noté que même de petits changements dans la significativité statistique peuvent entraîner des changements importants dans l'interprétation des résultats (Gelman et Stern 2006). Cette divergence peut conduire à des conclusions trompeuses et souligne la nécessité d'une réévaluation des méthodes statistiques (Nuzzo 2014).

Formuler des hypothèses après les résultats est nommé « HARKing », c'est une autre pratique courante dans la communication scientifique qui peut compromettre la reproductibilité (Kerr 1998). Le HARKing implique de présenter des hypothèses après-coup comme si elles étaient des hypothèses a priori dans les rapports de recherche. Cette pratique peut conduire à des interprétations biaisées et inexactes des données, car elle permet aux chercheurs de choisir sélectivement des hypothèses qui s'alignent avec leurs résultats. Bien que les motivations derrière le HARKing soient variées, il est largement considéré comme inapproprié et a des implications négatives pour l'intégrité scientifique.

Il est important de faire attention à ne pas surestimer l'importance de résultats qui semblent significatifs du point de vue statistique. Gelman et Stern (2006) ont expliqué que des variations importantes dans les niveaux

de significativité statistique pourraient en réalité représenter des changements mineurs et non significatifs dans les données réelles. Cette tendance à sur-interpréter les résultats est une erreur différente d'autres problèmes communs en statistique, comme confondre la significativité statistique avec l'importance pratique, diviser les résultats en catégories « significatif » ou « non significatif », et choisir des seuils de significativité de façon arbitraire. Cette erreur est fréquente et il est donc crucial que les étudiants et les professionnels en soient conscients pour éviter de mal interpréter les données.

Les erreurs de type I et de type II sont des concepts dans les tests d'hypothèses liés à un rejet incorrect ou à un échec à rejeter une hypothèse nulle. Une erreur de type I, également connue sous le nom de faux positif, se produit lorsque l'hypothèse nulle est vraie, mais est rejetée à tort. Par exemple, conclure qu'un médicament est efficace alors qu'il ne l'est pas. Une erreur de type II, ou faux négatif, se produit lorsque l'hypothèse nulle est fausse mais échoue à être rejetée par erreur, comme conclure qu'un médicament est inefficace alors qu'il fonctionne réellement. La puissance statistique est la probabilité de rejeter correctement une fausse hypothèse nulle, ce qui signifie qu'elle mesure la capacité d'un test à détecter un effet lorsqu'il y en a un. La significativité statistique, d'autre part, indique la probabilité que la différence ou l'association observée dans les données ne soit pas due au hasard, souvent évaluée par une « valeur p ». Une puissance statistique élevée et un résultat significatif statistiquement (typiquement une valeur p inférieure à 0,05) contribuent à la fiabilité des résultats des tests. Ce seuil se révèle être souvent bien insuffisant et J.P.A. Ioannidis a proposé de le ramener à 0,005 pour réduire le taux potentiel de faux positifs des résultats obtenus (Ioannidis 2005), l'article initial très cité a été

récemment corrigé en 2022 (erreur mineure, une paire de parenthèse manquante dans la table 2) (Ioannidis 2022).

La puissance statistique est une question significative dans des domaines comme la psychologie. Dans le cadre hybride des tests de significativité de l'hypothèse nulle (NHST) prévalent dans la science psychologique, la puissance statistique est définie comme la probabilité de rejeter correctement une fausse hypothèse nulle, évitant ainsi une erreur de type II. À mesure que la puissance statistique augmente, la probabilité d'une erreur de type II diminue. Cependant, la puissance statistique a tendance à être sous-estimée par rapport au niveau de significativité. Typiquement, les chercheurs fixent un seuil de 5% pour les erreurs de type I mais acceptent souvent un taux plus élevé de 20% pour les erreurs de type II, indiquant une plus grande préoccupation pour éviter les faux positifs que les faux négatifs. Malgré le seuil conventionnel de 20% pour les erreurs de type II, l'attention réelle à la puissance statistique est souvent insuffisante, conduisant à un taux d'erreur de type II effectif qui pourrait être aussi élevé que 80%. Ce problème est évident dans divers domaines, soulignant une lacune critique dans les pratiques statistiques, comme décrit dans (Button et Munafò 2017), et cela peut conduire à des problèmes de reproductibilité (Clayson *et al.* 2019). Une solution serait d'augmenter la taille de l'échantillon et d'accroître la précision de la variable pour diminuer l'erreur de mesure, donc nécessiterait plus de calculs (Button et Munafò 2017). Dans (Strong et Alvarez 2019), les auteurs abordent la crise de la réplication dans la recherche psychologique en proposant l'utilisation du « bootstrap resampling » combiné avec des données pilotes ou synthétiques pour améliorer la puissance statistique et la reproductibilité des expériences. Cette méthode permet une analyse personnalisée des conceptions expérimentales, en tenant compte de facteurs tels que la taille

de l'échantillon, le nombre d'essais et les critères d'exclusion, souvent négligés dans les outils traditionnels d'analyse de puissance. L'article explique comment le « bootstrap resampling » aide à estimer à la fois la variabilité de l'échantillonnage et de la mesure en simulant des expériences plusieurs fois, ce qui améliore la précision des prédictions de puissance. En utilisant une boîte à outils MATLAB fournie par les auteurs, les chercheurs peuvent appliquer ces méthodes de simulation à leurs données pilotes pour concevoir des expériences qui sont plus susceptibles de produire des résultats reproductibles.

Certains outils sont conçus pour aider avec les statistiques, comme R Markdown, un outil conçu pour simplifier l'analyse statistique reproductible, le rendant adapté à la fois pour la recherche avancée et les cours d'introduction à la statistique (Baumer *et al.* 2014), ou bien Pernet *et al.* (2013) qui ont introduit une boîte à outils Matlab open-source conçue pour des analyses de corrélation robustes. La méthode de corrélation de Pearson, couramment utilisée en psychologie, a une portée limitée car elle ne détecte que les relations linéaires et est très influencée par les valeurs extrêmes, qui peuvent donner une image faussée des données. Pour pallier ces limites, une boîte à outils Matlab en accès libre propose deux alternatives : « *percentage-bend correlation and skipped-correlations* ». La première méthode réduit l'impact des valeurs extrêmes en leur attribuant moins de poids, tandis que la seconde les élimine directement de l'analyse. Ces approches offrent une meilleure approximation des véritables relations entre les données, en maintenant un équilibre entre la minimisation des erreurs positives et le maintien de la capacité à détecter des associations réelles.

4.4. Culture scientifique

Le paysage actuel de la recherche scientifique est affecté par plusieurs problèmes qui entravent la reproductibilité, essentielle pour l'avancement de la science. Un problème majeur est le manque d'incitations de la part des journaux et des conférences à fournir tous les artéfacts liés à un article de recherche, tels que les ensembles de données, le code ou des méthodes détaillées, qui sont essentiels pour une transparence et une reproductibilité complètes. Nosek a écrit : *« En raison de fortes incitations à l'innovation et de faibles incitations à la confirmation, la reproduction directe est rarement pratiquée ou publié*e », et « *Les découvertes innovantes rapportent des récompenses de publication, d'emploi et de titularisation ; les découvertes répliquées produisent un haussement d'épaules.* » (Collaboration 2012), citation présente dans (Collberg & Proebsting, 2016). Cela crée une culture qui décourage les efforts de reproduction, amenant souvent les chercheurs en début de carrière à se détourner de telles entreprises au profit d'activités qui renforcent leurs profils académiques. Ce problème est aggravé par la mentalité enracinée du « *publish or perish* » qui met la pression sur les chercheurs pour produire continuellement de nouvelles découvertes, parfois au détriment d'une recherche rigoureuse et de qualité. Le processus de révision par les pairs n'est pas non plus sans failles. Souvent, il ne privilégie pas la reproductibilité des études, ce qui peut conduire à la publication de résultats qui ne peuvent pas être vérifiés indépendamment. Enfin, le biais de publication en faveur des résultats positifs entraîne souvent une sous-représentation des résultats négatifs ou nuls, faussant davantage le paysage de la recherche. Collectivement, ces problèmes posent des défis importants à l'intégrité et à la fiabilité de la recherche scientifique, appelant à des changements systémiques dans la manière dont nous menons et communiquons la science. Bajpai *et al.* (2017) discutent de trois éléments

qui posent des problèmes pour la reproductibilité : le manque d'incitations des journaux (qui privilégient uniquement les articles innovants), le processus de révision en double aveugle qui oblige les auteurs à cacher des informations ou des données potentiellement cruciales, et les relecteurs (reviewers) qui ne testent pas la reproductibilité. Le papier de Bajpai *et al.* propose d'inclure une section sur la reproductibilité dans les articles pour encourager les auteurs, promouvoir les articles reproductibles et améliorer le processus de relecture. Baker (2016) présente 14 facteurs qui peuvent contribuer à la perte de reproductibilité, principalement liés à des problèmes statistiques, à la pression de publication et au code non disponible, entre autres. Il propose 11 solutions qui abordent directement les problèmes soulevés, comme l'amélioration de la formation statistique des chercheurs. Munafò *et al.* (2017) identifient les biais cognitifs, les améliorations des méthodes, une collaboration accrue entre les chercheurs et une relecture par les pairs améliorée comme problèmes et solutions pour améliorer la reproductibilité. Ten Hagen (2016) soutient que les journaux privilégient excessivement la nouveauté, ce qui mine la recherche reproductible, car cela décourage les chercheurs de tenter de répliquer des résultats précédemment publiés. Fanelli (2010) montre que la pression des financements et de réussir à sortir une publication pousse les chercheurs à publier uniquement des résultats positifs, limitant la publication de résultats négatifs qui pourraient également contribuer aux progrès scientifiques en révélant ce qui ne fonctionne pas ou n'est pas vrai, y compris les résultats négatifs de reproduction d'articles précédemment publiés.

4.5. Environnement logiciel

L'environnement logiciel est un aspect crucial de la recherche reproductible, car l'utilisation des ordinateurs est devenue omniprésente dans le monde scientifique. Un grand nombre de publications utilisent désormais du code, des scripts ou des données en entrée pour générer certains résultats.

Même si vous partagez votre code et vos données, il est très peu probable que la personne tentant de reproduire les résultats ait exactement le même environnement logiciel que vous. Par conséquent, elle pourrait ne pas être capable d'exécuter correctement le code en raison d'incompatibilités potentielles, de différentes bibliothèques, ou de logiciels non disponibles (Hinsen 2013). Pour résoudre ce problème, Hinsen suggère d'utiliser des bibliothèques fiables et éprouvées, d'écrire du code clairement (tout en tenant compte de la performance, en particulier dans le contexte du calcul à hautes performances), de documenter les formats utilisés, d'évaluer les dépendances et de fournir des exemples prêts à l'emploi pour faciliter la prise en main. En 2015, Hinsen continue de fournir des informations précieuses, conseillant la prudence concernant l'ajout de données d'entrée non mentionnées dans les workflows, et les éventuels bugs logiciels ou matériels qui en découlent (Hinsen 2014). Desquilbet *et al.* (2019) discutent également de ces différents défis dans leur livre. La couche de l'environnement logiciel englobe différents niveaux. Tout d'abord, nous avons le système d'exploitation. Les calculs peuvent différer à travers l'utilisation de différents systèmes d'exploitation, car l'utilisation du matériel peut différer. L'utilisation de systèmes basés sur Linux (open source) est obligatoire pour la recherche reproductible, car nous avons besoin de logiciels open source (contrairement à Windows ou MacOS qui ne sont pas open source, donc rendent plus difficile

l'obtention de la même configuration pour assurer la reproductibilité). Linux est fortement conseillé car il est largement utilisé, entièrement open source et offre de nombreux outils pour mener à bien une recherche reproductible. Deuxièmement, comme mentionné, les codes utilisent presque toujours des bibliothèques. Ainsi, lorsqu'ils sont partagés, si la machine de destination n'a pas accès aux mêmes bibliothèques, avec les mêmes versions, le code ne fonctionnera pas. Chaque bibliothèque peut également avoir des dépendances avec d'autres. Cela est connu sous le nom de « Dependency Hell » (Boettiger 2015). Des outils de haut niveau, comme des cadriciels (frameworks par la suite) ou des bibliothèques, peuvent également agir comme des boîtes noires, qui ne permettent pas à un autre chercheur d'utiliser le code pour sa propre recherche. C'est ce que Hinsen a appelé "code réutilisable VS code rééditable" (Hinsen 2018).

En calcul à hautes performances, les compilateurs sont des logiciels importants. Le calcul intensif nécessite d'éviter le gaspillage de d'énergie et de temps de calcul, or les langages compilés sont connus pour être plus efficaces que les langages interprétés. Ainsi, les codes exécutés sur des clusters de calcul ou des superordinateurs sont principalement produits par des compilateurs C, C++ ou Fortran. Malheureusement, de nombreux scientifiques qui ne sont pas spécialisés en informatique, des biologistes ou des chimistes, peuvent par exemple trouver plus facile de travailler avec Python ou R, qui sont des langages utiles avec de nombreuses bibliothèques que ce soit pour les aspects statistiques ou encore pour développer des modèles d'apprentissage automatique. Différentes versions des compilateurs, ou du langage de programmation, peuvent également entraîner une perte de reproductibilité. L'importance de l'ensemble de la pile logicielle a déjà été prêchée par Claerbout selon la

citation de Donoho que nous avons mentionnée plus tôt (Buckheit et Donoho 1995).

4.6. Workflow

Le problème du workflow (littéralement flux de travail ou processus métier) dans la recherche reproductible tourne autour de la nécessité de capturer et de communiquer l'ensemble du processus d'analyse des données, d'expérimentation et d'exécution du code de manière claire et organisée, dans un ordre précis. En effet, la plupart du temps, un article n'utilise pas un seul code ou script, mais une succession de codes s'appliquant à différentes données. Un workflow reproductible devrait permettre à d'autres chercheurs de vérifier et de reproduire de manière indépendante les résultats présentés dans une étude. Cependant, en pratique, les workflows manquent souvent de transparence, rendant difficile pour les autres de comprendre les étapes suivies, les paramètres utilisés et les transformations de données appliquées. Ce manque de clarté peut venir d'une documentation incomplète ou ambiguë, rendant difficile la reproduction de la séquence exacte d'opérations qui a conduit aux résultats rapportés. De plus, les workflows peuvent impliquer plusieurs outils, bibliothèques et dépendances logicielles, et la gestion des interactions entre ces composants ajoute de la complexité au processus de reproductibilité. Le problème est aggravé lorsque les workflows sont répartis sur différents scripts, notebook ou même différents langages de programmation, il devient alors plus difficile de s'assurer que chaque détail est capturé de manière précise et cohérente. De plus, le contrôle de version devient crucial dans la gestion des changements apportés au workflow au fil du temps. Sans un suivi de version approprié, il peut être difficile de retracer l'état exact du workflow lorsqu'un résultat particulier a été obtenu.

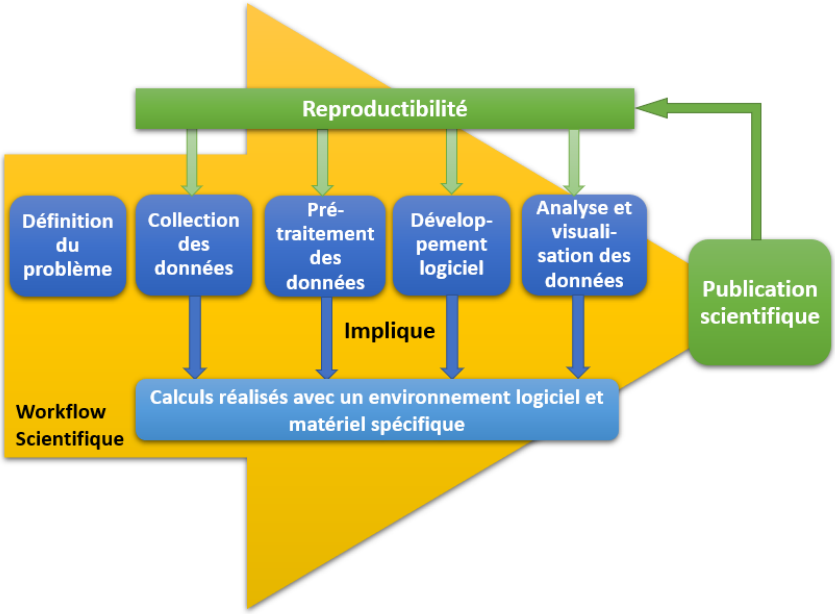

Figure 2. *Exemple de workflow scientifique, amenant à une publication.*

Aborder le problème du workflow en recherche reproductible nécessite d'adopter les meilleures pratiques pour documenter les étapes, fournir des explications claires sur la raison des prises de décisions, et assurer la disponibilité de toutes les données et codes nécessaires. Sarah Cohen-Boulakia et ses collègues ont réalisé une étude assez exhaustive sur les workflows, axée sur les sciences de la vie (Cohen-Boulakia *et al.* 2017). Stanisic *et al.* (2015) ont également travaillé sur ce thème. Nous discuterons des outils utilisables par la suite. La figure 2 présente une représentation visuelle d'un concept de workflow d'une publication scientifique. Cette illustration met en évidence un workflow standard comprenant des étapes telles que la collecte de données, la préparation des données, l'élaboration de solutions logicielles et la conduite d'analyses de données, aboutissant à la création d'un article scientifique. Pour reproduire une telle étude, il est

nécessaire qu'un scientifique soit capable de rejouer cette séquence entière, appelée le workflow. Le résultat précis de chaque étape dépend d'un environnement logiciel particulier, ce qui est une contrainte supplémentaire.

4.7. Génie logiciel

Nous avons mentionné que l'ingénierie logicielle est sujet important pour la recherche reproductible et qu'il peut être souvent négligé. En effet, avec l'omniprésence de l'informatique, de nombreux domaines scientifiques utilisent la programmation pour mener leurs recherches. Désormais, vous utilisez du code pour préparer des données, les transformer, les visualiser, réaliser vos statistiques, concevoir vos expériences, etc. Pour le calcul à hautes performances, vous aurez principalement besoin de compétences de bas niveau (réalisation de scripts au niveau du système d'exploitation, connaissance des systèmes d'exploitation et de langages informatiques efficaces comme C, C++ ou Fortran). Pour des objectifs plus généraux et l'analyse de données, les scientifiques utilisent principalement Matlab, Python ou R, ainsi que certains frameworks, bibliothèques et autres outils de haut niveau. Toutes ces compétences peuvent être difficiles à acquérir pour certains d'entre eux, notamment pour ceux qui n'ont pas suivi un cursus classique en informatique (qu'ils soient biologistes, physiciens, sociologues ou chimistes…). Alors que les technologies de haut niveau offrent un moyen plus facile de gérer le calcul, elles sont également moins stables et plus obscures que les technologies de bas niveau, car elles cachent la complexité dans des boîtes noires, rendant la reproductibilité plus difficile. Ce sont les « défauts » des technologies modernes de haut niveau.

Lors de la publication, fournir des artefacts est une bonne chose, mais la qualité logicielle du code produit est essentielle pour avoir un code maintenable, lisible et évolutif. De plus, pour aider à garantir la reproductibilité, plusieurs outils ont été développés. Cependant, ils nécessitent souvent certaines compétences en informatique. Les non-informaticiens utilisent des ordinateurs comme un outil, sans avoir la connaissance suffisante de ce que fait réellement l'ordinateur (et peuvent supposer par exemple le déterminisme absolu de leurs machines). Dans (Hinsen 2018), Konrad Hinsen présente un exemple de pourquoi il est important de savoir ce qui se passe en coulisse. Dans un exemple, il charge un jeu de données sur la santé avec le logiciel Pandas à partir d'un fichier csv (il a pris cinq ans de données). Pandas essaie automatiquement de trouver les types de données appropriés pour les données, Hinsen a remarqué un biais. Alors que le type utilisé pour ses données était « int64 », lorsqu'il a chargé l'ensemble du jeu de données (33 ans contre 5 ans), les données ont été reconnues comme étant désormais du type « objet » (et non plus « int64 »). Dans la documentation de Pandas, il est mentionné une « conversion intelligente des données tabulaires », mais les règles utilisées ne sont pas explicites. Bien sûr, vous pouvez gérer cela avec un peu d'effort, mais cela montre qu'il y a des complexités cachées auxquelles un non-expert pourrait ne pas penser. L'article souligne également le fait qu'un code réutilisable et rééditable ne sont pas la même chose. Si vous fournissez du code avec votre article, et dans votre analyse, vous utilisez une fonction complexe d'une bibliothèque de haut niveau, d'autres chercheurs pourront-ils éditer cette fonction pour leur propre usage ? Les fonctions de haut niveau peuvent agir comme une boîte noire, et toute mise à jour de la bibliothèque peut vous faire perdre la reproductibilité au bit près. Un code rééditable sera plus compréhensible pour la majorité.

Comme discuté précédemment, il existe des cas où un groupe de chercheur pourrait cesser d'utiliser leur propre logiciel car la seule personne compétente connaissant son fonctionnement ne fait plus partie de l'équipe. Cela souligne la nécessité d'adhérer à des principes robustes d'ingénierie logicielle pour prévenir ce type de scénarios. De plus, dans une situation plus typique, un logiciel peut devenir inaccessible en raison d'une défaillance matérielle rendant le code inutilisable (suppression par exemple), ou lorsque le langage de programmation utilisé pour développer le programme pourrait devenir si rare ou si archaïque qu'il est peu pratique pour d'autres chercheurs de l'adopter.

Chapitre 5

Les problèmes de reproductibilité propre au calcul haute performance

5.1. Exécution de programmes en parallèle

Le calcul parallèle implique l'exécution simultanée de multiples tâches pour un traitement plus rapide de la charge de travail. Avec le calcul parallèle viennent de nouveaux défis, tels que les changements dans l'ordre d'exécution d'opérations en virgule flottante, qui peut introduire des résultats numériques non reproductibles (Goldberg 1991). La combinaison du comportement non déterministe dans les programmes parallèles et de la non-associativité des opérations en virgule flottante pose des défis significatifs en termes de reproductibilité, comme le montre l'exemple de la Figure 3.

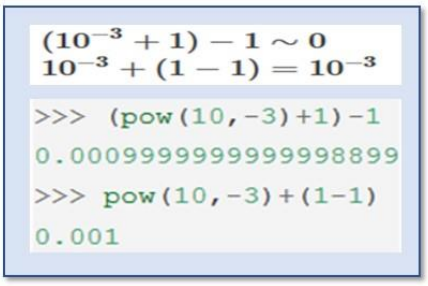

Figure 3. *Exemple de la non-associativité des opérations à virgule flottante.*

Dans (Hunold 2015), cet état de l'art nous apprend qu'une majorité de chercheurs en informatique utilisant le calcul à hautes performances sont préoccupés par la reproductibilité de leurs articles. Une grande majorité, 94%, croient que la reproductibilité des articles devrait être améliorée dans

le domaine du calcul parallèle. Dans les mêmes proportions, ils pensent que les articles de recherche actuels dans le domaine du calcul parallèle sont très peu reproductibles. En calcul parallèle, l'ordre d'exécution des tâches peut varier en raison de facteurs tels que l'ordonnancement des tâches, l'équilibrage de charge et le multithreading. Dans (Chohra *et al.* 2016), les auteurs supposent que le non-déterminisme peut provenir de l'ordonnancement dynamique des données, de la disponibilité des ressources physiques et de différents ensembles d'instructions du processeur. Ce comportement non déterministe peut conduire à différents résultats lorsque le même programme est exécuté plusieurs fois dans les mêmes conditions. Cela pose des défis pour la reproductibilité, car les résultats peuvent varier de manière imprévisible et cela rend le débogage et la validation des résultats particulièrement difficiles. De plus, la non-associativité de l'arithmétique en virgule flottante exacerbe le problème. Les opérations en virgule flottante telles que la multiplication et l'addition ne sont pas associatives dans l'espace des fractions Q où sont situés les nombres réels en virgule flottante. Cela signifie que changer l'ordre des opérations peut donner des résultats différents. Lorsque ces opérations non associatives sont combinées avec le calcul parallèle, atteindre des résultats identiques au niveau binaire, ce qui peut être un aspect critique du débogage, devient un défi encore plus significatif dans le calcul Exascale (Demmel et Nguyen 2013b). Les systèmes Exascale, se référant aux superordinateurs capables d'effectuer 10^{18} opérations en virgule flottante par seconde, présentent une puissance de calcul immense mais posent également des défis significatifs pour la reproductibilité. Parmi les deux opérations en virgule flottante non associatives, l'addition est plus sensible. Cela implique la production d'algorithmes de compensation, comme des sommes compensées par exemple, qui sont particulièrement

utile pour la phase de réduction. Il s'agit d'une étape cruciale dans les algorithmes parallèles où les résultats partiels calculés par différentes unités de traitement en parallèle (par exemple, des threads, des processeurs, des nœuds de calcul) sont combinés ou agrégés pour produire un résultat global. Pour les opérations flottantes, l'ordre et la façon dont on organise les calculs pour compenser les erreurs d'arrondis sont importants.

Nous présenterons des solutions pour les problèmes de reproductibilité de ces calculs parallèles dans la section 7. Avec le nouveau superordinateur Frontier (Top500) et de nombreux autres superordinateurs proposés depuis plus d'une décennie, nous avons dépassé le million de cœurs capables de réaliser des calculs en parallèle.

L'utilisation de grilles de calcul (« Grid Computing ») ou de bibliothèques parallèles telles que Message Passing Interface (MPI), avec le passage de messages asynchrones sur de grandes simulations, peut fréquemment conduire à une exécution où l'on change l'ordre des opérations dans les calculs sur des nombres en arithmétique flottante. Dans ce cas, nous avons une perte de répétabilité, ce qui augmentera considérablement la difficulté de débogage.

5.2. Simulations de Monte-Carlo et nombres pseudo-aléatoires

Lors de notre recherche bibliographique de la recherche reproductible pour le calcul à hautes performances, nous avons souvent vu l'affirmation selon laquelle les simulations de Monte Carlo ne sont pas déterministes. Bien qu'il soit vrai que les simulations de Monte Carlo peuvent être non déterministes pour les mêmes raisons que d'autres programmes, telles que l'arithmétique à virgule flottante désordonnée, le parallélisme, etc., nous soutenons que qualifier les simulations de Monte Carlo de non

déterministes en raison de l'utilisation d'une source pseudo-aléatoire peut être trompeur. En fait, lors de la production de résultats scientifiques, la méthode de Monte Carlo utilise précisément des modèles déterministes de hasard appelés « Générateurs de Nombres Pseudo Aléatoires » (PRNG). C'est l'approche scientifique utilisée pour simuler le hasard en le maîtrisant et en fournissant de très bonnes propriétés statistiques. C'est un aspect important dans la conception des générateurs de nombres pseudo-aléatoires, car il est alors possible d'obtenir la répétabilité pour le débogage de chaque expérience « indépendante ». Lors de l'exécution de modèles stochastiques, tels que les simulations de Monte Carlo, les scientifiques expérimentés font un usage intelligent des états des PRNGs. Dans le cas des anciens générateurs il s'agissait de simple graines (seed), afin d'assurer la répétabilité des résultats. La génération de nombres pseudo ou quasi aléatoires est totalement déterministe, ce qui est un atout fort pour les études scientifiques. Cependant cela suppose un usage correct des PRNGs pour obtenir un calcul stochastique reproductible, particulièrement lorsqu'il s'agit de Monte Carlo parallèle. Hellekalek a justement averti les chercheurs en simulation lors de la conférence sur l'informatique parallèle et distribuée de 1998 : « *Ne faites pas confiance au Monte Carlo parallèle* » (Hellekalek 1998). Nous avons également trop souvent vu le mauvais conseil d'initialiser votre PNRG avec « time(NULL) » pour permettre un « vrai aléatoire » y compris dans un ouvrage sur le calcul parallèle ! C'est un mauvais conseil qui ne devrait pas être appliqué pour pouvoir déboguer ou faire un travail statistiquement correct et reproductible. Cette approche utilisable pour des jeux, ou pour produire rapidement quelques jeux d'essais sur un programme débogué est encore trop souvent enseignée. Pour permettre la reproductibilité avec les PRNGs, vous devez maîtriser et sauvegarder méthodiquement les états initiaux et sélectionner une

technique de parallélisation avant d'exécuter simultanément des instances « indépendantes » de simulations de Monte Carlo (Hill *et al.* 2013). La figure 4 présente une méthode de parallélisation de flux aléatoires grâce à l'utilisation d'un espacement aléatoire, permettant l'exécution concurrente de calculs parallèles indépendants. Cette approche repose sur le fait que les flux ne se chevauchent pas afin de maintenir leurs opérations distinctes. La méthode d'espacement aléatoire est particulièrement efficace et simple pour les PRNGs ayant de très grandes périodes, où la probabilité de chevauchement des flux est extrêmement faible lors de la sélection de flux aléatoires sur l'ensemble de la période du PRNG. Dans les cas où le risque chevauchement entre les flux est statistiquement trop élevé, des méthodes alternatives comme le fractionnement de séquences sans recouvrement peuvent être utilisées. De plus, il est crucial pour les chercheurs dans la communauté scientifique de comprendre que le terme historique de graine (seed) est différent de l'état complet d'un PRNG moderne. L'état du PRNG dicte les valeurs qu'il produit. Inversement, utiliser une graine implique l'application d'une fonction spécifique pour transformer la graine en un état complet. Il est important de noter que ce processus de transformation peut varier selon les différentes plateformes et technologies, de sorte que la même graine peut conduire à différents flux de nombres aléatoires pour le même algorithme supposé de PRNG, dans différents langages de programmation ou frameworks, ceci impact en particulier les outils d'apprentissage sur lesquels bien des études d'intelligence artificielle se basent.

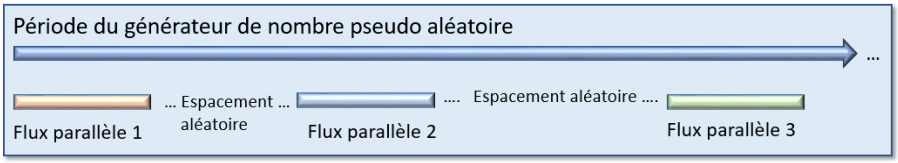

Figure 4. *Exemple de partition de flux stochastiques en utilisant la méthode d'espacement aléatoire*

Nous pouvons rencontrer des problèmes en traitant la parallélisation des PRNGs, cependant, nous disposons désormais de PRNGs de haute qualité qui peuvent être parallélisés correctement. Si vous utilisez les bonnes méthodes adaptées aux meilleurs PRNGs, comme décrit dans (Hill *et al.* 2013), il est possible de paralléliser les simulations de Monte Carlo et d'obtenir une répétabilité avec des résultats identiques au niveau binaire pour toutes les expériences stochastiques (Hill 2015). Voici une courte liste de certains PRNGs de bonne qualité utilisés pour le calcul parallèle, à partir de la sélection retenue dans (Antunes et Hill 2023) :

- Philox et Threefry, proposés par Salmon et ses collègues lors de la Conférence Supercomputing de 2011 (Salmon *et al.* 2011). Ils reposent sur des techniques cryptographiques comme AES (Rijmen et Daemen 2001). Ils proposent une technique de paramétrisation solide et facile pour résoudre le problème de distribution de flux stochastiques « indépendants » dans les applications parallèles. En raison de ses contraintes cryptographiques, on peut le considérer comme un générateur lent, mais statistiquement, ce type de générateur est très bon même si nous avons noté certains problèmes de reproductibilité (Hill 2015). Les PRNGs sécurisés d'un point de vue cryptographique sont également connus sous le nom de CS-PRNG (CS pour Crypto-Secure).

- MRG32k3a, est un générateur récursif combiné, il a été minutieusement sélectionné par L'Ecuyer par force brute pour répondre aux tests statistiques les plus rigoureux qu'il a lui-même développés (L'ecuyer 1999). Ce générateur facilite le fait d'avoir de nombreux flux parallèles, c'est un atout dans le calcul parallèle. Cependant, il peut être jusqu'à 19 fois plus lent que le Mersenne Twister dans sa mise en œuvre en C/C++, le rendant moins attrayant pour le calcul parallèle intensif. MRG32k3a est le choix par excellence pour la plus haute qualité statistique, tandis que Mersenne Twister offre de meilleures performances avec une solidité statistique légèrement moindre.
- WELL, développé par Panneton, L'Ecuyer, et Matsumoto (Panneton *et al.* 2006), améliore le Mersenne Twister d'un point de vue statistique. Provenant des registres à décalage à rétroaction linéaire (LFSR), WELL n'a pas connu une utilisation ou un développement des techniques de parallélisation aussi importante que Mersenne Twister.
- PCG, un PRNG récent, a été créé en 2014 par O'Neill (O'neill 2014). Il est plébiscité pour ses propriétés statistiques supérieures par rapport aux autres générateurs. La documentation de Numpy recommande son utilisation en raison de ces qualités, mais nous avons observé des failles en faisant un grand nombre de tests avec la bibliothèque de tests statistique TestU01 (discutée plus tard). D'autre part, la documentation de numpy mentionne que des faiblesses statistiques ont été repérées lors de l'utilisation de PCG64 dans un contexte de parallélisme massif. De fait ils ont introduit la version nommée PCG64DXSM que nous n'avons pas encore testée.

- Mersenne Twister (MT), introduit par Matsumoto et Nishimura (Matsumoto et Nishimura 1998), est rapidement devenu connu en raison de sa période gigantesque de 2^{19937} - 1. C'était le premier d'une famille de générateurs, certains conçus pour les GPU et les Field Programmable Gate Array (FPGA). La version SFMT utilise les possibilités vectorielles des processeurs modernes (Single Instruction Multiple Data), et est plus rapide que le MT original, il a de meilleures propriétés statistiques et propose une période encore plus grande allant jusqu'à 2^{216091} - 1, mais il est beaucoup moins connu que le MT original (Saito et Matsumoto 2006). Cependant, aucun des PRNGs de cette famille n'est adapté aux applications cryptographiques. Les failles connues ne posent pas de problèmes aux applications de simulation et son implémentation optimisée est particulièrement rapide contrairement à ce que l'on peut lire parfois.
- La dernière version de Xoshiro par (Blackman et Vigna 2021), basée sur un principe « XOR, shift, rotate ». Les auteurs proposent l'applications de « scramblers » non-linéaire sur les sorties du générateur Xoshiro (ainsi que sur Xoroshiro – Xor Rotate Shift Rotate de la même famille). Les auteurs affirment, que tout en maintenant la rapidité du générateur, ils ont fourni des améliorations pour éviter les échecs dans les tests statistiques liés à la linéarité tels que MatrixRank et LinearComp de TestU01 (sur lequel échoue le Mersenne Twister, un échec au test MatrixRank peut également survenir pour MT qui n'est pas un générateur de qualité cryptographique).

L'utilisation de certaines versions de ces algorithmes n'est pas toujours explicitement spécifiée par les langages, les bibliothèques ou les

frameworks (cadriciels). Par conséquent, cela peut entraîner des différences dans la qualité statistique et des différences dans le flux de nombres aléatoires générés, et ceci peut conduire à des difficultés pour reproduire le travail d'autres personnes, qui se basent simplement sur le nom des PRNGs utilisés. Par exemple, les langages R et Python, en considérant le module random de la bibliothèque standard, utilisent tous les deux le générateur Mersenne Twister de Matsumoto et Nishimura (générateur par défaut pour R et unique générateur pour Python). Par contre, R utilise la première version (1998) de MT qui renvoie un entier de 32 bits divisé par 2^{32} (et utilise donc seulement 32 des 53 bits de la mantisse d'un double) et qui connais des problèmes lors d'initialisation dans certains états. Il s'agit d'un ancien défaut connu sous le nom de « zero excess initialization state ». Il se manifeste lorsque l'état du générateur est initialisé avec un excès de zéros, affectant ainsi la qualité statistique des nombres générés. Ce problème a été corrigé par la suite. Python utilise la version de 2002 qui a corrigé ce défaut dans la version initiale de MT et dans ses variantes (Saito *et al*, 2008). Cette version combine deux tirages successifs d'entiers de 32 bits afin d'obtenir un double pour lequel tous les bits de la mantisse sont utilisés (ce dernier point n'est malheureusement pas documenté dans Python). Cet exemple illustre bien comment différentes versions d'un même algorithme peuvent entraîner des variations dans les flux de nombres aléatoires générés ainsi que dans la qualité statistique de ces derniers.

Une autre famille de nombres aléatoires existe : les vrais nombres aléatoires. Les vrais nombres aléatoires sont des nombres générés de manière imprévisible et non déterministe (contrairement aux nombres pseudo aléatoire évoqués précédemment). Un générateur de vrai nombres aléatoire (TRNG, pour True Random Number Generator) est un

dispositif ou une méthode qui génère de véritables nombres aléatoires. Contrairement aux PRNGs, qui utilisent des algorithmes déterministes et un état initial pour produire des séquences de nombres qui semblent aléatoires, les TRNGs s'appuient sur des processus physiques imprévisibles ou des sources de hasard pour générer de vrais nombres aléatoires. Certaines sources de hasard couramment utilisées dans les TRNGs peuvent être le bruit électronique, la désintégration radioactive, le bruit atmosphérique ou le bruit optique. Pour faire de la science reproductible avec des TRNGs, vous devez sauvegarder chaque nombre aléatoire que vous utilisez. Lorsqu'on traite des simulations de Monte Carlo lourdes, comme en physique des hautes énergies, utiliser des TRNGs n'est pas une option viable, car la sauvegarde de millier de milliards de nombres utiliserait trop de ressources matérielles. De plus, les vrais nombres aléatoires sont souvent lents à générer, comparés aux nombres pseudo aléatoires.

Une autre catégorie de générateurs de nombres aléatoires est connue sous le nom de générateurs de nombres quasi aléatoires (QRNG – Quasi Random Number Generator), qui sont conçus pour générer des séquences de nombres qui minimisent l'écart à l'uniformité de façon déterministe. Cette approche est intéressant pour le calcul intégral ou pour les problèmes de « space-filling » que l'on utilise notamment dans la réalisation de plans d'expériences très utiles en calcul à hautes performances.

5.3. Optimisation

Dans le monde du calcul haute performance, l'optimisation est une notion importante. Lorsque nous parlons d'optimiser l'exécution d'un processus sur un CPU, nous pouvons penser aux options de compilation

–O2 et –O3 (pour C, C++ et Fortran), et nous pouvons également penser aux opérations Fused Multiply-Add (FMA), Advances Vector Extension (AVX), et à l'utilisation de cartes accélératrices avec des « Tensor Cores » capables de réaliser des multiplications de petites matrices en un cycle. Le FMA est une opération arithmétique qui combine la multiplication et l'addition en une seule instruction. La nature fusionnée de cette opération signifie que la multiplication et l'addition sont effectuées à l'intérieur du processeur au niveau matériel en une étape « unique », améliorant l'efficacité computationnelle des deux opérations suivantes : « a * b + c ». AVX est une extension du jeu d'instruction de l'architecture « x86 » qui permet des opérations SIMD (Single Instruction, Multiple Data). L'architecture SIMD permet au processeur de traiter plusieurs éléments de données en parallèle avec une seule instruction vectorielle, augmentant ainsi le débit de calcul. AVX introduit des registres vectoriels plus larges (par exemple, 256 ou 512 bits), permettant à une seule instruction vectorielle d'opérer simultanément sur plusieurs éléments de données en simple ou double précision. Cela est particulièrement bénéfique dans les applications spécifiques, où la même opération doit être effectuée simultanément sur plusieurs points de données. Par exemple, avec la norme AVX-512, un registre vectoriel de 512 bits peut contenir cent vingt-huit nombres à virgule flottante de simple précision (32 bits) ou soixante-quatre nombres à virgule flottante de double précision (64 bits). Lors de la compilation avec l'optimisation agressive –O3, toutes ces fonctionnalités d'optimisation peuvent conduire à une perte de répétabilité. Nous avons observé une perte de répétabilité d'une exécution à l'autre avec un contexte identique sur des superordinateurs comme décrit par le Prof. Thomas Ludwig, directeur du DKRZ lors de sa présentation à l'atelier reproductibilité de la conférence Européenne de super calcul de

Francfort en 2019 (Ludwig 2019). La suppression des optimisations vectorielles « agressives » permet de retrouver un comportement normal avec une performance moindre.

Avec la généralisation des exécutions dynamiques à l'intérieur des CPU (également connues sous le nom de « out-of-order », des changements de l'ordre d'exécution des instructions), créées pour mieux alimenter la pile d'exécution des CPU, nous observons depuis plus de problèmes avec l'arithmétique à virgule flottante en HPC. De plus, comme le montrent Mytkowicz *et al.* (2009), même l'évaluation de la performance d'une fonctionnalité d'optimisation comme –O3 peut être non reproductible.

5.4. Hétérogénéité du matériel

Dans un environnement de calcul haute performance, tel que les grilles de calcul utilisées par le CERN, les ressources matérielles peuvent être très hétérogènes. Cette hétérogénéité peut avoir des implications significatives pour la reproductibilité, car elle introduit des variations en termes de performance, de précision et de comportement d'exécution entre différents systèmes (Boyer 2022). En effet, l'hétérogénéité potentielle des plateformes matérielles et des réseaux soulève des préoccupations quant à la reproductibilité. La planification dynamique nécessaire pour s'adapter aux ressources et charges changeantes rend difficile l'exécution constante des opérations dans le même ordre lors de différentes exécutions sur une plateforme à mémoire distribuée. La figure 5 illustre un scénario de charge de travail standard pour les expériences LHCb du CERN. Dans cette représentation, divers nœuds de travail sont engagés dans la récupération et le traitement des tâches. Notamment, ces nœuds sont équipés d'un large éventail de composants matériels, illustrant le concept de calcul en grille dans un environnement de calcul à hautes performances.

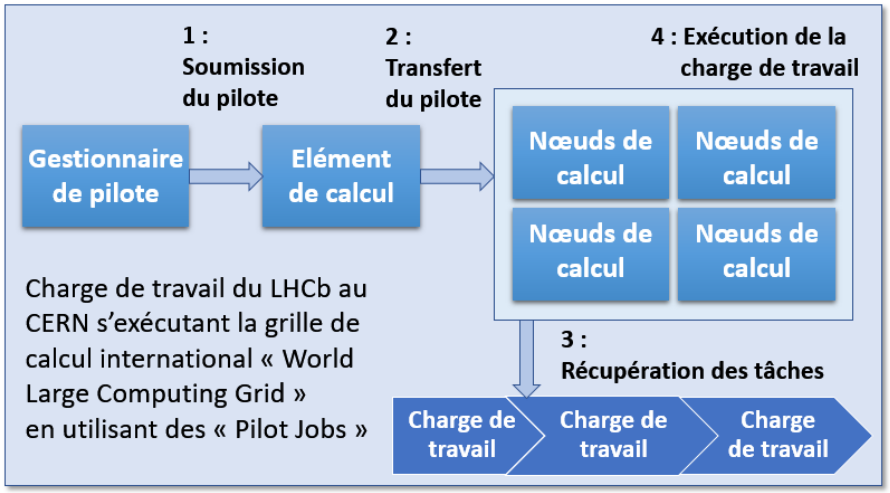

Figure 5. *Charge de travail typique pour les expériences du LHCb*

Nous pouvons également rencontrer une variabilité de performance en raison de l'hétérogénéité du matériel. Différentes architectures matérielles présentent des capacités de traitement variables, telles que la vitesse d'horloge des CPUs, le nombre de cœurs, la bande passante de la mémoire, les configurations du BIOS et la taille du cache. En conséquence, le même code exécuté sur différentes machines peut présenter des caractéristiques de performance différentes. Cette variabilité peut entraîner des écarts dans le temps nécessaire pour achever les calculs, rendant difficile la comparaison et la reproduction cohérente des résultats à travers diverses configurations matérielles.

De plus, les CPUs peuvent également être très différents au niveau de leur conception, mettant en œuvre différentes technologies et ensembles d'instructions, y compris SIMD (par exemple, AVX dont nous venons de parler), qui peuvent accélérer certains calculs. Cependant, le code utilisant des ensembles d'instructions spécifiques peut produire des résultats

différents sur des CPU qui ne prennent pas en charge ces instructions. Cette divergence peut entraîner des différences de performance et, dans certains cas, une divergence numérique dans les résultats. Cela peut également affecter la précision en virgule flottante (par exemple, double précision, précision étendue). L'utilisation de différentes précisions peut entraîner des variations dans la précision et l'exactitude numérique des résultats finaux. Dans certains cas, ces différences peuvent être négligeables, mais dans d'autres, lorsqu'elles sont répétées intensivement, elles peuvent affecter de manière significative le résultat final.

Toujours concernant le matériel, les mémoires peuvent varier, et avec elles, leurs performances. Des hiérarchies de mémoire diverses, telles que différentes tailles de cache et latences d'accès à la mémoire, peuvent affecter l'efficacité des applications intensives en mémoire (certaines applications étant limitée en performance par leurs accès à la mémoire). Cela aussi peut entraîner des performances variables.

Sur les grilles de calcul, nous cherchons également non seulement la parallélisation, mais aussi l'équilibrage de charge, et cet aspect est difficile. Un code optimisé pour une architecture parallèle spécifique peut ne pas utiliser efficacement d'autres architectures, conduisant à des schémas différents de parallélisation et, par conséquent, à différentes performances et résultats numériques.

Enfin, la pile logicielle peut être différente, comme les compilateurs, les bibliothèques, le système d'exploitation, etc. Toutes ces variations peuvent conduire à un code généré ou exécuté différent, modifiant potentiellement la performance et le comportement numérique de l'application.

Pour toutes ces raisons, nous pouvons faire face à des bogues spécifiques à la plateforme. De tels problèmes peuvent conduire à des

résultats non reproductibles sur certains systèmes, rendant difficile l'identification et la correction de la cause principale.

5.5. Informatique quantique

Une nouvelle branche du calcul haute performance émerge avec le calcul quantique. Cela est prometteur pour l'optimisation de la résolution de problèmes (Hogg et Portnov 2000). Des outils sont développés pour faciliter l'utilisation du calcul quantique, qui peut rester obscur pour les personnes habituées à l'informatique déterministe classique. (Shaydulin *et al.* 2021) proposent un toolkit Python nommé QAOAKit, conçu pour la recherche d'Algorithme d'Optimisation Quantique Approximatif (QAOA). Il sert de dépôt unifié de paramètres QAOA pré-optimisés et de générateurs de circuits quantiques. En intégrant des paramètres connus pour le problème MaxCut et en fournissant des outils de conversion pour divers cadres de simulation quantique, QAOAKit facilite la reproductibilité, la comparaison et l'extension des résultats de recherche en optimisation quantique, abordant des questions ouvertes sur la performance et le comportement de l'algorithme.

IBM est à l'avant-garde de l'accès aux machines quantiques avec leur plateforme IBM Quantum Experience. Cette plateforme permet aux chercheurs et aux passionnés de réaliser des expériences sur les processeurs quantiques d'IBM, qui peuvent avoir jusqu'à 127 qubits depuis 2023. Cet accès remarquable est facilité par Qiskit, un framework de développement de logiciels de calcul quantique open-source, qui permet aux utilisateurs d'écrire des algorithmes quantiques en Python. Il y a également QASM (Quantum Assembly Language), qui offre un moyen d'exprimer des circuits quantiques à un niveau inférieur, permettant un contrôle plus fin des opérations quantiques.

De plus, les méthodes de création de qubits sont variées et font l'objet de recherches actives. En général, les qubits peuvent être réalisés en utilisant plusieurs systèmes physiques. Les méthodes les plus utilisées incluent les ions piégés, où des ions individuels sont confinés et manipulés avec des champs électromagnétiques; les circuits supraconducteurs qui exploitent les propriétés mécaniques quantiques de circuits électroniques macroscopiques refroidis à près de zéro absolu; et les qubits à semi-conducteurs, qui sont fabriqués en utilisant des principes similaires à ceux des puces informatiques conventionnelles. Chacune de ces méthodes a ses propres avantages et défis, notamment en termes de passage à l'échelle, de temps de cohérence et de taux d'erreurs. Les chercheurs essayent continuellement d'améliorer ces systèmes, dans le but de créer des qubits plus fiables et durables qui pourraient ouvrir la voie à un calcul quantique plus utile.

Cependant, avec le calcul quantique pour l'optimisation haute performance, de nouveaux défis de reproductibilité apparaissent. Nous pouvons voir que la reproductibilité dans le calcul quantique est l'une des principales préoccupations, nous avons besoin de milliers d'excellents qubits pour obtenir un seul qubit « parfait ». Dans (Mauerer et Scherzinger 2022), ils restent à un niveau élevé, proposant des solutions pour gérer les paquets et les dépendances, ce qui est un problème qui n'est pas seulement propre au calcul quantique, au contraire. Si nous nous penchons plus profondément sur l'aspect technique du calcul quantique, nous pouvons en effet comprendre qu'il a des problèmes intrinsèques (Dasgupta et Humble 2021a; Dasgupta et Humble 2021b; Dasgupta et Humble 2022; Hill *et al.* 2023) en raison du comportement non déterministe, et du fait qu'il est physiquement difficile d'avoir des qubits fiables en raison de la décohérence, du bruit, du manque de correction d'erreur, etc. IBM offre

un accès gratuit pour utiliser ses machines quantiques, et ainsi, beaucoup de recherches sont menées sur leurs machines. Dans (Hill *et al.* 2023), les auteurs ont mis en œuvre un algorithme de Grover à trois qubits. L'oracle est codé (avec des portes quantiques) pour trouver la propriété « être égal à cinq ». Ils ont exécuté l'algorithme sur un simulateur quantique, montrant qu'ils devraient obtenir la bonne solution avec une probabilité de 94%; le calcul quantique étant stochastique par essence, les résultats sont donnés par probabilités. Mais lors de l'exécution sur différentes machines quantiques réelles, les résultats n'étaient pas si bons et ils ont trouvé beaucoup de variabilité. Au mieux, ils ont obtenu la bonne solution avec 60% (ce qui est bien moins que ce qui était prévu selon la simulation, mais est encore utilisable). Le problème de reproductibilité est survenu lors de l'exécution de cette expérience plusieurs fois sur la même machine quantique, ou sur différentes machines quantiques. Avec exactement le même code et les mêmes paramètres, ils ont obtenu à chaque fois des résultats statistiquement différents, la plupart n'étant pas ceux attendus. Cela remet en question l'utilisabilité des ordinateurs quantiques actuels, mais il faut noter des améliorations rapides lorsque nous avons testé le même algorithme sur 3 et 5 qubits un an plus tard (Antunes et Hill 2024). Les simulateurs ou les machines quantiques hybrides sont intéressants pour développer des compétences et dans un but d'apprentissage, mais nous ne pourrons produire des résultats scientifiques approfondis sans une reproductibilité fiable.

Le calcul quantique est une technologie jeune et passionnante. Il est probable que nous verrons bientôt des technologies innovantes et obtiendrons des machines quantiques générales avec des qubits très utilisables dans les prochaines décennies.

5.6. Apprentissage artificiel

Une utilisation récente des ressources de calcul à hautes performances traite de l'apprentissage artificiel avec des données massives (Big Data). L'apprentissage artificiel implique l'utilisation de l'aléatoire pour entraîner les modèles. Cependant, les bonnes pratiques pour la reproductibilité dans le monde de l'apprentissage artificiel ne sont pas si bien répandues. Il peut être difficile d'obtenir des séquences de nombres pseudo-aléatoires reproductibles pour les scientifiques utilisant des frameworks de haut niveau (Onose 2023). De plus, les frameworks d'apprentissage profond eux-mêmes subissent des mises à jour et des changements de version, ce qui peut introduire des écarts lors de la comparaison des résultats obtenus avec différentes versions de logiciels.

L'apprentissage artificiel est souvent associé au big data, qui peut également présenter des défis pour la reproductibilité, principalement en raison de la variabilité des ensembles de données ou de l'échantillonnage des données. À mesure que la taille des ensembles de données augmente, les techniques de filtrage et d'échantillonnage deviennent de plus en plus importantes. Différentes approches d'échantillonnage peuvent entraîner des variations dans les sous-ensembles de données utilisés pour l'analyse, et cela peut potentiellement impacter la reproductibilité des résultats. Lorsque nous considérons la classification, les résultats numériques exacts sont beaucoup moins importants. Cependant, le traitement parallèle et les systèmes distribués, couramment utilisés dans l'analyse de masses de données, peuvent contribuer à un comportement non déterministe, rendant la reproductibilité difficile. Les algorithmes parallèles peuvent présenter des comportements différents en fonction de facteurs tels que l'ordre d'exécution ou l'allocation des ressources informatiques. Par conséquent, obtenir les mêmes résultats à travers différentes exécutions

parallèles, simplement à des fins de débogage, peut devenir très difficile. De plus, lorsqu'on traite d'une quantité massive de données, il devient plus difficile de partager de telles données, et cela affecte la capacité d'autres chercheurs à reproduire les résultats. Avec (Collberg et Proebsting 2016), nous avons vu que les recherches financées par des institutions publiques sont plus facilement reproductible car le partage de code est une pratique plus courante. Les entreprises privées ont un poids lourd dans le monde de l'apprentissage automatique et sont moins enclines à partager leur code et leurs données puisqu'elles sont en concurrence pour obtenir un avantage industriel.

Enfin, beaucoup de ressources informatiques sont maintenant proposées par les unités de traitement graphique de type GPU (Graphical Processing Unit) qui sont devenues généraliste (GP-GPU : General Purpose-GPU). Ce matériel est largement utilisé dans le domaine de l'IA, en particulier pour l'entraînement de modèle, car les GPU modernes intègrent un grand nombre de « Tensor cores ». Lors de l'utilisation des GPU pour le calcul à hautes performances, la reproductibilité peut faire face à plusieurs défis. La précision en virgule flottante est moins importante car l'entraînement peut souvent être réalisé en demi-précision (FP16) ou parfois moins, les opérations de classification n'ayant pas les mêmes besoins en terme de précision. Cependant, des erreurs d'arrondi non déterministes peuvent survenir lors des calculs numériques. Cela peut entraîner de légères variations dans les résultats même lorsque le même code et les mêmes données sont utilisés. De plus, différentes architectures de GPU peuvent produire des résultats incohérents, rendant la reproductibilité difficile sur diverses configurations matérielles. Les bibliothèques et pilotes de GPU contribuent également aux défis de reproductibilité, leur maintenabilité est réduite dans le temps (parfois

seulement 5 ans). De nombreuses bibliothèques d'optimisation dans l'apprentissage artificiel et le Big Data sont spécifiques au fournisseur de matériel, s'appuyant sur des interfaces propriétaires et des fonctionnalités fournies par les fabricants de GPU. Les incompatibilités ou les différences entre les versions de bibliothèques ou les pilotes de GPU peuvent introduire des variations dans les résultats et entraver la reproductibilité. L'exécution parallèle sur les GPU peut également introduire des comportements « non déterministes » qui affectent davantage la reproductibilité. Les interférences entre les noyaux de GPU, les variations dans la planification des threads et les conditions de favorisation de la vitesse lors de l'exécution parallèle peuvent entraîner des sorties différentes à travers différentes exécutions, même avec les mêmes données d'entrée et le même code (Taufer *et al.* 2010; Jézéquel *et al.* 2015).

5.7. Erreurs silencieuses (ou « soft errors »)

Rarement mentionnées, les erreurs silencieuses jouent un rôle important dans la fiabilité du calcul à hautes performances. Les perturbations de nature électrique ou magnétique au sein d'un système informatique peuvent entraîner le basculement involontaire d'un bit de la mémoire vive dynamique (DRAM : Dynamic Random Access Memory) vers son état opposé. Initialement, ce phénomène était principalement attribué aux particules alpha émises par les impuretés dans l'emballage des puces. Les particules alpha ou la radiation alpha se composent de deux protons et deux neutrons, elles sont généralement émises pendant le processus de désintégration alpha, mais peuvent aussi être produites d'autres manières (Rutherford et Royds 1908). Cependant, Normand (1996) a démontré que la principale cause d'erreurs isolées et temporaires dans les puces DRAM est liée au rayonnement ambiant. Les neutrons,

provenant principalement des interactions secondaires des rayons cosmiques, ont été identifiés comme une source clé de ce rayonnement, capable d'altérer le contenu d'un ou plusieurs bits de mémoire ou de perturber les circuits responsables de leurs fonctions de lecture ou d'écriture. Une vaste étude, portant toujours sur la DRAM, a tenté de déterminer la fréquence des erreurs de mémoire (Schroeder *et al.* 2009). Dans cette étude, le comportement des erreurs de DRAM dans des scénarios réels s'écarte considérablement des hypothèses courantes. Notamment, les taux d'erreur observés de la DRAM sont nettement plus élevés que ce que l'on pensait auparavant, allant de 25 000 à 70 000 erreurs par milliard d'heures d'appareil par Mbit, et plus de 8% des modules de mémoire en ligne double (DIMM) connaissent des erreurs annuellement. Contrairement aux croyances antérieures, il est démontré que les erreurs matérielles, plutôt que les erreurs silencieuses (aussi appelées « soft errors »), dominent le paysage des erreurs de mémoire. Étonnamment, la température, qui est un facteur influent dans des environnements contrôlés, a un impact minimal sur le comportement des erreurs dans des contextes pratiques lorsqu'on tient compte d'autres variables.

Les CPU ne sont pas à l'abri de ce problème. Dans le domaine du calcul avancé, l'augmentation de la surface, la réduction des tailles de fabrication (gravures au nanomètre et en dessous dans les laboratoires) et la densité accrue des composants ont conduit à une augmentation des renversements de bits, qui peuvent causer des erreurs silencieuses. Bien que de telles anomalies soient supposées se produire plus fréquemment dans la DRAM, d'autres composants comme les portes logiques et les unités arithmétiques des CPUs peuvent également être vulnérables (Elliott *et al.* 2013). Cette étude vise à évaluer l'impact d'un renversement de bit unique sur des opérations en virgule flottante spécifiques. L'auteur se

penche sur les erreurs résultant du renversement de bits particuliers dans la représentation en virgule flottante IEEE, évitant de se fier à des informations propriétaires comme les taux de renversement de bits et les conceptions de circuits spécifiques aux fournisseurs. Dans (Dixit *et al.* 2021), ils considèrent également les CPU dans le cadre de la gestion d'erreurs matérielles. Ils argumentent : « *Par exemple, lorsque vous effectuez 2*3, le CPU peut donner un résultat de 5 au lieu de 6 silencieusement sous certaines conditions micro-architecturales, sans indication de la mauvaise opération dans les journaux d'événements ou d'erreurs du système. En conséquence, un service utilisant le CPU peut ne pas être conscient de l'inexactitude des calculs et continuer à consommer les valeurs incorrectes dans l'application.* »

De ces études, nous savons que la DRAM et les CPUs peuvent conduire à ce que nous appelons des erreurs silencieuses. Alors que la technologie évolue, en travaillant avec des puces toujours plus petites, nous augmentons la probabilité d'avoir des renversements de bit. Maintenant, à l'échelle Exascale, nous avons des superordinateurs comme Frontier qui possède 8 699 904 cœurs. À cette échelle, le temps moyen avant défaillance tombe à quelques heures de calcul seulement. Dans la section suivante, nous verrons les solutions proposées pour pallier ces problèmes.

Chapitre 6

Quelles solutions pour améliorer la reproductibilité ?

6.1. Gestion des versions et archivage

Pour garantir la reproductibilité des parties computationnelles d'un article, il est nécessaire d'effectuer un suivi de version, d'archiver, de documenter et de partager le code. Tout d'abord, le suivi de version est effectivement très important lorsqu'il s'agit de codes et de scripts. Imaginez un scénario avec une expérience informatique, aboutissant à des figures intéressantes. Ensuite, vous continuez à mettre à jour votre code. Plus tard, vous voulez retravailler sur ces figures, peut-être parce que vous avez soumis un article de recherche et que les relecteurs vous l'ont demandé, ou peut-être juste dans votre processus de recherche afin de mieux comprendre les figures produites. Comme vous avez modifié le code, vous êtes désormais incapable de générer les mêmes figures, vous n'obtenez plus les mêmes résultats intéressants et vous n'avez pas l'ancienne version du code. Cette situation n'est pas rare et aide à comprendre le besoin d'outils pour contrôler les versions du code. Deuxièmement, il est crucial d'archiver votre code de manière permanente. Le code d'erreur HTML « 404 not found » apparaît bien plus souvent qu'il ne le devrait, nous devons faire attention à l'endroit où nous stockons nos différentes versions de code. Troisièmement, vous devez documenter votre code afin que d'autres personnes puissent l'utiliser et le

comprendre. Et enfin, vous devez partager votre code pour que les gens puissent y accéder (Kitzes *et al.* 2018).

Concernant les outils de suivi de version, deux des plus importants sont Git et Apache Subversion. Git est devenu l'outil dominant dans ce domaine et est largement utilisé pour le suivi de version. Git offre de nombreuses fonctionnalités, dans le cadre d'une recherche reproductible, nous préconisons son intégration dans le processus de développement, bien que les débutants puissent avoir besoin d'une phase d'apprentissage initiale. En essence, adopter Git représente une des bases de bonnes pratiques en génie logiciel.

Plusieurs plateformes sont disponibles pour l'archivage du code. Github et Gitlab sont parmi les plus courantes. Ils fonctionnent comme des dépôts, où l'on peut stocker son code. Lors du développement d'un code pour un projet de recherche, l'utilisation de l'un de ces outils, en conjonction avec Git, est presque obligatoire. Cependant, ces outils n'ont pas été initialement conçus pour la recherche reproductible. Il convient de noter que ces plateformes, étant des entités commerciales (dans le cas de Github), ne garantissent pas l'accessibilité perpétuelle du code. Des alternatives comme Zenodo et Software Heritage offrent des solutions plus robustes pour l'archivage du code (Hinsen 2020b). Développés respectivement par le CERN et Inria, Zenodo et Software Heritage fournissent des moyens d'archiver du code, des données ou toute ressource numérique associée à un article, chacun identifié par un ID unique (soit DOI ou SWHID). Zenodo est davantage orienté vers l'archivage du code et des données d'un article, tandis que Software Heritage est un projet plus vaste, comprenant le stockage de toutes les versions existantes de code open source dans le monde, telles que les bibliothèques et non seulement les artefacts d'articles de recherche.

6.2. Programmation lettrée et documentation

Pour résoudre le problème de la documentation (rendre le code aussi clair que possible) et, peut-être aussi un peu, pour résoudre le problème des workflows, la programmation lettrée a été introduite. Il est essentiel, lors du partage de code, de le rendre accessible (dans le sens de la facilité d'utilisation). À cette fin, le concept de Programmation lettrée (Literate Programming) a émergé en 1984 avec Knuth (1984). Le principe est d'entrelacer le code avec du texte en langage naturel afin de rendre l'ensemble du contenu plus compréhensible. L'application actuelle de ce principe prend la forme de Notebooks : des documents interactifs qui permettent de combiner du texte avec des blocs de code. Ils se présentent sous différentes formes pour différents langages de programmation. Pour faciliter la rédaction de ce type de fichier, des langages de composition de texte sont utilisés : principalement LaTeX et Markdown, ce dernier étant beaucoup plus léger et compréhensible sans interpréteur, comme discuté par Pouzat (2021). Knuth avait développé l'outil initial pour la programmation lettrée, appelé WEB, qui comprenait deux programmes principaux, Tangle et Weave (Knuth 1984). Ce système était adapté pour le langage de programmation Pascal et générait des documents formatés en utilisant TeX. Plus tard, Knuth et Levy (1994) ont créé une version pour le langage C, nommée cweb. Une évolution contemporaine de ces outils est noweb (Johnson et Johnson 1997), conçue pour être adaptable à différents langages. Ses programmes principaux, notangle et noweave, sont tous deux codés en C. Les documents produits via noweave peuvent être formatés en utilisant TeX, LaTeX ou troff, ou même être affichés dans un navigateur web sous forme HTML. Des utilitaires logiciels comme WEB, cweb et noweb permettent aux auteurs de créer à la fois du contenu écrit et du code, mais ils ne disposent pas de mécanismes pour

exécuter directement le code dans les documents. Au lieu de cela, le code destiné à l'exécution est extrait, résultant en des fichiers de code source qui sont ensuite transmis à un compilateur ou interpréteur. Un des outils le plus largement adopté pour réaliser une recherche reproductible était Sweave, qui voit une utilisation croissante au sein de la communauté de programmation R. L'utilisation de Sweave pour la recherche reproductible a donné naissance à des outils analogues comme SASweave et Statweave. Certains de ces outils sont destinés à des langages statistiques autres que R et sont compatibles avec d'autres systèmes d'écriture de documents que LaTeX, englobant des formats comme les formats OpenDocument et Microsoft Word (Lenth et Højsgaard 2007; Baier et Neuwirth 2007; Lenth 2012). Sweave et ses variantes ne permettent pas de modifier l'ordre des blocs de code lors du processus de « tangling ». De ce fait, leur capacité à supporter la programmation lettrée est seulement partielle. Aujourd'hui, parmi les outils de programmation littéraire les plus connus, nous pouvons mentionner :

- Jupyter (Kluyver *et al.* 2016), un notebook principalement utilisé avec le langage Python.
- Rstudio (Allaire 2012), développé principalement pour le langage R.
- Org-mode (Schulte *et al.* 2012), un outil beaucoup plus polyvalent, utilisable avec tous les langages.

Ces trois outils sont présentés en détail dans le MOOC de recherche reproductible de l'Inria (Pouzat *et al.* 2018) animé par Arnaud Legrand, Konrad Hinsen et Christophe Pouzat. L'utilisation de notebook et, plus largement, la programmation lettrée, est vivement encouragée par la communauté scientifique. Stanisic *et al.* (2015) proposent une solution de workflow pour la recherche reproductible basée sur l'utilisation de Git et Og-mode. Desquilbet *et al.* (2019) discutent de l'utilisation des notebooks

dans leur livre sur la recherche reproductible. Stodden *et al.* (2013) compilent une liste d'outils pour la recherche reproductible, les notebooks en faisant partie intégrante. Ragan-Kelley *et al.* (2018) proposent même un outil nommé Binder, permettant l'exécution directe des dépôts de notebooks Jupyter sur le web sans installation, en reliant à un dépôt Git. Delescluse *et al.* (2012) mentionnent également Org-mode et Sweave pour la recherche reproductible, dans le contexte du langage R.

Avec l'augmentation de l'utilisation du langage de programmation Python, Jupyter est probablement le notebook le plus utilisé et le plus facile à prendre en main. Cependant, pour sa polyvalence, Org-mode est probablement l'outil le plus plébiscité, en particulier pour les amateurs de Emacs, car Org-mode se base sur cet éditeur de texte (Stallman 1981).

Lorsque l'on traite de grands projets logiciels complexes, on ne peut pas les coder entièrement en utilisant la programmation lettrée. Dans ce cas, le projet sera codé en utilisant un schéma de développement plus classique (génie logiciel), et on utilisera la programmation lettrée en finalité pour l'exécution ou l'analyse des résultats et des données. Par conséquent, pour traiter de tels projets, vous aurez besoin d'autres outils de documentation que la simple programmation lettrée. Il n'y a pas qu'une seule manière de faire une bonne documentation (Sommerville 2001), et vous aurez besoin de bonnes pratiques issues du génie logiciel. Une manière répandue de documenter du code orienté objet est de produire des diagrammes pour modéliser le code en utilisant le langage de modélisation unifié (UML) (Booch *et al.* 1996).

En plus de la programmation lettrée et d'une documentation plus complète avec des outils de modélisation, l'adhésion aux principes FAIR peut contribuer à renforcer la reproductibilité dans la recherche scientifique. Ces principes plaident en faveur de la facilitation de la

recherche d'éléments numériques tels que les données, les algorithmes et les outils, en les rendant facilement trouvables, accessibles, interopérables et réutilisables. La mise en œuvre de ces principes garantit que les résultats de la recherche sont identifiables grâce à des identifiants uniques et des métadonnées riches, accessibles via des protocoles bien définis, interopérables grâce à l'utilisation de formats et de vocabulaires standard, et réutilisables avec des licences d'utilisation claires. L'utilisation des principes FAIR facilite la reproduction des résultats de recherche et favorise également la collaboration et l'innovation en permettant aux chercheurs de partager facilement et de s'appuyer mutuellement sur leur travail (Wilkinson *et al.* 2016).

6.3. Workflow

Le workflow d'un article de recherche est une partie importante de la recherche reproductible. Il est essentiel de comprendre l'ordre dans lequel les opérations doivent être exécutées. L'un des premiers outils à émerger est le fichier makefile. L'outil make associé a permis l'automatisation de la construction d'un exécutable, mais pas seulement. Le premier article à le considérer à cette fin est paru en 2000 (Schwab *et al.* 2000), avec Jon Claerbout, que nous avions déjà vu comme l'un des premiers contributeurs de la recherche reproductible en 1992. Schwab *et al.* décrivent comment utiliser « make » et des outils associés, ainsi que des conventions de nommage pour les fichiers, afin de garantir que les lecteurs ainsi que les auteurs puissent être capable de reproduire les calculs. Avec (Cohen-Boulakia *et al.* 2017), nous pouvons voir que la biologie est l'un des domaines qui possède le plus d'outils de gestion des workflows pour améliorer la reproductibilité, peut-être parce que la biologie est un domaine qui produit et analyse beaucoup de données. Dans (Oinn *et al.*

2004), nous pouvons lire « les expériences in silico en bioinformatique impliquent l'utilisation conjointe d'outils informatiques et de stockage d'informations. Un nombre croissant de ces ressources est mis à disposition avec un accès sous forme de services Web. Les scientifiques en bioinformatique devront orchestrer ces services Web dans des workflows dans le cadre de leurs analyses ». De nombreux outils de workflow ont été créés depuis le fichier makefile en 2000 (plus de 300) (Amstutz *et al.* 2024). Ils visent à être faciles d'utilisation, souvent en fournissant une interface graphique (GUI). Taverna (Oinn *et al.* 2004) était un système de gestion de workflow scientifique open source qui fournit un environnement pour la conception de workflows et un moteur d'exécution de ces workflows. Taverna optimise la structure du workflow, relie les services Web et les activités, et prend en charge l'exécution sur des grilles de calcul ou du cloud computing. Cependant, le projet est désormais abandonné. Galaxy (Giardine *et al.* 2005) est un système de workflow populaire dans la communauté bioinformatique qui utilise depuis peu un format de spécification de workflow JSON ou YAML. Galaxy fut compatible avec Common Workflow Language (CWL), mais ce n'est plus le cas (Amstutz *et al.* 2016). Il fournit une interface graphique pour explorer et partager les exécutions de workflows. Galaxy gère les dépendances des activités pour la parallélisation, génère et surveille les tâches, et utilise une planification dynamique pour la distribution des tâches. OpenAlea (Pradal *et al.* 2015) est un système de workflow scientifique open source lié à un outil de modélisation basé sur le langage Python. Il permet aux utilisateurs d'exporter des workflows dans un format CWL. Cependant, il présente des limites en termes de complexité, de manque d'un dépôt centralisé et de difficulté à visualiser la provenance des données. Nextflow (Di Tommaso *et al.* 2014) est un système de

workflow en ligne de commande pour la gestion de workflows scientifiques parallèles complexes. Il utilise une spécification basée sur du texte et prend en charge des workflows de processeurs et opérateurs. Il présente des limites en termes de gestion des dépendances des outils et de manque d'informations de provenance de données. Pegasus, développé par Deelman *et al.* en 2004 (Deelman *et al.* 2004), est un outil utilisé dans plusieurs domaines scientifiques. Il offre plusieurs caractéristiques importantes comme la capacité de s'adapter à différents environnements (portabilité), l'utilisation d'algorithmes avancés pour organiser les tâches de manière optimale, le passage à l'échelle, la gestion efficace des informations sur l'origine des données, et la tolérance aux pannes. Pegasus est composé de cinq éléments principaux : un moteur qui associe les tâches aux ressources disponibles, un système pour exécuter les tâches sur l'ordinateur local, un ordonnanceur de tâches, un moteur d'exécution à distance, et un dispositif pour surveiller le processus. Ensemble, ces éléments permettent de créer et d'exécuter un workflow efficace sur différentes plateformes informatiques. Swift (Zhao *et al.* 2007), tout comme Pegasus, est utilisé dans de nombreuses disciplines et se spécialise dans les workflows qui nécessitent une grande quantité de données. Il gère ces workflows en plusieurs étapes : définition du programme, planification, exécution, gestion de la provenance et approvisionnement. Swift supporte la division des workflows, la résilience aux erreurs, et l'allocation dynamique de ressources sur divers sites d'exécution. Kepler (Altintas *et al.* 2004) est un système de gestion de workflow qui permet différentes méthodes d'exécution. Il intègre une interface graphique. Kepler est capable de planifier de manière statique ou dynamique, offre une tolérance aux pannes et peut exécuter des workflows sur des services Web, des systèmes basés sur des grilles de calcul ou utilisant le framework

Hadoop (Borthakur 2007). Chiron (Ogasawara *et al.* 2013) adopte une approche basée sur une base de données pour gérer l'exécution parallèle de workflows scientifiques intensifs en données. Il utilise un modèle algébrique et des opérateurs pour gérer les données et les activités du workflow. Chiron assure le suivi du workflow, gère différents types de parallélisme (par les données, indépendant, en pipeline) et propose une planification dynamique. Il stocke les informations d'exécution et de provenance dans une base de données structurée. Triana (Taylor *et al.* 2007) est un outil doté d'une interface graphique, développé initialement pour l'analyse de données dans le projet GEO 600. Triana est conçu pour les applications distribuées. Enfin, Snakemake est un moteur de workflow avancé offrant un langage de définition de workflow basé sur Python, facile à lire. Il permet une exécution flexible, allant d'un seul cœur à la distribution de calculs parallèles sur des clusters avec de nombreux cœurs, sans nécessiter de modifier le workflow. Cet outil est particulièrement réputé dans le domaine de la bioinformatique (Köster et Rahmann 2012).

Taverna, Chiron, Triana et Galaxy ont des fonctionnalités telles que l'interface graphique pour la conception des workflows, la prise en charge du parallélisme, la planification dynamique et l'exécution de workflows dans des environnements de grille de calcul et de cloud computing. Taverna prend en charge le partage d'informations sur les workflows, tandis que Galaxy se spécialise dans l'exécution de workflow en bioinformatique. Sequanix propose une interface graphique dynamique pour Snakemake (Desvillechabrol *et al.* 2018). Un état de l'art sur la gestion des systèmes de workflow est disponible (Liu *et al.* 2015), et un article sur les workflows plus axée sur les sciences de la vie a également été publiée en 2017 (Cohen-Boulakia *et al.* 2017), tandis que dans (Ivie et Thain 2018),

Ivie et Thain proposent un état de l'art plus général sur la gestion des workflows en recherche reproductible.

6.4. Environnement logiciel

6.4.1. *Gérer les dépendances*

Lorsque vous écrivez un programme, vous vous reposez toujours sur votre environnement logiciel. À la fin, lorsque vous publierez vos résultats, d'autres chercheurs pourraient rencontrer des obstacles pour relancer votre programme. Il se peut qu'ils ne puissent pas reproduire vos résultats car ils n'ont pas le même environnement logiciel que vous. Avoir une bonne documentation ne suffit pas, tout comme décrire le workflow ou archiver l'historique des versions du code.

Plusieurs solutions existent pour gérer les dépendances logicielles. Par exemple, l'utilisation de « apt-get » sur les distributions Linux, ou l'utilisation de « pip » lors de travaux avec Python. Apt-get est un système de gestion de paquets qui permet l'installation, la mise à jour et la suppression facile de paquets logiciels. Il résout automatiquement les dépendances, garantissant que toutes les bibliothèques et composants logiciels nécessaires sont installés. Pip, quant à lui, est un installateur de paquets pour Python qui simplifie la gestion des bibliothèques Python et de leurs dépendances. Il permet aux utilisateurs d'installer, de mettre à jour et de supprimer facilement des paquets Python à partir de l'Index des paquets Python (PyPI). Mais ils nécessitent tous les deux l'installation manuelle des paquets (même s'ils peuvent gérer par eux-mêmes les dépendances entre ces paquets). Même si de tels outils sont bons lorsqu'ils sont utilisés seuls, ils ne sont pas adaptés à la recherche reproductible. Le langage Python est désormais adopté pour de nombreuses expérimentations scientifiques. Il existe des outils spécifiques pour Python

tels que Conda, qui est un gestionnaire de paquets et d'environnements couramment utilisé dans la communauté scientifique Python. Conda permet l'installation à la fois de paquets Python et de paquets non-Python, ce qui le rend utile pour gérer des environnements logiciels complexes avec des dépendances qui vont au-delà des bibliothèques Python. Enfin, une fonctionnalité plus avancée vous permet de créer un environnement virtuel. Venv est un outil qui aide à créer des environnements Python isolés. Il permet aux utilisateurs de créer des environnements distincts pour différents projets, évitant ainsi les conflits entre les dépendances. Lorsqu'un nouvel environnement virtuel est créé, il est livré avec sa propre copie de l'interpréteur Python et une copie locale de l'outil pip. Cela garantit que toutes les bibliothèques installées dans l'environnement virtuel sont isolées de l'environnement Python global, ce qui facilite la gestion et la reproduction des dépendances spécifiques au projet. Venv est particulièrement utile lorsque l'on travaille sur plusieurs projets ou que l'on teste différentes versions de bibliothèques.

Bien que des outils tels que apt-get, pip, Conda et Venv soient largement utilisés et offrent des moyens pratiques de gérer les environnements logiciels et les dépendances, ils présentent des limites par rapport à des outils plus spécifiques conçus pour la reproductibilité. Tout d'abord, comme ils ne sont pas conçus pour la reproductibilité, la stabilité des environnements logiciels peut être difficile à atteindre avec ces outils, car les versions exactes et les configurations des paquets ne sont pas forcément explicitement capturées. Cela peut entraîner une perte de reproductibilité. Par exemple, si vous utilisez une bibliothèque telle que Pandas ou Numpy en Python, les développeurs pourraient modifier certaines parties du code sans changer le numéro de version, de sorte que des outils comme pip (et ceux qui en dépendent) ne pourraient pas

clairement identifier les deux versions du code. Une autre faiblesse de ces outils est qu'ils nécessitent souvent des privilèges d'administration pour installer des paquets à l'échelle d'un système d'exploitation, ce qui n'est pas adapté à un environnement de calcul à hautes performances, où vous n'aurez pas les privilèges administrateurs. Venv offre un certain niveau d'isolation, mais il se limite aux bibliothèques Python uniquement.

Pour ces raisons, certains outils ont émergé pour fournir une solution satisfaisante pour améliorer la reproductibilité de la partie computationnelle d'un article de recherche. CDE (Guo et Engler 2011) est un outil conçu pour résoudre les problèmes de dépendance. Il regroupe le code, les données et l'environnement. Aucun accès root n'est nécessaire. Voici comment fonctionne l'outil : il utilise ptrace et fait un paquetage automatisé du code, des données et de l'environnement pour s'exécuter sur une machine Linux. Un problème connu est qu'il pourrait ne pas inclure toutes les dépendances, qui devraient être ajoutées manuellement. De plus, il ne fonctionne qu'avec des noyaux Linux compatibles. Il fonctionne de la manière suivante: Alice lance la commande « cde script.py data.dat ». Cette action exécute le script Python, capture des informations via ptrace et crée un répertoire « cde-package ». Alice compresse le répertoire et l'envoie à Bob. Bob exécute ensuite la commande « cde-exec script.py data.dat ». Au lieu de rechercher des bibliothèques dans le chemin de Bob, cde-exec utilise ptrace ou strace pour modifier l'utilisation des bibliothèques et utilise celles stockées dans le dossier envoyé par Alice. Il peut y avoir une perte de performance potentielle allant de 2% à 28%. Cette solution peut ne pas être adaptée à un environnement HPC en raison de la perte potentielle de performance. L'outil Sumatra (Davison 2012) améliore la recherche reproductible en fournissant un système de capture automatisée des métadonnées. La bibliothèque principale de Sumatra met

en œuvre des fonctionnalités telles que la capture de l'environnement matériel et logiciel, la capture des données d'entrée et de sortie, et la capture du contexte scientifique. Cette fonctionnalité de base peut être utilisée par différentes interfaces qui couvrent les différentes méthodes de travail (ligne de commande, interface graphique, etc.). Les outils Sumatra n'ont pas nécessairement besoin de capturer toutes les informations pour être efficaces, plus il y a d'informations enregistrées, plus il est facile de reproduire les résultats à l'avenir. Sumatra est conçu pour être facile à utiliser et ne pas ralentir le workflow habituel des scientifiques. Les outils de Sumatra se composent d'une bibliothèque de base implémentée en Python, d'un outil en ligne de commande et d'une interface basée sur un navigateur web. L'outil en ligne de commande permet de capturer le contexte de calcul, les entrées et les sorties, ainsi que de visualiser les calculs précédents. L'interface basée sur un navigateur web offre des fonctionnalités supplémentaires pour visualiser, rechercher et annoter les enregistrements de calcul. De même que CDE, ReproZip (Chirigati *et al.* 2016) est un outil qui vise à rendre les expériences computationnelles reproductibles sur différentes plates-formes et ceci longtemps après leur création. Il capture automatiquement la provenance d'une expérience en traçant les appels système, en utilisant ptrace, et utilise ces informations pour créer un package reproductible léger qui inclut uniquement les fichiers nécessaires à sa reproduction. ReproZip ajoute une fonctionnalité pour être compatible non seulement avec les systèmes Linux, mais aussi avec Windows ou MacOS. Ses limites concernent les environnements distribués, tels que les clusters MPI ou Hadoop. Cependant, ni Sumatra ni ReproZip ne fournissent une analyse des performances, comme le fait CDE. Comme ReproZip utilise ptrace de même que CDE, une hypothèse plausible serait que les performances devraient être similaires. En fin de

compte, ces outils sont conçus pour la recherche reproductible, mais ne sont probablement pas adaptés au calcul à hautes performances.

Étant donné que la manipulation de tels outils nécessite une phase d'apprentissage, l'objectif final serait de trouver un outil qui puisse être utilisé par le plus de monde possible et qui soit adapté au calcul à hautes performances. Pour gérer les dépendances de bibliothèques, l'outil que nous recommandons est Guix (Courtès et Wurmus 2015). Guix est un outil de gestion de logiciels qui adopte une méthode fonctionnelle pour gérer les dépendances des logiciels. Selon cette méthode, les étapes de construction et d'installation des logiciels sont considérées comme des fonctions pures. Autrement dit, les résultats de ces étapes dépendent uniquement des données fournies en entrée. Cette technique permet de stocker efficacement les résultats sur le disque dur, assurant que les mêmes données d'entrée produiront toujours les mêmes résultats. Pour appliquer cette méthode, Guix exerce un contrôle rigoureux sur l'environnement de construction. Inspiré par Nix (Dolstra *et al.* 2004), Guix utilise un service système (daemon) avec des privilèges administrateurs pour créer des logiciels dans des conteneurs Linux isolés, notamment dans un environnement chroot. Ces conteneurs disposent de leurs propres identifiants utilisateurs et espaces de noms isolés pour différents aspects tels que la gestion des processus, la communication entre processus, le réseau, etc. L'environnement chroot se limite aux répertoires spécifiquement déclarés, empêchant ainsi l'accès à des outils ou bibliothèques non autorisés durant la construction. Les espaces de noms isolés empêchent également toute communication avec l'extérieur durant la construction. Les résultats de chaque construction sont enregistrés dans un espace de stockage commun, habituellement situé dans /gnu/store. Chaque élément dans ce stockage porte un nom comprenant un hash basé

sur toutes les données d'entrée utilisées lors de la construction. Ce hash inclut non seulement les compilateurs et les bibliothèques, mais aussi les scripts de construction et les variables d'environnement. Ce processus tient compte de l'ensemble des dépendances de chaque logiciel, incluant récursivement les outils et bibliothèques utilisés. Cette méthode garantit la reproductibilité des constructions, permettant ainsi au système de retracer l'intégralité du réseau de dépendances pour chaque logiciel construit.

Nous avons constaté que Guix est probablement le meilleur outil pour gérer les dépendances. De plus, cet outil ne fournit pas seulement un fichier « binaire » illisible à exécuter pour reproduire n'importe quel traitement (mais dans ce cas, nous ne savons pas vraiment ce que nous exécutons), mais il offre une vision claire et complète de l'environnement logiciel que nous utilisons. Chaque version est identifiée par un hash unique, avec la source disponible sur un dépôt en ligne Guix. Un tutoriel clair et facile à utiliser a été proposé en 2023, il permet de produire des articles de recherche reproductibles[1]. Nous recommandons la lecture de ce tutoriel. Les inconvénients de Guix sont qu'il n'est pas encore très répandu sur les clusters de calcul, et que le « daemon » s'exécutant en fond nécessite des privilèges administrateurs, de sorte que la première configuration et l'installation de Guix doivent être effectuées par un administrateur système du cluster. Lorsque vous travaillez localement avec Guix, il vous permet, si les futurs utilisateurs n'ont pas Guix, de générer une image Docker ou une image Apptainer (nouveau nom du container Singularity adapté au calcul à hautes performances). Nous présenterons ces deux outils par la suite.

[1] https://hpc.guix.info/blog/2023/06/a-guide-to-reproducible-research-papers/- dernier accès le 25/04/2024.

6.4.2. *Machines virtuelles*

Pour rendre la partie informatique de vos recherches plus reproductible, un autre type d'outil existant est celui des machines virtuelles (VM). Une des premières mentions détaillées de ce concept remonte à une cinquantaine d'années (Goldberg 1974). Avec les machines virtuelles modernes, nous utilisons un hyperviseur comme plateforme de virtualisation permettant à plusieurs systèmes d'exploitation de fonctionner simultanément sur une seule machine physique. Il existe actuellement deux types d'hyperviseurs, comme décrit dans la Figure 5, natifs et hébergé (hosted). L'hyperviseur natif, également connu sous le nom de « bare metal », est un logiciel qui s'exécute directement sur le matériel; l'hyperviseur natif interagit directement avec le matériel de l'hôte, fournissant une plateforme sur laquelle plusieurs systèmes d'exploitation peuvent être exécutés par l'hyperviseur. Ce type d'hyperviseur est léger et optimisé pour le noyau de l'hôte. A contrario, l'hyperviseur hébergé est un logiciel qui s'exécute à l'intérieur d'un autre système d'exploitation.

Une machine virtuelle offre un système d'exploitation indépendant, spécialement conçu pour vos expérimentations. Cette isolation permet de capturer et de reproduire fidèlement tous les processus et paramètres impliqués dans vos recherches. Cette caractéristique est particulièrement utile pour permettre à d'autres chercheurs de reproduire vos expériences dans un environnement identique. Toutefois, l'usage de machines virtuelles peut entraîner une réduction des performances, ce qui peut être un frein pour les applications nécessitant une grande puissance de calcul. Malgré cela, pour la plupart des applications, les avantages de reproductibilité et d'isolation offerts par les machines virtuelles compensent largement ce désavantage. Dans (Stodden *et al.* 2013), les auteurs mentionnent VirtualBox et VMWare comme outils permettant la

création et l'utilisation de machines virtuelles pour la recherche reproductible. De même dans (Ruiz *et al.* 2014). Ces deux outils utilisent le deuxième type d'hyperviseur (hosted) qui est beaucoup plus facile à utiliser (la facilité d'utilisation étant un critère important pour la recherche reproductible). Mais celui peut aussi entraîner plus de perte de performance (ce qui n'est pas adapté au HPC). Certaines publications étudient les performances de la virtualisation de type 1, tandis que d'autres se penchent sur le type 2, souvent dans le contexte de l'informatique en nuage (ou « cloud computing »). Dans ce contexte, les machines virtuelles sont omniprésentes pour fournir un système d'exploitation utilisable à de nombreuses personnes en se basant sur le même matériel sous-jacent. Le but principal n'est donc pas ici la recherche reproductible, mais simplement proposer de l'accès à des machines via le « cloud ».

Concernant les hyperviseurs de type 1, Gilbert *et al.* (2005) évaluent les performances de la virtualisation dans le contexte des applications de physique de hautes énergies au CERN dans un environnement de grille de calcul. Leurs résultats ont montré que la virtualisation peut avoir jusqu'à 15 % de perte de performance par rapport au temps d'exécution. Ils travaillaient avec un hyperviseur de type 1, ESX VMware. Dans (Matthews *et al.* 2007), les auteurs n'ont pas trouvé que des outils de virtualisation tels que Xen ou VMware génèrent une perte de performance, tandis que dans (Padala *et al.* 2007), les auteurs trouvent que Xen génère beaucoup de perte de performance, probablement en raison d'un nombre plus élevé de perte de données dans le cache de niveau 2 (L2 cache miss). Dans (Acharya *et al.* 2018), leurs résultats montrent que sur les processeurs x86, l'hyperviseur (en utilisant KVM) fonctionne moins bien que d'autres technologies. Cependant, ce n'est pas le cas sur les processeurs ARM.

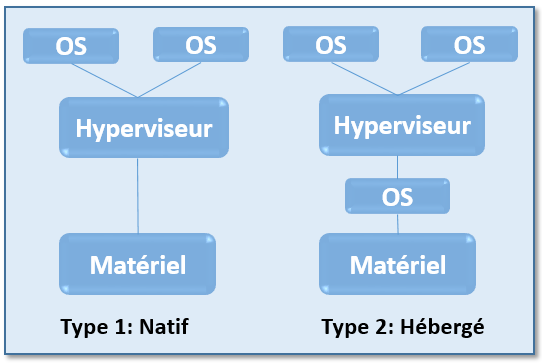

Figure 5. *Hyperviseur type 1 et hyperviseur type 2*

Tous les articles cités précédemment évaluent les performances des hyperviseurs natif (type 1). Et nous pouvons constater que ce n'est probablement pas une solution adaptée pour le calcul à hautes performances en raison de la perte de performance induite par leur utilisation, mais ce n'est également pas adapté à la recherche reproductible en raison de sa complexité de mise en place. Nous n'avons trouvé aucun article sur la recherche reproductible évoquant l'utilisation des technologies et outils des hyperviseurs de type 1. Cependant, pour les hyperviseurs hébergés (type 2), tels que VirtualBox et VMware (dans sa version hébergée), il existe des publications du monde de la recherche reproductible qui évoquent leur utilisation pour rendre les articles plus facilement reproductibles.

Dans (Đorđević *et al.* 2021), un article récent de 2021, les auteurs étudient les performances de VirtualBox par rapport à une exécution native, et concluent que « en ce qui concerne VirtualBox, la chute de performance est notable ». Dans (Beserra *et al.* 2015), les auteurs se situent dans le contexte du calcul haute performance, contrairement aux travaux précédents. Ils ont cherché à analyser les performances de KVM

(hyperviseur de type 1) et de VirtualBox (hyperviseur de type 2). Pour le traitement, ils ont effectué trois tests différents (HPL, DGEMM et FFT). Sur le premier, KVM a atteint des performances proches des performances natives et a obtenu 14% de performance de plus que VirtualBox. Sur le deuxième, KVM a eu des performances 5% inférieures à une exécution native, mais a performé 10% mieux que VirtualBox. Sur le troisième test (FFT), KVM et la performance native ont performé 30% mieux que VirtualBox. En ce qui concerne l'accès à la mémoire, VirtualBox a performé environ 25% moins bien que KVM et l'exécution native. Dans l'ensemble, on peut clairement voir que les hyperviseurs de type 2 ne conviennent pas au calcul à hautes performances. Un document fusionnant l'utilisation de la virtualisation et du cloud computing dans le cadre de la recherche reproductible a été proposé par Bill Howe (Howe 2012). Il a proposé d'utiliser les services d'Amazon pour collaborer via le cloud. Cependant, l'utilisation d'une entreprise privée en tant que tiers n'est pas toujours adaptée à la science ouverte internationale.

Un autre inconvénient de l'utilisation de machines virtuelles pour la recherche reproductible est leur opacité et leur demande en ressources. La distribution de gros fichiers de machines virtuelles sur les réseaux peut être fastidieuse, et le contenu d'une machine virtuelle reste obscur jusqu'à son exécution. Bien qu'elles permettent une certaine reproductibilité, les machines virtuelles ne sont pas optimales en termes de passage à l'échelle, d'adaptabilité et de performance, engendrant par ce dernier aspect un coût énergétique supplémentaire dans le contexte où cette consommation est très sensible (calcul intensif).

6.4.3. *Conteneurs*

Les conteneurs sont un autre type de virtualisation plus léger que les machines virtuelles et qui fonctionne au niveau du système d'exploitation. Les conteneurs et les technologies de virtualisation matérielles sont similaires en ce qu'ils permettent à plusieurs applications isolées de s'exécuter sur une seule machine hôte. Cependant, il existe des différences significatives dans la manière dont ils y parviennent. La virtualisation matérielle implique l'exécution d'un système d'exploitation complet dans un environnement isolé. Cela nécessite une quantité importante de ressources, notamment de la mémoire et de l'espace de stockage. Le démarrage d'un système d'exploitation invité peut prendre plusieurs minutes. En revanche, les conteneurs partagent le noyau sous-jacent du système d'exploitation hôte et isolent les processus s'exécutant à l'intérieur de ceux-ci des autres processus s'exécutant sur l'hôte. Cela signifie que les conteneurs utilisent directement les fonctionnalités de l'hôte, ce qui se traduit par une utilisation plus efficace des ressources. Plusieurs conteneurs peuvent être exécutés sur un seul hôte, et le démarrage d'un nouveau conteneur est rapide, similaire au démarrage d'une application native. Les conteneurs sont donc en quelque sorte des paquets autonomes et légers qui incluent une application et toutes ses dépendances, partageant le noyau du système d'exploitation hôte. Comparés aux machines virtuelles, ils sont beaucoup plus petits et plus efficaces en termes de ressources, ce qui permet un démarrage rapide avec une utilisation réduite des ressources. Ils ont été initialement conçus pour un déploiement consistant dans divers environnements, du développement à la mise en production. Les conteneurs offrent une solution plus efficace et légère pour l'exécution d'applications isolées que la virtualisation traditionnelle évoquée précédemment.

Actuellement, les conteneurs sont une des solutions les plus plébiscitées pour assurer la reproductibilité dans les articles de recherche utilisant une partie informatique. Les conteneurs encapsulent une application et ses dépendances dans une unité autonome, légère et portable. Ces conteneurs fournissent un environnement isolé pour exécuter des applications sur différents systèmes d'exploitation et environnements informatiques, garantissant un comportement cohérent et la reproductibilité des résultats, de manière similaire à ce qui a été obtenu avec les machines virtuelles, mais de manière plus efficace (Boettiger 2015).

La première technologie de conteneurisation étudiée est Docker (Merkel 2014). Plusieurs articles ont plaidé en faveur de Docker dans le cadre de la recherche reproductible (Boettiger 2015; Nüst *et al.* 2020). Étant donné que Docker était la première technologie étudiée, elle a été comparée à la virtualisation basée sur un hyperviseur concernant le partage de l'environnement logiciel. Dans (Felter *et al.* 2015), les auteurs ont travaillé avec Kernel Virtual Machine (KVM) (Kivity *et al.* 2007), qui est un hyperviseur de type 1 souvent étudié. Ils ont exécuté le benchmark Linpack sur Linux natif, Docker et KVM. Une exécution de Linpack passe la majeure partie de son temps à effectuer des opérations mathématiques en virgule flottante. Les performances étaient presque identiques sur Linux et Docker, mais les performances de KVM étaient nettement moins bonnes. Ils ont effectué plusieurs tests différents, pas seulement des opérations arithmétiques en virgule flottante, et ont conclu que Docker égalait ou dépassait les performances de KVM dans tous les cas. Chae *et al.* (2019) ont comparé Docker à KVM, et leurs résultats montrent que Docker est plus rapide que KVM, en constatant que Docker performe mieux que KVM en termes d'utilisation du CPU, du disque dur et de la

RAM. Rad *et al.* (2017) ont également constaté que Docker performe mieux que les machines virtuelles. Une fois de plus, dans (Chung *et al.* 2016), les auteurs ont constaté que Docker génère moins de surcharge que les machines virtuelles. Potdar *et al.* (2020) ont comparé Docker et les machines virtuelles en termes de performances CPU, de débit mémoire, d'E/S disque, de test de charge et de mesure de vitesse d'opération, et ont observé que Docker performe mieux que les machines virtuelles dans tous les tests. Ruiz *et al.* (2015) confirme également que les technologies de virtualisation utilisant des conteneurs performent mieux.

À partir de cet ensemble d'études, nous constatons que les conteneurs comme Docker sont de meilleurs outils à utiliser que les machines virtuelles dans le contexte de la recherche reproductible. Cependant, nous n'avons pas encore répondu à la question de savoir si Docker induit une surcharge de performance par rapport à l'exécution native, et s'il convient au contexte du calcul à hautes performances, c'est donc ce que nous abordons ci-après.

Bien que Docker puisse sembler être une bonne option, il ne convient pas au calcul à hautes performances. La première raison est que Docker nécessite des privilèges administrateur. Au sein des clusters de calcul, aucun utilisateur ne devrait avoir de privilèges administrateur. Pour cette raison, Docker n'est pas utilisable dans un contexte de calcul à hautes performances. Le deuxième point est que nous avons trouvé des mentions de préoccupations en matière de sécurité dans diverses études sur Docker (Jacobsen et Canon 2015; Priedhorsky et Randles 2017; Zhou *et al.* 2022).

D'autres conteneurs ont été développés pour pouvoir être utilisé dans le contexte du calcul à hautes performances : Charliecloud (Priedhorsky et Randles 2017), Shifter (Gerhardt *et al.* 2017), Singularity (Kurtzer *et al.* 2017) (renommé Apptainer), Sarus (Benedicic *et al.* 2019) et Podman

(Gantikow *et al.* 2020). Charliecloud exécute des conteneurs sans nécessiter de droit administrateur ou de « daemon ». Il convertit les images Docker en fichiers tar et les décompresse sur les nœuds HPC. Il est considéré sécurisé et supporte également MPI, une norme pour la communication entre les processus dans les systèmes parallèles. De plus, Charliecloud offre une solution pour les problèmes de compatibilité des bibliothèques logicielles. Il le fait en intégrant des fichiers provenant du système hôte directement dans les images de conteneurs. Cette méthode assure une meilleure compatibilité des applications en conteneur avec l'environnement dans lequel elles sont exécutées. Shifter est un moteur de conteneurs pour le calcul haute performance (HPC) créé par NERSC. Il utilise Docker pour construire des images, les transformant en format « ext4 » puis les compressant en « squashfs » pour le stockage sur des systèmes de fichiers parallèles. Il prend en charge MPI et gère la compatibilité des pilotes GPU en remplaçant le pilote GPU au démarrage du conteneur. Apptainer est un outil populaire dans l'enseignement supérieur et l'industrie pour le HPC. Il fonctionne sans privilèges administrateur et ne requiert pas de processus « daemon », supportant les GPU, MPI, et particulièrement la technologie InfiniBand, essentielle pour l'interconnexion dans les systèmes HPC. Apptainer permet l'exécution portable d'applications via un fichier image unique au format SIF et propose des modèles hybrides pour les applications MPI. Sarus, un autre moteur de conteneurs HPC, s'appuie sur « runc » pour créer des conteneurs conformément à la spécification Open Container Initiative (OCI). Il se compose d'un répertoire de système de fichiers racine et d'un fichier de configuration JSON. Podman exécute des conteneurs sans élévation des privilèges, en utilisant l'espace de noms utilisateur. Il prend en charge le même runtime que Sarus et Docker (runc), ainsi que le

runtime plus rapide crun. Il introduit le concept de pod pour regrouper des conteneurs pour des applications complexes. Toutes ces technologies reposent sur une fonctionnalité de base de Linux appelée Linux Containers (LXC) (Senthil Kumaran 2017).

La recherche a montré qu'aucun des conteneurs (virtualisation au niveau du système d'exploitation), n'induit une surcharge significative (n'affecte pas négativement les performances) ou a au moins des performances proches de celles d'une exécution native. Younge *et al.* (2017) montre que Singularity fonctionne à des performances natives dans le cadre du calcul à hautes performances. Xavier *et al.* (2013) montre que la virtualisation LXC offre des performances natives. Une étude plus complète de 2019 (Torrez *et al.* 2019) a étudié les performances de Shifter, Charliecloud et Singularity. Ils n'ont trouvé aucune différence significative de performances entre les trois environnements et l'exécution native, à l'exception possible d'une légère variation de l'utilisation de la mémoire. Young *et al.* (2018) ont étudié les performances de Singularity et Docker, et les résultats montrent que Singularity n'entraîne aucune perte de performance, mais Docker entraîne une légère dégradation. Hale *et al.* (2017) ainsi que Le et Paz (2017) confirment également que Singularity n'entraîne aucune perte de performance. Abraham *et al.* (2020) étudie davantage le débit d'entrée/sortie sur Docker, Charliecloud, Singularity et Podman sur un système de fichiers Lustre (un système de fichiers distribué utilisé dans des environnements de calcul à hautes performances au-delà de 40 nœuds). Les résultats « montrent une surcharge de temps de démarrage pour Docker et Podman, ainsi qu'une surcharge réseau au démarrage pour Singularity et Charliecloud. Nos évaluations d'E/S montrent qu'avec une parallélisation croissante, Charliecloud génère une surcharge importante sur le Metadata Server (MDS) et le Object storage

server (OSS) de Lustre. De plus, nous constatons que le débit observé des conteneurs sur Lustre est comparable à celui des conteneurs exécutés à partir du stockage local ». Casalicchio et Perciballi (2017) ont constaté une légère surcharge pour Docker, d'environ 5 % à 10 % pour la charge CPU et de 10 % à 30 % de surcharge d'E/S disque. Axés sur des scénarios de calcul à hautes performances, Hu *et al.* (2019) n'ont trouvé aucune surcharge pour Singularity sur les applications parallèles MPI et GPU.

Sur la base de ces connaissances, nous constatons que Apptainer (Singularity) et d'autres technologies de conteneurs, à l'exception de Docker, conviennent au calcul haute performance et à la recherche reproductible. Nous pensons qu'Apptainer est la meilleure solution à utiliser et à apprendre actuellement, car il est l'outil le plus répandu sur les clusters de calcul. Cependant, comme Apptainer a récemment changé (évolution à partir de Singularity), une étude à jour pourrait être utile pour confirmer qu'Apptainer reste parfaitement adapté aux performances natives ou proches de celles-ci.

Un inconvénient des conteneurs est qu'ils sont initialement conçus pour le déploiement, et non pour l'archivage. Bien qu'il soit possible de créer un Dockerfile qui spécifie les étapes exactes pour construire une image de conteneur, il n'y a toujours aucune garantie que l'image résultante sera entièrement reproductible. Cela signifie que même si deux personnes utilisent le même Dockerfile, elles peuvent obtenir des images de conteneur légèrement différentes. Une des raisons de ce manque de reproductibilité est que les images de conteneurs sont des binaires, ce qui signifie qu'elles sont compilées et non facilement lisible ou modifiables. Cela rend difficile la compréhension de toutes les dépendances et configurations à l'intérieur d'une image de conteneur. Si quelqu'un souhaite reproduire les calculs effectués à l'intérieur d'un conteneur, il

devrait recréer l'environnement de conteneur exact. Cependant, en raison de la non-reproductibilité des images de conteneurs, ce processus peut ne pas donner des résultats identiques. Par conséquent, bien que les conteneurs soient efficaces pour le déploiement d'applications, ils peuvent ne pas être la solution idéale pour obtenir une reproductibilité totale dans le contexte de la recherche scientifique. Cependant, dans la pratique, les conteneurs sont les outils les plus utilisés et restent adaptés à la recherche reproductible. Comme mentionné précédemment, Guix pourrait actuellement être la meilleure solution que nous connaissons pour la reproductibilité computationnelle. Des états de l'art complets des conteneurs pour le calcul à hautes performances ont été récemment publiés en 2022 et 2023, respectivement (Zhou *et al.* 2022) et (Keller Tesser et Borin 2023).

6.5. Calcul parallèle

6.5.1. *La reproductibilité des calculs flottants*

L'arithmétique en virgule flottante n'est pas associative. Par conséquent, lorsqu'on traite des opérations désordonnées, on peut perdre la reproductibilité. Une méthode pour résoudre ce problème consiste à mettre en œuvre directement des solutions algorithmiques pour gérer les calculs en virgule flottante. Demmel et Nguyen ont largement contribué à ce sujet. En 2013, Demmel et Nguyen (Demmel et Nguyen 2013a) ont abordé le défi de parvenir à une reproductibilité avec les sommes en virgule flottante, en particulier face à la planification dynamique des processeurs qui peuvent modifier l'ordre d'exécution des opérations et à la non-associativité des opérations en virgule flottante dans les environnements de calcul parallèle. Ils proposent une technique de somme en virgule flottante qui est reproductible indépendamment de l'ordre de la

somme, en utilisant l'algorithme Rump (Rump *et al.* 2010), plus efficace que l'utilisation d'une arithmétique avec haute précision. Cette solution fait un compromis entre efficacité et précision. Les performances de cette solution sont améliorées avec l'algorithme OneReduction, toujours de Demmel et Nguyen (Demmel et Nguyen 2014), en utilisant des nombres en virgule flottante indexés et en nécessitant une seule opération de réduction pour réduire le coût de communication sur les plateformes parallèles à mémoire distribuée. Cependant, ces solutions n'améliorent pas la précision. Le résultat calculé, même s'il est reproductible, reste exposé à des problèmes de précision, en particulier lorsque l'on traite un problème où une petite modification des données d'entrée peut entraîner de grandes variations dans le résultat (Chohra *et al.* 2016). Selon Stodden (Stodden *et al.* 2013), l'utilisation du type long-double ou de la somme de Kahan peut augmenter la reproductibilité sans augmenter beaucoup le temps de calcul. En ce qui concerne les algorithmes de somme compensée rapides et précis, les travaux récents de Blanchad *et al.* (2020) et de Lange (2022) sont intéressant. L'arithmétique par intervalles est également une solution envisagée (Revol et Théveny 2014).

Certains outils ont également été développés pour améliorer la reproductibilité de l'arithmétique en virgule flottante. La bibliothèque Intel MKL présente le CNR (Reproductibilité Numérique Conditionnelle) (Rosenquist 2012). Cette fonctionnalité limite l'utilisation des extensions d'ensemble d'instructions pour garantir des résultats numériques reproductible sur différentes architectures. Cependant, cette approche a tendance à considérablement réduire les performances, notamment sur les architectures plus récentes. De plus, elle exige un nombre de threads identique d'une exécution à l'autre pour maintenir des résultats cohérents. Verrou (Févotte et Lathuilière 2016) est un outil construit sur l'outil

Valgrind qui utilise l'arithmétique de Monte Carlo (MCA) pour surveiller la précision des opérations en virgule flottante dans les simulations numériques sans nécessiter de modification du code source ou de recompilation. Conçu pour les applications à petite échelle et les codes industriels complexes, Verrou aide à diagnostiquer les inexactitudes découlant des calculs en virgule flottante, contribuant au processus de vérification et de validation dans des industries telles qu'EDF (Électricité De France), qui reposent sur des simulations numériques pour la sécurité et l'efficacité de leurs centrales nucléaires.

Cependant, pour gérer la recherche reproductible en calcul haute performance, des solutions de niveau supérieur à celles qui travaillent directement sur l'arithmétique en virgule flottante sont préférées. De bons algorithmes en virgule flottante ne peuvent pas résoudre tous les problèmes induits par la parallélisation complexe des simulations stochastiques.

6.5.2. *Enregistrer et rejouer*

Lorsqu'il s'agit de traiter des workflows et des exécutions non déterministes, tels que nous en trouvons en calcul parallèle, il peut être difficile de répéter même nos propres expériences et d'obtenir des résultats identiques bit à bit, bien que cela soit nécessaire pour le débogage. Les outils d'enregistrement et de relecture peuvent être utiles pour répéter ou reproduire un calcul non déterministe, comme suggéré par Chapp *et al.* (2018). Nous avons synthétisé ce dernier travail pour voir comment de tels outils ont été initialement proposés pour permettre le débogage de calculs parallèles sur des clusters. L'exécution dynamique modifiant l'ordre des opérations, avec des bibliothèques à hautes performances comme MPI, qui peut en effet rendre les calculs non déterministes d'une exécution à

l'autre. MPI est une bibliothèque largement utilisée pour gérer les communications et la concurrence sur des machines distribuées.

Tout d'abord, certains outils d'enregistrement et de relecture sont conçus pour une architecture à mémoire partagée, où tous les processeurs travaillent sur la même mémoire. L'outil ODR (Altekar et Stoica 2009) est un outil logiciel conçu pour les programmes multiprocesseurs qui a pour objectif d'obtenir une relecture déterministe du résultat en enregistrant uniquement une partie des données d'exécution et en employant une stratégie de recherche lors de la relecture pour converger vers une exécution reproduisant les sorties originales du programme. En évitant le besoin d'une reproduction bit à bit de l'ensemble de l'exécution et en contournant les problèmes de concurrences des données, ODR peut reproduire les comportements dans des applications multiprocesseurs telles qu'Apache et MySQL avec un facteur de surcoût moyen d'enregistrement de 1,6 fois. PRES (Park *et al.* 2009) utilise l'approche « enregistrement et relecture » pour le débogage d'exécution parallèle multiprocesseurs en développant des « schémas d'exécution » pendant l'enregistrement. Ils explorent de manière itérative les chemins d'exécution potentiels qui correspondent à ces schémas pour reproduire étroitement une exécution sur laquelle nous avons eu un bogue. Selon eux, la plupart des bogues sont fidèlement répliqués en moins de 10 essais de relecture. Respec, proposé par Lee *et al.* (Lee *et al.* 2010) est un système de relecture déterministe qui exécute simultanément la relecture et l'exécution, vérifiant périodiquement les divergences entre les deux et maintenant des points de contrôle des états mutuellement acceptés. Les évaluations révèlent un surcoût de 18% pour les programmes à deux threads et de 55% pour les applications à quatre threads lorsqu'ils sont testés sur les benchmarks PARSEC et SPLASH-2. ScalaMemTrace (Budanur *et al.* 2011) fournit une

représentation compressée de la trace mémoire, identifiant les motifs comportementaux récurrents à travers la hiérarchie de mémoire. Évalué avec des charges de travail axées sur le calcul haute performance, ScalaMemTrace maintient une taille de trace presque constante jusqu'à 64 threads. La fidélité de relecture de cet outil était reconnue comme ayant des limitations, mais atteignant une précision de 91% sur le benchmark AMG. PinPlay d'Intel (Patil *et al.* 2010), propose des capacités d'enregistrement et de relecture adaptables, telles que la relecture de sous-groupes, et assure la compatibilité avec d'autres outils associés à Pin, visant la polyvalence. Lorsqu'il est évalué sur des charges de travail de calcul à hautes performances, PinPlay enregistre un surcoût de temps d'exécution très significatif, étant de 10 à 18 fois durant la phase de « replay » de plusieurs applications MPI. Light (Liu *et al.* 2015) de Liu *et al.* détermine et journalise de manière succincte les traces de données essentielles pour une relecture précise, en se concentrant spécifiquement sur la dépendance de flux des accès à la mémoire partagée. Les évaluations sur un ensemble diversifié de benchmarks indiquent que Light présente un surcoût de journalisation de 44% et un surcoût d'espace de 10% par rapport aux techniques traditionnelles. PORRidge (Utterback *et al.* 2017), est conçu pour les programmes Cilk Plus, qui sont une extension des langages de programmation C et C++ permettant de gérer le parallélisme des données et des tâches. Cette approche utilise une technique d'enregistrement et de relecture qui ne dépend pas du type de processeur. Elle se distingue en se concentrant sur les enregistrements associés à chaque verrouillage, plutôt qu'à chaque fil d'exécution (thread). Cette méthode influence le planificateur de Cilk pour qu'il se conforme aux ordres d'accès préalablement enregistrés, en utilisant une représentation améliorée du graphe acyclique dirigé. Des tests réalisés sur une gamme variée de

benchmarks ont montré que le surcoût lié à l'enregistrement variait, atteignant parfois 3,39 fois le niveau normal, avec une moyenne de 1,62 fois. Rerun (Hower et Hill 2008) est un système conçu pour les architectures multiprocesseurs. Il introduit un mécanisme de relecture déterministe, c'est-à-dire qu'il permet de reproduire de manière fiable et précise le déroulement d'un programme. Au lieu de se concentrer sur l'enregistrement de chaque conflit de mémoire individuel, Rerun tire parti des « épisodes atomiques ». Ces épisodes sont des séquences d'instructions exécutées par un seul thread qui ne génèrent pas de conflits avec d'autres threads. L'avantage majeur de cette approche est qu'elle nécessite une quantité de mémoire relativement faible pour chaque cœur du processeur. QuickRec (Pokam *et al.*, 2013) est une extension pour l'architecture Intel, offrant des capacités d'enregistrement et de relecture assistées par matériel pour les programmes multithreads sur des systèmes multicœurs. En utilisant la plateforme d'émulation QuickIA (Chitlur *et al.* 2012) et un noyau Linux modifié, QuickRec bénéficie d'une génération minimale de journaux mémoire et de surcoût de performances, bien que sa pile logicielle introduise un surcoût moyen d'environ 13%. Samsara (Ren *et al.* 2015) capitalise sur le « hardware-assisted virtualization » (HAV), en utilisant une méthode de relecture déterministe dans les systèmes multiprocesseurs sans nécessiter de modifications matérielles. En utilisant un schéma d'enregistrement basé sur les blocs qui évite toutes les détections d'accès mémoire (une source principale de surcoût dans les méthodes précédentes), les auteurs soutiennent que Samsara réduit considérablement la taille du fichier journal à 1/70ème et réduit le surcoût d'enregistrement à une moyenne de 2,3 fois par rapport aux solutions logicielles classiques. Plus récemment en 2017, Castor (Mashtizadeh *et al.* 2017) offre une solution par défaut d'enregistrement et de relecture pour

les applications multicœurs, mettant l'accent sur des surcoûts minimes et prévisibles. Notamment, bien que Castor atteigne un faible surcoût d'enregistrement pour la majorité des benchmarks PARSEC, le benchmark Radiosity voit des surcoûts atteignant 25% lors d'une exécution à 10 threads en raison des impacts sur le cache. Castor peut fonctionner avec les langages C, C++ et Go.

D'autres outils sont conçus pour une architecture à mémoire distribuée (contrairement à la mémoire partagée vu précédemment), où chaque processeur dispose de sa propre mémoire privée, et les processeurs doivent communiquer en s'envoyant des messages les uns aux autres, le plus souvent en utilisant la bibliothèque MPI. Scala-H-Trace (Wu *et al.* 2011) utilise une compression de trace agressive, capturant les variations des paramètres de communication et d'E/S via des histogrammes probabilistes, garantissant des tailles de fichier de log presque constantes même avec des contextes variables. Scala-H-Trace rejoue de manière déterministe ces traces probabilistes sans blocages, ressemblant étroitement aux applications originales, avec des temps de relecture étant de 12% à 15% supérieurs aux temps d'exécution originaux. De manière similaire à Scala-H-Trace, Scalatrace II (Wu et Mueller 2013) utilise des techniques de compression de trace, efficaces pour les applications HPC aux comportements irréguliers, en employant un schéma de codage de bas niveau qui améliore l'adaptabilité et l'interprétation des données. Guermouche *et al.* (2011) introduisent un protocole de points de contrôle non coordonnés adapté aux applications HPC avec MPI, qui ne journalise que les messages sélectionnés et n'exige pas un redémarrage complet du processus après une défaillance. Meneses *et al.* (2010) utilisent une technique conçue pour atténuer le surcoût mémoire inhérent à la journalisation des messages en mémoire en regroupant les processeurs en

groupe, où seuls les messages inter-groupes nécessitent une journalisation. Xue *et al.* (2009) ont introduit MPIWiz, basé sur une méthode qu'ils ont nommée relecture reproductible de sous-groupe, une technique hybride de relecture déterministe pour les applications MPI qui équilibre les avantages et les limitations de la relecture des données et de la relecture des instructions dans l'ordre original. En utilisant MPIWiz, le système capture le contenu des messages entre les groupes de processus et seulement les ordres des messages au sein d'un groupe, montrant une augmentation de 27% du temps d'exécution lors de l'enregistrement et permettant une relecture en seulement 53% du temps d'exécution de base de l'application. Gioachin *et al.* (2010) ont présenté un mécanisme hybride de relecture en trois étapes où les premières passes adoptent une relecture d'ordre minimaliste, passant ensuite à des relectures de données plus intensives mais limitées à un nombre progressivement réduit de processus, facilitant ainsi la traçabilité ciblée des erreurs. Rex (Perianayagam *et al.* 2010), introduit par Perianayagam *et al.*, est un ensemble d'outils conçus pour enregistrer, archiver et rejouer de manière exhaustive des expériences logicielles, garantissant la fidélité de la reproduction malgré d'éventuels changements dans les ensembles de données externes, l'indisponibilité du logiciel original ou les paramètres d'entrée non documentés. Il ajoute un surcoût minimal en termes de temps d'exécution, d'environ 1,6 %, et un surcoût en termes d'espace d'archivage variant entre 5 et 7 Go. Un état de l'art complet des outils d'enregistrement et de relecture se trouve dans (Chapp *et al.* 2018).

Comme nous pouvons le voir, de nombreux outils ont été proposés assez récemment car le problème du non-déterminisme est omniprésent en calcul à hautes performances, et il s'intensifie avec l'avancement des technologies, notamment la diminution de la taille de gravure des

processeurs et l'augmentation des puissances de calcul. Beaucoup de ces outils ont un effet négatif non négligeable sur les performances, ce qui réduit leur utilisation. Cependant, ce sont les dernières solutions possibles pour la mise au point des programmes. Il faut se rappeler sans cesse que garantir la répétabilité et la reproductibilité sont des aspects essentiels pour le débogage. En effet, bon nombre de ces outils ont été créés à des fins de débogage, et non pas pour la reproductibilité. Néanmoins, en calcul à hautes performances, il est intéressant de savoir que c'est une solution existante qui permet d'effectuer des expériences reproductibles.

6.6. Gestion des erreurs silencieuses

Les erreurs silencieuses sont courantes dans les systèmes de calcul à hautes performances, et plus encore à l'échelle des calculs « exaflopiques ». Ce phénomène a été étudié, et plusieurs solutions existent.

Au niveau de la mémoire, la technologie de correction d'erreur (ECC memory - Error Correction Code) a été conçue pour protéger les mémoires RAM. La mémoire ECC est une forme spécialisée de stockage de données informatiques qui utilise un code de correction d'erreur pour identifier et corriger les occurrences de corruption de données sur une certaine quantité de bits dans la mémoire. Elle est utilisée dans des situations où la corruption des données est inacceptable, pour le calcul scientifique mais aussi dans les systèmes de contrôle industriel, les bases de données critiques et les caches mémoire essentiels. La mémoire ECC est conçue pour empêcher les erreurs dues à un seul bit qui pourraient affecter l'intégrité de la mémoire, garantissant que les données lues correspondent aux données initialement écrites, même si un bit a été altéré involontairement. En revanche, la plupart des mémoires non-ECC classiques ne disposent pas de capacités de détection et de correction des

erreurs, seules certaines configurations de mémoires non-ECC comportent une prise en charge de la parité, ce qui permet la détection des erreurs sans correction. Dans l'étude de Baumann (2005), Baumann a travaillé sur l'observation du taux d'erreurs pour une SRAM non protégée par ECC. « Ceci est calculé en utilisant un taux d'erreurs silencieuses (SER) dû à l'exposition aux radiations, ce qui donne un estimé de 50000 FIT (Failure-In-Time : un FIT équivaut à une défaillance sur 1 milliard d'heures de fonctionnement). Ils recommandent donc l'utilisation de l'ECC, qui divise le taux d'erreur par 1000 pour les SRAM. » (Dixit *et al.* 2021). D'autres contre-mesures matérielles pour gérer les erreurs silencieuses incluent la redondance et la parité. La redondance consiste à dupliquer des composants ou des données critiques de manière à ce que, en cas d'erreur sur l'un d'entre eux, le composant ou les données redondants puissent prendre le relais de manière transparente. La parité, une forme plus simple de vérification d'erreur, ajoute un bit supplémentaire à un ensemble de données pour garantir une parité paire ou impaire (la somme des bits étant paire ou impaire). En cas d'erreur, le bit de parité change, ce qui indique un problème.

Des solutions logicielles existent également. Fiala *et al.* (2012) propose une bibliothèque associée à MPI pour gérer les erreurs silencieuses, basée sur la redondance. Dans (Benson *et al.* 2015), les auteurs proposent de gérer les erreurs silencieuses avec un nouveau paradigme pour détecter de telles erreurs au niveau de l'application. Cette méthode consiste à comparer régulièrement les valeurs obtenues par des calculs avec celles issues d'un processus de vérification peu coûteux. En analysant les écarts entre ces deux séries de résultats, des systèmes de détection d'erreurs sont établis. Les chercheurs emploient l'analyse numérique pour déterminer des procédures de vérification adaptées aux problèmes de valeurs initiales

rencontrés dans les équations différentielles ordinaires et les équations aux dérivées partielles. Plus précisément, ils utilisent des techniques telles que les méthodes de Runge-Kutta et les méthodes linéaires à plusieurs étapes pour les équations différentielles ordinaires, ainsi que des approches de différences finies, tant implicites qu'explicites, et pour les équations aux dérivées partielles. Pour illustrer leur méthode, les auteurs se réfèrent à des exemples comme l'équation de la chaleur et les équations de Navier-Stokes. Il est intéressant de noter que, lors de tests incluant des erreurs introduites volontairement, la stratégie proposée a démontré une capacité à identifier presque toutes les erreurs significatives sans causer de baisse notable de la performance de calcul. Hoemmen et Heroux (2011) proposent une version résiliente de la méthode itérative GMRES. Cette version résiliente est conçue pour corriger les erreurs et améliorer la tolérance aux erreurs de la méthode. Cependant, Bronevetsky et de Supinski (2008) ainsi que Casas *et al.* (2012) suggèrent que des études empiriques ont montré que certaines méthodes itératives pourraient être vulnérables aux erreurs. Ces études indiquent que toutes les méthodes itératives ne corrigent pas intrinsèquement les erreurs. Huang et Abraham (1984) introduisent l'utilisation de méthodes de somme de contrôle pour améliorer l'intégrité de la multiplication de matrices, en détectant efficacement les erreurs dans le processus. Du *et al.* (2012) proposent l'application de méthodes de somme de contrôle pour améliorer la robustesse de la décomposition LU (Lower-Upper) à hautes performances, détectant ainsi les erreurs potentielles dans le processus de factorisation LU. Dans (Aupy *et al.* 2013), la solution proposée pour gérer les erreurs de corruption silencieuse des données repose sur la révision des stratégies traditionnelles de sauvegarde et de récupération par retour en arrière. L'accent est spécifiquement mis sur la résolution d'erreurs latentes

qui ne sont pas immédiatement détectées. Ils introduisent deux modèles pour gérer de telles erreurs. Dans le premier modèle, les erreurs sont détectées après un certain délai, suivant une distribution de probabilité souvent représentée sous forme de distribution exponentielle. La solution calcule la période optimale pour la sauvegarde, dans le but de minimiser le temps perdu pendant que les nœuds ne sont pas engagés dans des calculs utiles. Étant donné que seul un nombre limité de points de sauvegarde peut être stocké en mémoire, il existe une possibilité de défaillance irrécupérable en raison de ressources limitées. Dans de tels cas, les auteurs déterminent la période minimale nécessaire pour un niveau acceptable de risque. Dans le deuxième modèle, les erreurs sont détectées grâce à un mécanisme de vérification. Contrairement au premier modèle, il n'y a pas de risque de défaillance irrécupérable car le mécanisme de vérification garantit la détection des erreurs. Cependant, les coûts introduits par ce mécanisme de vérification comptent dans le calcul du temps perdu. L'objectif principal est de trouver une période de sauvegarde optimale qui minimise le temps perdu, en tenant compte du compromis entre la détection précoce des erreurs, le stockage limité des points de sauvegarde et les surcoûts introduits par les mécanismes de vérification. Les auteurs appliquent ces modèles à des scénarios réels et tiennent compte de divers paramètres d'application et d'architecture pour démontrer leur faisabilité et leur efficacité.

Des technologies plus récentes peuvent offrir de nouvelles solutions, telles que l'apprentissage automatique. Dans (Wang *et al.* 2018), ils proposent un détecteur basé sur un réseau de neurones capable de détecter les corruptions silencieuses des données, même plusieurs itérations après leur injection. L'efficacité de ce détecteur proposé est évaluée à l'aide de six applications Flash et de deux mini-applications Mantevo. Les résultats

des expériences montrent que ce détecteur peut identifier avec succès plus de 89 % des corruptions silencieuses des données tout en maintenant un faible taux de faux positifs de moins de 2 %. Plus d'informations sur ce sujet peuvent être trouvées dans un état de l'art sur les techniques de modélisation et d'amélioration de la fiabilité des systèmes informatiques (Mittal et Vetter 2015).

Chapitre 7

Exemples pratiques de reproduction d'articles et recommandations

Pour illustrer les principes théoriques et mettre en évidence les obstacles communs, nous fournissons dans cette section quelques exemples de tentatives de reproduction d'articles scientifiques.

En 2020, Hinsen a tenté de reproduire son propre travail intitulé « Structural flexibility in proteins - impact of the crystal environment » (Hinsen 2020a). Il a rencontré des défis significatifs pour reproduire les résultats de ses recherches informatiques en raison de l'évolution des environnements logiciels et des problèmes de gestion des données. Son travail original utilisait des versions antérieures de Python et des bibliothèques telles que le Molecular Modeling Toolkit (MMTK), ScientificPython et netCDF, qui ne sont plus compatibles avec les mises à jour logicielles récentes comme Python 3 et les dernières versions de NumPy qui ont cessé de supporter les interfaces obsolètes. La transition de Python 2 à Python 3 a été particulièrement problématique en raison de changements majeurs dans l'API C de Python et une définition différente des divisions (en Python 2, « / » est un opérateur de division entière, tandis qu'il est un opérateur de division flottante en Python 3). De plus, des scripts cruciaux pour générer des graphiques et d'autres sorties n'ont pas été publiés ou ont été perdus en raison de défaillances matérielles, notamment concernant le stockage sur des CD-ROM et sur des bandes magnétiques illisibles (pour lesquelles il n'a pas été possible de trouver un lecteur fonctionnel). Ces problèmes ont été exacerbés par des changements dans les formats de données utilisés par des bases de

données essentielles comme la Protein Data Bank, qui peuvent avoir mis à jour ou corrigé leurs entrées, affectant la reproductibilité des analyses basées sur les données originales. Bien que l'auteur ait pu obtenir des résultats assez proches de l'article original, il n'a pas pu reproduire exactement tous les résultats (comme les figures par exemple) en raison de la perte de certains codes comme précité.

En 2023, Legrand et Velho ont tenté de reproduire leur propre travail sur SimGrid (Legrand et Velho 2023). Dans leurs efforts pour répliquer leur précédent papier, ils ont fait face à une série d'obstacles importants, principalement en raison de l'indisponibilité de l'article original, des codes sources et des ensembles de données, exacerbés par les défis posés par les dépendances logicielles obsolètes. L'article original et les codes sources associés n'étaient pas facilement accessibles en raison de problèmes de droits d'auteur et de l'arrêt du logiciel GTNetS, crucial pour les simulations. Pour surmonter ces obstacles, les auteurs ont téléchargé les documents et codes nécessaires sur le dépôt en libre accès HAL, GitHub ou Software Heritage, garantissant ainsi que les chercheurs futurs pourraient accéder librement à ces matériaux. Ils ont également relevé le défi de recréer un environnement logiciel compatible en développant une image Docker qui émulait les anciens systèmes et en utilisant Debian snapshot archive, ce qui permet de fournir la plateforme nécessaire pour exécuter les dépendances obsolètes. Même si Paul Velho a fait des efforts pour documenter le workflow à l'époque, les auteurs concluent que d'autres scientifiques n'auraient probablement pas réussi à reproduire le travail original, principalement en raison de sa complexité et du fait que même si une partie du code source était publiquement disponible, elle restait cachée, et donc peu de scientifiques auraient pu le trouver. Les auteurs ont minutieusement documenté leur processus de reproduction,

incluant des instructions détaillées de configuration et des modifications aux scripts et aux workflows de simulation pour garantir la compatibilité avec les systèmes contemporains. Cela met en évidence l'importance d'une documentation approfondie, de la préservation des artefacts numériques et de l'accès libre aux matériaux de recherche pour soutenir la reproductibilité académique.

Un dernier exemple sur l'apprentissage automatique illustre certains problèmes de HPC. Dans (Langezaal *et al.* 2023), les auteurs ont rencontré plusieurs défis dans leur tentative de reproduire l'étude « Label-free explainability for unsupervised models ». Un obstacle important était le temps considérable requis pour l'exécution des expériences, certaines prenant plus de 32 heures sur une machine à faible coût. Ce problème a imposé des exigences substantielles sur les ressources et de la patience. De plus, ils ont rencontré des complications dues à des bugs dans la base de code, qui ont ajouté de la complexité au travail de reproductibilité. Ces bugs ont transformé ce qui serait typiquement des exécutions de ligne de commande simples en une tâche plus laborieuse, compliquant davantage leurs efforts. Lorsqu'on considère des expériences qui consomment beaucoup de temps comme dans le domaine du HPC, un obstacle majeur à la reproductibilité pourrait simplement être le manque de ressources matérielles ou d'allocation de temps de calcul suffisant.

Nos recommandations pour faciliter la recherche reproductible dans le domaine du calcul haute performance incluent les directives suivantes : Choisissez d'utiliser des technologies open source et courantes. Étant donné que ces technologies ne seront pas maintenues indéfiniment, des initiatives telles que l'archive « Debian snapshot » ou Software Heritage s'avèrent très importante. Utilisez des plateformes telles que GitHub ou GitLab pour favoriser le processus de développement par le contrôle de

version. Utilisez des notebooks computationnels, tels que Org-mode ou Jupyter Notebook, pour rationaliser la gestion des données et des workflows de résultats. Il est impératif de maintenir une documentation approfondie de toutes les procédures. Une fois terminé, la plateforme Zenodo devrait être utilisée pour archiver tout le code et les données. Pour permettre à d'autres scientifiques de reproduire vos calculs avec précision, nous conseillons vivement l'utilisation de conteneurs légers, tels que Apptainer ou Guix ; ces outils facilitent la reproduction sans compromettre les performances ; notamment, Guix qui s'intègre également avec Software Heritage et qui aide à créer un environnement spécifique à un moment fixé. Si votre code est hautement spécialisé pour une architecture matérielle spécifique, cette spécificité peut entraver la reproductibilité, surtout si des optimisations agressives et une parallélisation sont employées. Malgré ces défis, de nombreux outils offrent désormais des capacités substantielles pour mener une recherche reproductible. L'obstacle principal reste dans des contextes très spécifiques de calcul haute performance, où un matériel particulier est nécessaire et le code est hautement optimisé. De plus, il existe une réticence générale à reproduire des tâches coûteuses en calcul. Un calcul informatique prenant des semaines voir des mois n’est pas prévu pour être refait, de fait, l'utilisation des outils mentionnés ci-dessus renforce la confiance dans le code et les résultats, réduisant ainsi le besoin de reproduire de tels calculs couteux en énergie et en temps. En fin de compte, il ne faut pas oublier qu'une partie importante des problèmes de reproductibilité survient initialement lorsque les auteurs échouent à partager les artefacts.

Chapitre 8

Des problèmes ouverts concernant la recherche reproductible en calcul haute performance

8.1. Portabilité des algorithmes, en particulier des générateurs de nombres pseudo-aléatoires

Vérifier la justesse des algorithmes en calcul à hautes performances est une tâche difficile. Un exemple frappant est l'incohérence observée lorsque l'on implémente un générateur de nombres pseudo-aléatoires dans différents langages de programmation ou technologies. Parmi les meilleurs PRNGs, Philox mentionné précédemment est un exemple où il faut être prudent sur la portabilité, MLFG est un autre exemple où nous avons rencontré des problèmes de portabilité. Lorsqu'un PRNG est initialisé avec le même état initial, nous nous attendons à obtenir une séquence identique de nombres quels que soient les différents environnements informatiques et bibliothèques (comme Numpy ou TensorFlow). Cependant, des variations apparaissent, jetant le doute sur la portabilité et la consistance de l'algorithme. Cela remet également en question la fiabilité des résultats scientifiques obtenus à partir de tels algorithmes. Les fondements de la science informatique déterministe reposent sur la prévisibilité et la fiabilité des algorithmes. Les disparités observées dans les sorties des PRNGs soulignent un problème plus large : sans mise en œuvre normalisée et vérification rigoureuse entre les plateformes, comment pouvons-nous garantir l'intégrité de nos résultats algorithmiques ? Cela appelle à un effort concerté pour développer et respecter la normalisation dans les implémentations de PRNG. De plus, cela nécessite la création de

logiciel de vérification robustes pouvant garantir la fidélité et la portabilité algorithmiques dans divers environnements informatiques, ce qui est essentiel pour l'avancement de la science.

8.2. Reproductibilité des données dans le cadre du « Big data »

La prolifération des données volumineuses dans le monde du calcul à hautes performances introduit des défis variés en matière de reproductibilité des données. À mesure que les ensembles de données deviennent de plus en plus volumineux, les systèmes de gestion des versions de données conventionnels peinent à suivre le rythme, ce qui entraîne d'importants obstacles à la maintenance d'un environnement de données consistant et reproductible. Il s'agit non seulement d'un problème technique, mais aussi méthodologique, où la nécessité de gérer de manière évolutive et efficace ces ensembles de données volumineux est primordiale. De plus, avec l'accent de plus en plus mis sur la science basée sur les données, il est impératif d'établir des pratiques robustes de partage de données adaptées à l'échelle et à la complexité immenses des écosystèmes de données volumineuses du HPC. Ces pratiques doivent non seulement faciliter le partage, mais aussi garantir l'intégrité et la reproductibilité des données. La mise à jour de bases de données volumineuses peut entraîner une perte de reproductibilité, et il est difficile de gérer l'archivage de toutes les versions d'une telle quantité de données. Aborder ces défis est essentiel pour permettre la découverte scientifique collaborative et maintenir la crédibilité de la recherche computationnelle.

8.3. Reproductibilité des temps de calculs et optimisations

La quête de réalisation de calcul toujours plus rapide dans le monde du calcul haute performance conduit souvent à un compromis avec la

reproductibilité, en particulier lorsqu'il s'agit de systèmes parallèles et distribués. Les techniques d'optimisation qui améliorent les performances, telles que l'adaptation du code à des architectures spécifiques ou l'exploitation du parallélisme, peuvent rendre les résultats moins reproductibles. Cette dichotomie pose un problème ouvert critique : comment pouvons-nous équilibrer la recherche de performances maximales avec l'assurance de résultats consistant ? C'est un acte délicat d'équilibrage qui nécessite une compréhension plus approfondie de l'interaction entre l'architecture du système, les stratégies d'optimisation et la nature des tâches. Un exemple déjà donné dans la section précédente est, par exemple, l'utilisation de la fonctionnalité « Fused Multiply-Add » (FMA) ou des « Advanced vector extensions » (AVX) ou encore des unités de calcul tensoriel, qui peuvent entraîner une perte de reproductibilité en HPC. Sur les clusters et les supercalculateurs, cela suppose par exemple de désactiver les instructions AVX et/ou FMA. Cette dernière correspond à une opération en virgule flottante de multiplication et d'addition réalisée en une seule étape horloge du CPU et avec un seul arrondi, et le changement dans l'ordre des opérations en virgule flottante et dans la précision entraîne un manque de reproductibilité si on compare cette exécution avec l'exécution successive des deux opérations de multiplication puis d'addition. La recherche dans ce domaine est vitale pour développer de nouvelles méthodes qui peuvent garantir des résultats reproductibles sans sacrifier trop les performances pour lesquelles les systèmes HPC sont utilisés.

8.4. Education, collaboration et intégration

L'intégration de la reproductibilité au cœur même de la méthode scientifique est un défi qui n'entre pas dans la case des solutions

techniques, touchant aux aspects éducatifs et collaboratifs. Il est impératif d'inculquer les principes de la recherche reproductible au sein des programmes éducatifs afin d'éduquer une nouvelle génération de scientifiques qui intègrent naturellement ces pratiques dans leur travail. Au-delà du monde universitaire, promouvoir des normes de collaboration est essentiel pour développer et maintenir des workflows reproductibles. Cela concerne non seulement le chercheur individuel, mais l'ensemble de l'écosystème scientifique, y compris les éditeurs de journaux scientifiques, les financeurs et les institutions. L'objectif est de créer une culture scientifique où la reproductibilité ne soit pas une réflexion après coup, mais un composant fondamental du processus scientifique, de l'acquisition des données à l'analyse.

8.5. Reproductibilité avec les nouveaux paradigmes de calcul

Les modèles de calculs émergents, notamment l'ordinateur quantique, posent des défis sans précédent en matière de reproductibilité. L'ordinateur quantique, par exemple, fonctionne selon un ensemble de principes différents de l'informatique classique, ce qui entraîne de nouveaux types d'erreurs et d'incertitudes. Le domaine en est encore à ses débuts, de nombreuses options techniques étant explorées pour créer des qubits fiables pour les circuits quantiques. Nous avons souvent besoin d'un millier de qubits excellents pour modéliser un seul qubit parfait, et les expériences qui fonctionnent correctement sur des simulateurs quantiques sont souvent non reproductibles sur de véritables machines quantiques (Hill *et al.* 2023). La nature probabiliste de l'informatique quantique rend de fait la répétabilité insaisissable et la reproductibilité encore plus essentielle. Les outils et pratiques actuels ne sont pas adaptés pour faire face à ces nouveaux problèmes, ce qui nécessite un effort de recherche

pour vérifier les résultats des calculs quantiques bruités et comprendre comment surmonter les obstacles à la reproductibilité dans ce paradigme de calcul disruptif.

Conclusion

Atteindre la reproductibilité en calcul à hautes performances présente des défis. Certains outils choisissent de privilégier la facilité d'utilisation au détriment de l'optimisation des performances, comme on peut l'observer avec les principes FAIR conçus pour des disciplines telles que la biologie. Pour d'autres la performance est recherchée prioritairement. Ainsi, il est intéressant de faire la distinction entre la recherche reproductible en général et la reproductibilité dans le contexte du calcul à hautes performances.

À l'avant-garde de l'exploration scientifique, le calcul à hautes performances incarne l'innovation technologique, mais se débat avec le principe fondamental de la reproductibilité. Cet article s'est lancé dans un voyage à travers le paysage complexe de la reproductibilité dans le monde du calcul informatique et plus précisément celui du calcul à hautes performances. Nous avons examiné la crise de la reproductibilité qui sévit non seulement dans ce domaine, mais aussi dans de nombreux autres domaines scientifiques. Nous avons souligné le rôle essentiel de l'ingénierie logicielle, de la gestion des différentes versions des codes, de la gestion des workflows et de la culture scientifique qui influence la reproductibilité, et avons proposé des solutions robustes telles que la programmation lettrée, la gestion avancée des workflows et des technologies de conteneurisation telles que Guix et Apptainer.

En ce qui concerne la recherche reproductible dans son ensemble, à notre avis, un problème concerne la culture scientifique des chercheurs informaticiens. Les agences de financement devraient mettre l'accent sur la promotion et l'incitation à la recherche reproductible, encourageant les

chercheurs à utiliser les outils existants. Avec des efforts raisonnables, il pourrait devenir courant d'accompagner les publications scientifiques de documents connexes (les artéfacts). Comme l'informatique est un domaine de recherche relativement jeune, de nombreux praticiens manquent de formation en épistémologie. Cela se manifeste souvent lorsque les informaticiens utilisent à tort le terme de « méthodologie » au lieu de « méthode » par exemple. Cependant, avec la poursuite de la promotion de la recherche reproductible, la pratique bien établie de tenir des cahiers de laboratoire en biologie commence à trouver son équivalent via les « notebooks » de type Jupiter qui commencent à se répandre en informatique.

Dans le cas de la reproductibilité en calcul haute performance, la situation est plus compliquée. Le calcul exaflopique est désormais une réalité. Les super calculateurs deviennent beaucoup plus denses et, même s'ils affichent les meilleures performances jamais obtenues, ils ont un temps de fonctionnement très limité, souvent en raison d'erreurs silencieuses. Nous avons constamment besoin de puissance de calcul accrue pour mener des explorations plus poussées avec des programmes complexes. Par exemple, un plan complet d'expériences avec toutes les combinaisons des facteurs possibles (paramètres d'entrée) reste bien souvent inabordable. Par conséquent, une augmentation de de la capacité d'échantillonnage de l'espace de toutes les expériences possibles est toujours la bienvenue. Cette approche nécessite toujours plus de puissance de calcul. Même si l'objectif principal est d'améliorer la vitesse de calcul, cela nécessite le déploiement d'architectures parallèles étendues et d'optimisations, et nous avons découvert au cours de la dernière décennie que cela se fait trop souvent au détriment de la reproductibilité. Les calculs couteux sont des calculs que nous ne voulons pas avoir besoin de

reproduire (Hinsen 2021). Cependant, dans ce cas, nous devons pouvoir avoir confiance dans les résultats obtenus, cela peut supposer l'archivage de données conséquentes comme dans (Boyer *et al.* 2022a; Boyer *et al.* 2022b). Atteindre la reproductibilité en calcul haute performance passe souvent par une perte de performances lorsque l'on accepte de désactiver certaines optimisations qui font perdre la répétabilité dans un contexte de calcul avec des nombres réels flottants. Nous pensons qu'il restera toujours des problèmes en matière de reproductibilité de simulations parallèles massivement optimisées. Pour des calculs parallèles plus modestes, des outils tels que Guix ou Apptainer (Singularity) se sont révélés efficaces avec des performances proches des performances natives. Pour l'instant, que vous travailliez sur une plateforme à mémoire partagée ou à mémoire distribuée, si vous optimisez fortement des calculs parallèles massifs, vous avez de grandes chances de rencontrer un manque de reproductibilité numérique.

Des défis émergents proviennent également des domaines de l'intelligence artificielle et du Big Data. Une part importante des ressources informatiques mondiales est actuellement consacrée à la formation de modèles d'IA et au stockage de vastes ensembles de données. Obtenir une reproductibilité en termes de résultats numériques, mais aussi en termes de performances, peut être difficile mais essentiel pour garantir l'explicabilité en intelligence artificielle. De plus, nous devons également répondre à l'impératif de réduire notre consommation énergétique, et l'on peut aussi travailler sur des algorithmes optimisés pour être plus économes en énergie.

Enfin, une nouvelle forme de calcul haute performance est disponible grâce à l'informatique quantique. Cette technologie est probablement celle qui pose le plus de problèmes ouverts et stimulants. Nous ne disposons

pas actuellement d'un ordinateur quantique général avec des qubits parfaits. De nombreuses options techniques sont testées pour fournir d'excellents qubits, mais ils ne sont pas parfaits, s'il est déjà possible de travailler sérieusement, il reste difficile de concevoir des circuits fiables en 2024. Bien que les simulateurs quantiques fonctionnent correctement, l'utilisation de machines quantiques se heurte encore à de nombreux défis pour obtenir une reproductibilité statistique.

En conclusion, nous voulons mettre en avant le fait que les ordinateurs servent d'outils fondamentaux, non seulement pour les informaticiens, mais aussi pour de nombreuses disciplines scientifiques et que de ce fait, toutes les disciplines sont concernées. Tout comme l'approche précise et systématique de la métrologie appliquée aux instruments des biologistes et des physiciens, les scientifiques doivent reconnaître les imperfections et les incertitudes inhérentes aux outils informatiques. Nous recommandons fortement le MOOC sur la Recherche reproductible qui aborde les principes méthodologiques et l'utilisation d'outils pour une science transparente. Enfin, il nous semble impératif que les principes de la méthode scientifique, y compris le principe de la reproductibilité, soient intégrés dans les programmes d'enseignement en informatique.

Références bibliographiques

Abbasi, K. (2020). *Covid-19: politicisation,"corruption," and suppression of science* British Medical Journal Publishing Group.

Abraham, S., Paul, A.K., Khan, R.I.S., Butt, A.R. (2020). On the use of containers in high performance computing environments. Dans *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 284–293.

Acharya, A., Fanguède, J., Paolino, M., Raho, D. (2018). A performance benchmarking analysis of hypervisors containers and unikernels on ARMv8 and x86 CPUs. Dans *2018 European Conference on Networks and Communications (EuCNC)*. IEEE, 282–9.

Allaire, J. (2012). RStudio: integrated development environment for R. *Boston, MA*, 770(394), 165–171.

Altekar, G., Stoica, I. (2009). ODR: Output-deterministic replay for multicore debugging. Dans *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 193–206.

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., Mock, S. (2004). Kepler: an extensible system for design and execution of scientific workflows. Dans *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.* IEEE, 423–424.

Amstutz, P., Crusoe, M.R., Tijanić, N., Chapman, B., Chilton, J., Heuer, M., Kartashov, A., Leehr, D., Ménager, H., Nedeljkovich, M. (2016). *Common workflow language, v1. 0* eScholarship, University of California.

Amstutz, P., Mikheev, M., Crusoe, M.R., Tijanić, N., Lampa, S. (2024). Existing Workflow systems. , 25 January 2024. [En ligne]. Disponible à l'adresse : https://s.apache.org/existing-workflow-systems [Consulté le 19 Mars 2024].

Antunes, B., Hill, D.R.C. (2023). Identifying quality mersenne twister streams for parallel stochastic simulations. Dans *2023 Winter Simulation Conference (WSC)*. IEEE, 2801-2812.

Aupy, G., Benoit, A., Hérault, T., Robert, Y., Vivien, F., Zaidouni, D. (2013). On the combination of silent error detection and checkpointing. Dans *2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing*. IEEE, 11–20.

Baier, T., Neuwirth, E. (2007). Excel:: Com. *Computational statistics*, 22(1), 91–108.

Bajpai, V., Kühlewind, M., Ott, J., Schönwälder, J., Sperotto, A., Trammell, B. (2017). Challenges with reproducibility. Dans *Proceedings of the Reproducibility Workshop*. 1–4.

Bajpai, V., Bonaventure, O., Claffy, K., Karrenberg, D. (2019). Encouraging reproducibility in scientific research of the internet. *Dagstuhl reports*, 8(10).

Bajpai, V., Brunstrom, A., Feldmann, A., Kellerer, W., Pras, A., Schulzrinne, H., Smaragdakis, G., Wählisch, M., Wehrle, K. (2019). The Dagstuhl beginners guide to reproducibility for experimental networking research. *ACM SIGCOMM Computer Communication Review*, 49(1), 24–30.

Baker, M. (2016). Reproducibility crisis. *Nature*, 533(26), 353–66.

Barba, L.A. (2018). Terminologies for reproducible research. *arXiv preprint arXiv:1802.03311*, 2018.

Baumann, R.C. (2005). Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and materials reliability*, 5(3), 305–316.

Baumer, B., Cetinkaya-Rundel, M., Bray, A., Loi, L., Horton, N.J. (2014). R Markdown: Integrating a reproducible analysis tool into introductory statistics. *arXiv preprint arXiv:1402.1894*, 2014.

Begley, C.G., Ellis, L.M. (2012). Raise standards for preclinical cancer research. *Nature*, 483(7391), 531–533.

Benedicic, L., Cruz, F.A., Madonna, A., Mariotti, K. (2019). Sarus: Highly scalable docker containers for hpc systems. Dans *High Performance Computing: ISC High Performance 2019 International Workshops, Frankfurt, Germany, June 16-20, 2019, Revised Selected Papers 34*. Springer, 46–60.

Benson, A.R., Schmit, S., Schreiber, R. (2015). Silent error detection in numerical time-stepping schemes. *The International Journal of High Performance Computing Applications*, 29(4), 403–421.

Beserra, D., Oliveira, F., Araujo, J., Fernandes, F., Araújo, A., Endo, P., Maciel, P., Moreno, E.D. (2015). Performance evaluation of hypervisors for hpc applications. Dans *2015 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 846–851.

Blackman, D., Vigna, S. (2021). Scrambled linear pseudorandom number generators. *ACM Transactions on Mathematical Software (TOMS)*, 47(4), 1–32.

Blanchard, P., Higham, N.J., Mary, T. (2020). A class of fast and accurate summation algorithms. *SIAM journal on scientific computing*, 42(3), A1541–A1557.

Boettiger, C. (2015). An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1), 71–79.

Booch, G., Jacobson, I., Rumbaugh, J. (1996). The unified modeling language. *Unix Review*, 14(13), 5.

Borthakur, D. (2007). The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 11(2007), 21.

Boyer, A.F. (2022). Contributions to Computing needs in High Energy Physics Offline Activities: Towards an efficient exploitation of heterogeneous, distributed and shared Computing Resources. Thèse de doctorat, CERN.

Boyer, A.F., Haen, C., Stagni, F., Hill, D.R.C. (2022a). Dirac site director: improving pilot-job provisioning on grid resources. *Future Generation Computer Systems*, 133, 23–38.

Boyer, A.F., Haen, C., Stagni, F., Hill, D.R.C. (2022b). Pilot-job provisioning on grid resources: Collecting analysis and performance evaluation data. *Data in Brief*, 42, 108104.

Bronevetsky, G., de Supinski, B. (2008). Soft error vulnerability of iterative linear algebra methods. Dans *Proceedings of the 22nd annual international conference on Supercomputing*. 155–164.

Buckheit, J.B., Donoho, D.L. (1995). *Wavelab and reproducible research* Springer.

Budanur, S., Mueller, F., Gamblin, T. (2011). Memory trace compression and replay for spmd systems using extended prsds?. *ACM SIGMETRICS Performance Evaluation Review*, 38(4), 30–36.

Button, K.S., Munafò, M.R. (2017). Powering reproducible research. *Psychological science under scrutiny: Recent challenges and proposed solutions*, 2017, 22–33.

Casalicchio, E., Perciballi, V. (2017). Measuring docker performance: What a mess!!!. Dans *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. 11–16.

Casas, M., de Supinski, B.R., Bronevetsky, G., Schulz, M. (2012). Fault resilience of the algebraic multi-grid solver. Dans *Proceedings of the 26th ACM international conference on Supercomputing*. 91–100.

Chae, M., Lee, H., Lee, K. (2019). A performance comparison of linux containers and virtual machines using Docker and KVM. *Cluster Computing*, 22(Suppl 1), 1765–1775.

Chapp, D., Sato, K., Ahn, D.H., Taufer, M. (2018). Record-and-replay techniques for HPC systems: A survey. *Supercomputing Frontiers and Innovations*, 5(1), 11–30.

Chirigati, F., Rampin, R., Shasha, D., Freire, J. (2016). Reprozip: Computational reproducibility with ease. Dans *Proceedings of the 2016 international conference on management of data*. 2085–2088.

Chitlur, N., Srinivasa, G., Hahn, S., Gupta, P.K., Reddy, D., Koufaty, D., Brett, P., Prabhakaran, A., Zhao, L., Ijih, N. (2012). QuickIA: Exploring heterogeneous architectures on real prototypes. Dans *IEEE International Symposium on High-Performance Comp Architecture*. IEEE, 1–8.

Chohra, C., Langlois, P., Parello, D. (2016). Reproducible, accurately rounded and efficient BLAS. Dans *European Conference on Parallel Processing*. Springer, 609–620.

Chung, M.T., Quang-Hung, N., Nguyen, M.-T., Thoai, N. (2016). Using docker in high performance computing applications. Dans *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*. IEEE, 52–57.

Claerbout, J.F., Karrenbach, M. (1992). Electronic documents give reproducible research a new meaning. Dans *SEG technical program expanded abstracts 1992*. Society of Exploration Geophysicists, 601–604.

Clayson, P.E., Carbine, K.A., Baldwin, S.A., Larson, M.J. (2019). Methodological reporting behavior, sample sizes, and statistical power in studies of event-related potentials: Barriers to reproducibility and replicability. *Psychophysiology*, 56(11), e13437.

Cohen-Boulakia, S., Belhajjame, K., Collin, O., Chopard, J., Froidevaux, C., Gaignard, A., Hinsen, K., Larmande, P., Le Bras, Y., Lemoine, F. (2017). Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities. *Future Generation Computer Systems*, 75, 284–298.

Collaboration, O.S. (2012). An open, large-scale, collaborative effort to estimate the reproducibility of psychological science. *Perspectives on Psychological Science*, 7(6), 657–660.

Collberg, C., Proebsting, T., Warren, A.M. (2015). Repeatability and benefaction in computer systems research. *University of Arizona TR*, 14(4).

Collberg, C., Proebsting, T.A. (2016). Repeatability in computer systems research. *Communications of the ACM*, 59(3), 62–69.

Collins, F.S., Tabak, L.A. (2014). Policy: NIH plans to enhance reproducibility. *Nature*, 505(7485), 612–613.

Courtès, L., Wurmus, R. (2015). Reproducible and user-controlled software environments in HPC with Guix. Dans *Euro-Par 2015: Parallel Processing Workshops: Euro-Par 2015 International Workshops, Vienna, Austria, August 24-25, 2015, Revised Selected Papers 21*. Springer, 579–591.

Dasgupta, S., Humble, T.S. (2022). Characterizing the reproducibility of noisy quantum circuits. *Entropy*, 24(2), 244.

Dasgupta, S., Humble, T.S. (2021a). Reproducibility in quantum computing. Dans *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 458–461.

Dasgupta, S., Humble, T.S. (2021b). Stability of noisy quantum computing devices. *arXiv preprint arXiv:2105.09472*, 2021.

Davison, A. (2012). Automated capture of experiment context for easier reproducibility in computational research. *Computing in Science & Engineering*, 14(4), 48–56.

Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M.-H., Vahi, K., Livny, M. (2004). Pegasus: Mapping scientific workflows onto the grid. Dans *Grid Computing: Second European*

AcrossGrids Conference, AxGrids 2004, Nicosia, Cyprus, January 28-30, 2004. Revised Papers. Springer, 11–20.

Delescluse, M., Franconville, R., Joucla, S., Lieury, T., Pouzat, C. (2012). Making neurophysiological data analysis reproducible: Why and how?. *Journal of Physiology-Paris*, 106(3–4), 159–170.

Demmel, J., Nguyen, H.D. (2013a). Fast reproducible floating-point summation. Dans *2013 IEEE 21st Symposium on Computer Arithmetic*. IEEE, 163–172.

Demmel, J., Nguyen, H.D. (2013b). Numerical reproducibility and accuracy at exascale. Dans *2013 IEEE 21st Symposium on Computer Arithmetic*. IEEE, 235–237.

Demmel, J., Nguyen, H.D. (2014). Parallel reproducible summation. *IEEE Transactions on Computers*, 64(7), 2060–2070.

Desquilbet, L., Granger, S., Hejblum, B., Legrand, A., Pernot, P., Rougier, N.P., de Castro Guerra, E., Courbin-Coulaud, M., Duvaux, L., Gravier, P. (2019). Vers une recherche reproductible Loic Desquilbet, Sabrina Granger, Boris Hejblum. 2019.

Desvillechabrol, D., Legendre, R., Rioualen, C., Bouchier, C., Van Helden, J., Kennedy, S., Cokelaer, T. (2018). Sequanix: a dynamic graphical interface for Snakemake workflows. *Bioinformatics*, 34(11), 1934–1936.

Di Cosmo, R., Zacchiroli, S. (2017). Software heritage: Why and how to preserve software source code. Dans *iPRES 2017-14th International Conference on Digital Preservation*. 1–10.

Di Tommaso, P., Chatzou, M., Baraja, P.P., Notredame, C. (2014). A novel tool for highly scalable computational pipelines. *figshare*, 2014.

Dixit, H.D., Pendharkar, S., Beadon, M., Mason, C., Chakravarthy, T., Muthiah, B., Sankar, S. (2021). Silent data corruptions at scale. *arXiv preprint arXiv:2102.11245*, 2021.

Dolstra, E., De Jonge, M., Visser, E. (2004). Nix: A Safe and Policy-Free System for Software Deployment. Dans *LISA*. 79–92.

Đorđević, B., Timčenko, V., Pavlović, O., Davidović, N. (2021). Performance comparison of native host and hyper-based virtualization VirtualBox. Dans *2021 20th International Symposium Infoteh-Jahorina (Infoteh)*. IEEE, 1–4.

Drummond, C. (2019). Is the drive for reproducible science having a detrimental effect on what is published?. *Learned Publishing*, 32(1), 63–69.

Drummond, C. (2009). Replicability is not reproducibility: nor is it good science. Dans *Proceedings of the Evaluation Methods for Machine Learning Workshop at the 26th ICML*. National Research Council of Canada Montreal, Canada.

Drummond, C. (2018). Reproducible research: a minority opinion. *Journal of Experimental & Theoretical Artificial Intelligence*, 30(1), 1–11.

Du, P., Luszczek, P., Dongarra, J. (2012). High performance dense linear system solver with resilience to multiple soft errors. *Procedia Computer Science*, 9, 216–225.

Eklund, A., Nichols, T.E., Knutsson, H. (2016). Cluster failure: Why fMRI inferences for spatial extent have inflated false-positive rates. *Proceedings of the national academy of sciences*, 113(28), 7900–7905.

Elliott, J., Mueller, F., Stoyanov, F., Webster, C. (2013). Quantifying the impact of single bit flips on floating point arithmetic. Rapport, North Carolina State University. Dept. of Computer Science

Errington, T.M., Mathur, M., Soderberg, C.K., Denis, A., Perfito, N., Iorns, E., Nosek, B.A. (2021). Investigating the replicability of preclinical cancer biology. *Elife*, 10, e71601.

Fanelli, D. (2010). Do pressures to publish increase scientists' bias? An empirical support from US States Data. *PloS one*, 5(4), e10271.

Fanelli, D. (2018). Is science really facing a reproducibility crisis, and do we need it to?. *Proceedings of the National Academy of Sciences*, 115(11), 2628–2631.

Felter, W., Ferreira, A., Rajamony, R., Rubio, J. (2015). An updated performance comparison of virtual machines and linux containers. Dans *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 171–172.

Ferguson, N.M., Laydon, D., Nedjati-Gilani, G., Imai, N., Ainslie, K., Baguelin, M., Bhatia, S., Boonyasiri, A., Cucunubá, Z., Cuomo-Dannenburg, G. (2020). Impact of non-pharmaceutical interventions (NPIs) to reduce COVID-19 mortality and healthcare demand. Imperial College COVID-19 Response Team. *Imperial College COVID-19 Response Team*, 20(10.25561), 77482.

Févotte, F., Lathuilière, B. (2016). VERROU: Assessing floating-point accuracy without recompiling. , 2016.

Fiala, D., Mueller, F., Engelmann, C., Riesen, R., Ferreira, K., Brightwell, R. (2012). Detection and correction of silent data corruption for large-scale high-performance computing. Dans *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.

Forstmeier, W., Wagenmakers, E.-J., Parker, T.H. (2017). Detecting and avoiding likely false-positive findings–a practical guide. *Biological Reviews*, 92(4), 1941–1968.

Gantikow, H., Walter, S., Reich, C. (2020). Rootless containers with Podman for HPC. Dans *International Conference on High Performance Computing*. Springer, 343–354.

Gelman, A., Stern, H. (2006). The difference between "significant" and "not significant" is not itself statistically significant. *The American Statistician*, 60(4), 328–331.

Gerhardt, L., Bhimji, W., Canon, S., Fasel, M., Jacobsen, D., Mustafa, M., Porter, J., Tsulaia, V. (2017). Shifter: Containers for hpc. Dans *Journal of physics: Conference series*. IOP Publishing, 082021.

Giardine, B., Riemer, C., Hardison, R.C., Burhans, R., Elnitski, L., Shah, P., Zhang, Y., Blankenberg, D., Albert, I., Taylor, J. (2005). Galaxy: a platform for interactive large-scale genome analysis. *Genome research*, 15(10), 1451–1455.

Gilbert, L., Tseng, J., Newman, R., Iqbal, S., Pepper, R., Celebioglu, O., Hsieh, J., Cobban, M. (2005). Performance implications of virtualization and hyper-threading on high energy physics applications in a grid environment. Dans *19th IEEE international parallel and distributed processing symposium*. IEEE, 10 pp.

Gioachin, F., Zheng, G., Kalé, L.V. (2010). Robust non-intrusive record-replay with processor extraction. Dans *Proceedings of the 8th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging*. 9–19.

Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM computing surveys (CSUR)*, 23(1), 5–48.

Goldberg, R.P. (1974). Survey of virtual machine research. *Computer*, 7(6), 34–45.

Goodman, S.N., Fanelli, D., Ioannidis, J.P. (2016). What does research reproducibility mean?. *Science translational medicine*, 8(341), 341ps12-341ps12.

Guermouche, A., Ropars, T., Brunet, E., Snir, M., Cappello, F. (2011). Uncoordinated checkpointing without domino effect for send-deterministic MPI applications. Dans *2011 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 989–1000.

Gundersen, O.E., Kjensmo, S. (2018). State of the art: Reproducibility in artificial intelligence. Dans *Proceedings of the AAAI Conference on Artificial Intelligence*.

Guo, P.J., Engler, D. (2011). {CDE}: Using System Call Interposition to Automatically Create Portable Software Packages. Dans *2011 USENIX Annual Technical Conference (USENIX ATC 11)*.

Halchenko, Y.O., Hanke, M. (2015). Four aspects to make science open "by design" and not as an after-thought. *GigaScience*, 4(1), s13742-015-0072–7.

Hale, J.S., Li, L., Richardson, C.N., Wells, G.N. (2017). Containers for portable, productive, and performant scientific computing. *Computing in Science & Engineering*, 19(6), 40–50.

Hellekalek, P. (1998). Don't trust parallel Monte Carlo! *ACM SIGSIM Simulation Digest*, 28(1), 82–89.

Herndon, T., Ash, M., Pollin, R. (2014). Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff. *Cambridge journal of economics*, 38(2), 257–279.

Hill, D.R.C., Antunes, B., Cluzel, T., Mazel, C. (2023). A few words about quantum computing, epistemology, repeatability and reproducibility. Dans *EU/MEeting 2023*.

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

Hill, D.R.C. (2015). Parallel random numbers, simulation, and reproducible research. *Computing in Science & Engineering*, 17(4), 66–71.

Hill, D.R.C. (2019). Repeatability, reproducibility, computer science and high performance computing: Stochastic simulations can be reproducible too…. Dans *2019 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 322–323.

Hill, D.R.C. (2022). Reproducibility of Simulations and High Performance Computing. Dans *ESM 2022, European Simulation and Modelling Conference*. 5–9.

Hill, D.R.C., Mazel, C., Breton, V. (2017). Reproductibilité et répétabilité numérique. Constats, conseils et bonnes pratiques pour le cas des simulations stochastiques parallèles et distribuées. *Revue des Sciences et Technologies de l'Information-Série TSI: Technique et Science Informatiques*, 36(3–6), 243–272.

Hinsen, K. (2017). Enjeux et défis de la recherche reproductible. Dans *Journée MaDICS-ReproVirtuFlow 2017*.

Hinsen, K. (2021). La reproductibilité des calculs coûteux. Dans *Reproductibilité de la Recherche*. 11–14.

Hinsen, K. (2014). Reproducibility, replicability, and the two layers of computational science. , 2014. [En ligne]. Disponible à l'adresse : https://khinsen.wordpress.com/2014/08/27/reproducibility-replicability-and-the-two-layers-of-computational-science/ [Consulté le 22 Août 2023].

Hinsen, K. (2018). Reusable vs. re-editable code. *Computing in Science and Engineering*, 20(3), 78–83.

Hinsen, K. (2020a). [Rp] Structural flexibility in proteins - impact of the crystal environment. *ReScience C*, 6(1), #5.

Hinsen, K. (2013). Software development for reproducible research. *Computing in Science & Engineering*, 15(04), 60–63.

Hinsen, K. (2020b). Staged computation: The technique you did not know you were using. *Computing in Science & Engineering*, 22(4), 99–103.

Hoemmen, M.F., Heroux, M.A. (2011). Fault-tolerant iterative methods. Rapport, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States)

Hogg, T., Portnov, D. (2000). Quantum optimization. *Information Sciences*, 128(3–4), 181–197.

Holmes, J. (2020). Software engineers calling to retract papers based on this code. , 2020. [En ligne]. Disponible à l'adresse : https://github.com/mrc-ide/covid-sim/issues/165 [Consulté le 1 Janvier 2024].

Howe, B. (2012). Virtual appliances, cloud computing, and reproducible research. *Computing in Science & Engineering*, 14(4), 36–41.

Hower, D.R., Hill, M.D. (2008). Rerun: Exploiting episodes for lightweight memory race recording. *ACM SIGARCH computer architecture news*, 36(3), 265–276.

Hu, G., Zhang, Y., Chen, W. (2019). Exploring the performance of singularity for high performance computing scenarios. Dans *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2587–2593.

Huang, K.-H., Abraham, J.A. (1984). Algorithm-based fault tolerance for matrix operations. *IEEE transactions on computers*, 100(6), 518–528.

Hunold, S. (2015). A survey on reproducibility in parallel computing. *arXiv preprint arXiv:1511.04217*, 2015.

Ioannidis, J.P. (2022). Correction: Why most published research findings are false. *PLoS Medicine*, 19(8), e1004085.

Ioannidis, J.P. (2015). How to make more published research true. *Revista Cubana de Información en Ciencias de la Salud (ACIMED)*, 26(2), 187–200.

Ioannidis, J.P. (2005). Why most published research findings are false. *PLoS medicine*, 2(8), e124.

Iqbal, S.A., Wallach, J.D., Khoury, M.J., Schully, S.D., Ioannidis, J.P. (2016). Reproducible research practices and transparency across the biomedical literature. *PLoS biology*, 14(1), e1002333.

Ivie, P., Thain, D. (2018). Reproducibility in scientific computing. *ACM Computing Surveys (CSUR)*, 51(3), 1–36.

Jacobsen, D.M., Canon, R.S. (2015). Contain this, unleashing docker for hpc. *Proceedings of the Cray User Group*, 2015, 33–49.

Janz, N. (2016). Bringing the gold standard into the classroom: replication in university teaching. *International Studies Perspectives*, 17(4), 392–407.

Jézéquel, F., Lamotte, J.-L., Saïd, I. (2015). Estimation of numerical reproducibility on CPU and GPU. Dans *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 675–680.

Johnson, A.L., Johnson, B.C. (1997). Literate programming using noweb. *Linux Journal*, 42, 64–69.

Keller Tesser, R., Borin, E. (2023). Containers in HPC: a survey. *The Journal of Supercomputing*, 79(5), 5759–5827.

Kerr, N.L. (1998). HARKing: Hypothesizing after the results are known. *Personality and social psychology review*, 2(3), 196–217.

Kitzes, J., Turek, D., Deniz, F. (2018). *The practice of reproducible research: case studies and lessons from the data-intensive sciences* Univ of California Press.

Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A. (2007). kvm: the Linux virtual machine monitor. Dans *Proceedings of the Linux symposium*. Dttawa, Dntorio, Canada, 225–230.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B.E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J.B., Grout, J., Corlay, S. (2016). Jupyter Notebooks-a publishing format for reproducible computational workflows. *Elpub*, 2016, 87–90.

Knuth, D.E. (1984). Literate programming. *The computer journal*, 27(2), 97–111.

Knuth, D.E., Levy, S. (1994). *The CWEB system of structured documentation: version 3.0* Addison-Wesley Longman Publishing Co., Inc.

Köster, J., Rahmann, S. (2012). Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19), 2520–2522.

Kovacevic, J. (2007). How to encourage and publish reproducible research. Dans *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*. IEEE, IV-1273-IV–1276.

Kurkowski, S., Camp, T., Colagrosso, M. (2005). MANET simulation studies: the incredibles. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(4), 50–61.

Kurtzer, G.M., Sochat, V., Bauer, M.W. (2017). Singularity: Scientific containers for mobility of compute. *PloS one*, 12(5), e0177459.

Lange, M. (2022). Toward accurate and fast summation. *ACM Transactions on Mathematical Software (TOMS)*, 48(3), 1–39.

Langezaal, E.R., Belleman, J., Veenboer, T., Noorthoek, J. (2023). [Re] Label-Free Explainability for Unsupervised Models. *ReScience C*, 9(2), #4.

Le, E., Paz, D. (2017). Performance analysis of applications using singularity container on sdsc comet. Dans *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*. 1–4.

L'ecuyer, P. (1999). Good parameters and implementations for combined multiple recursive random number generators. *Operations Research*, 47(1), 159–164.

Lee, D., Wester, B., Veeraraghavan, K., Narayanasamy, S., Chen, P.M., Flinn, J. (2010). Respec: efficient online multiprocessor replayvia speculation and external determinism. *ACM Sigplan Notices*, 45(3), 77–90.

Legrand, A., Velho, P. (2023). [Re] Velho and Legrand (2009) - Accuracy Study and Improvement of Network Simulation in the SimGrid Framework. *ReScience C*, 6(1), #20.

Lenth, R.V. (2012). StatWeave users' manual. *URL http://www. stat. uiowa*, 2012.

Lenth, R.V., Højsgaard, S. (2007). SASweave: Literate programming using SAS. *Journal of Statistical Software*, 19, 1–20.

Liu, J., Pacitti, E., Valduriez, P., Mattoso, M. (2015). A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13, 457–493.

Ludwig, T. (2019). Bitwise reproducibility with exascale machines. , June 2019. [En ligne]. Disponible à l'adresse : https://www.cs.fsu.edu/~nre/nre-2019/presentations/03-Ludwig.2019-06-20-ISC-Reproducibility.pdf [Consulté le 1 Janvier 2024].

Manninen, T., Havela, R., Linne, M.-L. (2017). Reproducibility and comparability of computational models for astrocyte calcium excitability. *Frontiers in neuroinformatics*, 11, 11.

Marwick, B. (2015). How computers broke science—and what we can do to fix it. *The Conversation*, 2015.

Mashtizadeh, A.J., Garfinkel, T., Terei, D., Mazieres, D., Rosenblum, M. (2017). Towards practical default-on multi-core record/replay. *ACM SIGPLAN Notices*, 52(4), 693–708.

Matsumoto, M., Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1), 3–30.

Matthews, J.N., Hu, W., Hapuarachchi, M., Deshane, T., Dimatos, D., Hamilton, G., McCabe, M., Owens, J. (2007). Quantifying the performance isolation properties of virtualization systems. Dans *Proceedings of the 2007 workshop on Experimental computer science*. 6-es.

Mauerer, W., Scherzinger, S. (2022). 1-2-3 reproducibility for quantum software experiments. Dans *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 1247–1248.

Meneses, E., Mendes, C.L., Kalé, L.V. (2010). Team-based message logging: Preliminary results. Dans *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE, 697–702.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux j*, 239(2), 2.

Mesnard, O., Barba, L.A. (2017). Reproducible and replicable computational fluid dynamics: it's harder than you think. *Computing in Science & Engineering*, 19(4), 44–55.

Miller, G. (2006). *A scientist's nightmare: software problem leads to five retractions* American Association for the Advancement of Science.

Ministère de l'enseignement supérieur et de la recherche (2021). Second National Plan for Open Science. , 2021. [En ligne]. Disponible à l'adresse : https://www.ouvrirlascience.fr/second-national-plan-for-open-science/ [Consulté le 1 Janvier 2024].

Mittal, S., Vetter, J.S. (2015). A survey of techniques for modeling and improving reliability of computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(4), 1226–1238.

Munafò, M.R., Nosek, B.A., Bishop, D.V., Button, K.S., Chambers, C.D., Percie du Sert, N., Simonsohn, U., Wagenmakers, E.-J., Ware, J.J., Ioannidis, J. (2017). A manifesto for reproducible science. *Nature human behaviour*, 1(1), 1–9.

Mytkowicz, T., Diwan, A., Hauswirth, M., Sweeney, P.F. (2009). Producing wrong data without doing anything obviously wrong!. *ACM Sigplan Notices*, 44(3), 265–276.

National Academies of Sciences, E. (2019). Reproducibility and replicability in science. , 2019.

Normand, E. (1996). Single event upset at ground level. *IEEE transactions on Nuclear Science*, 43(6), 2742–2750.

Nüst, D., Sochat, V., Marwick, B., Eglen, S.J., Head, T., Hirst, T., Evans, B.D. (2020). *Ten simple rules for writing Dockerfiles for reproducible data science* Public Library of Science San Francisco, CA USA.

Nuzzo, R. (2014). Statistical errors. *Nature*, 506(7487), 150.

Ogasawara, E., Dias, J., Silva, V., Chirigati, F., De Oliveira, D., Porto, F., Valduriez, P., Mattoso, M. (2013). Chiron: a parallel engine for algebraic

scientific workflows. *Concurrency and Computation: Practice and Experience*, 25(16), 2327–2341.

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A. (2004). Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17), 3045–3054.

O'neill, M.E. (2014). PCG: A family of simple fast space-efficient statistically good algorithms for random number generation. *ACM Transactions on Mathematical Software*, 2014.

Onose, E. (2023). How to Solve Reproducibility in Machine Learning. , 13 November 2023. [En ligne]. Disponible à l'adresse : https://neptune.ai/blog/how-to-solve-reproducibility-in-ml [Consulté le 1 January 2024].

Open Science Collaboration (2015). Estimating the reproducibility of psychological science. *Science*, 349(6251), aac4716.

Padala, P., Zhu, X., Wang, Z., Singhal, S., Shin, K.G. (2007). Performance evaluation of virtualization technologies for server consolidation. *HP Labs Tec. Report*, 137, 19.

Panneton, F., L'ecuyer, P., Matsumoto, M. (2006). Improved long-period generators based on linear recurrences modulo 2. *ACM Transactions on Mathematical Software (TOMS)*, 32(1), 1–16.

Park, S., Zhou, Y., Xiong, W., Yin, Z., Kaushik, R., Lee, K.H., Lu, S. (2009). Pres: probabilistic replay with execution sketching on multiprocessors. Dans *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 177–192.

Patil, H., Pereira, C., Stallcup, M., Lueck, G., Cownie, J. (2010). Pinplay: a framework for deterministic replay and reproducible analysis of parallel

programs. Dans *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*. 2–11.

Perianayagam, S., Andrews, G.R., Hartman, J.H. (2010). Rex: a toolset for reproducing software experiments. Dans *2010 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 613–617.

Pernet, C.R., Wilcox, R., Rousselet, G.A. (2013). Robust correlation analyses: false positive and power validation using a new open source matlab toolbox. *Frontiers in psychology*, 3, 606.

Piller, C., Servick, K. (2020). *Two elite medical journals retract coronavirus papers over data integrity questions. Science ScienceMag. org. 2020* [En ligne]. Disponible à l'adresse : https://www.science.org/content/article/two-elite-medical-journals-retract-coronavirus-papers-over-data-integrity-questions.

PLOS one Materials and Software Sharing. [En ligne]. Disponible à l'adresse : https://journals.plos.org/plosone/s/materials-and-software-sharing [Consulté le 1 Janvier 2024].

Popper, K. (2005). *The logic of scientific discovery* Routledge.

Potdar, A.M., Narayan, D.G., Kengond, S., Mulla, M.M. (2020). Performance evaluation of docker container and virtual machine. *Procedia Computer Science*, 171, 1419–1428.

Pouzat, C. (2022). Pourquoi devrions-nous arrêter d'embêter les gens avec la «recherche reproductible» et autres «bonnes pratiques»? *Statistique et Société*, 10(1), 53–57.

Pouzat, C. (2021). Un panorama de la recherche reproductible. *Bulletin de la ROADEF*, 2021.

Pouzat, C., Legrand, A., Hinsen, K. (2018). Recherche Reproductible: Principes Methodologiques pour une Science Transparente. 2018. [En

ligne]. Disponible à l'adresse : https://www.fun-mooc.fr/fr/cours/recherche-reproductible-principes-methodologiques-pour-une-science-transparente/ [Consulté le 2 Janvier 2024].

Pradal, C., Fournier, C., Valduriez, P., Cohen-Boulakia, S. (2015). OpenAlea: scientific workflows combining data analysis and simulation. Dans *Proceedings of the 27th International Conference on Scientific and Statistical Database Management*. 1–6.

Priedhorsky, R., Randles, T. (2017). Charliecloud: Unprivileged containers for user-defined software stacks in hpc. Dans *Proceedings of the international conference for high performance computing, networking, storage and analysis*. 1–10.

Rad, B.B., Bhatti, H.J., Ahmadi, M. (2017). An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS)*, 17(3), 228.

Ragan-Kelley, B., Willing, C., Akici, F., Lippa, D., Niederhut, D., Pacer, M. (2018). Binder 2.0-Reproducible, interactive, sharable environments for science at scale. Dans *Proceedings of the 17th python in science conference*. F. Akici, D. Lippa, D. Niederhut, and M. Pacer, eds., 113–120.

Randall, D., Welser, C. (2018). *The Irreproducibility Crisis of Modern Science: Causes, Consequences, and the Road to Reform.* ERIC.

Ren, S., Li, C., Tan, L., Xiao, Z. (2015). Samsara: Efficient deterministic replay with hardware virtualization extensions. Dans *Proceedings of the 6th Asia-Pacific workshop on systems*. 1–7.

Revol, N., Théveny, P. (2014). Numerical reproducibility and parallel computations: Issues for interval algorithms. *IEEE Transactions on Computers*, 63(8), 1915–1924.

Rijmen, V., Daemen, J. (2001). Advanced encryption standard. *Proceedings of federal information processing standards publications, national institute of standards and technology*, 19, 22.

Rosenquist, T. (2012). Run-to-run Numerical Reproducibility with the Intel® Math Kernel Library and Intel® Composer XE 2013. , 2012.

Rougier, N.P., Hinsen, K., Alexandre, F., Arildsen, T., Barba, L.A., Benureau, F.C., Brown, C.T., De Buyl, P., Caglayan, O., Davison, A.P. (2017). Sustainable computational science: the ReScience initiative. *PeerJ Computer Science*, 3, e142.

Royal Society (1660). History of the Royal Society. , 1660. [En ligne]. Disponible à l'adresse : https://royalsociety.org/about-us/history/ [Consulté le 1 Janvier 2024].

Ruiz, C., Jeanvoine, E., Nussbaum, L. (2015). Performance evaluation of containers for HPC. Dans *Euro-Par 2015: Parallel Processing Workshops: Euro-Par 2015 International Workshops, Vienna, Austria, August 24-25, 2015, Revised Selected Papers 21*. Springer, 813–824.

Ruiz, C., Richard, O., Emeras, J. (2014). Reproducible software appliances for experimentation. Dans *Testbeds and Research Infrastructure: Development of Networks and Communities: 9th International ICST Conference, TridentCom 2014, Guangzhou, China, May 5-7, 2014, Revised Selected Papers 9*. Springer, 33–42.

Rump, S.M., Ogita, T., Oishi, S. (2010). Fast high precision summation. *Nonlinear Theory and Its Applications, IEICE*, 1(1), 2–24.

Rutherford, E., Royds, T. (1908). LXVIII. The action of the radium emanation upon water. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 16(95), 812–818.

Saito, M., Matsumoto, M. (2006). SIMD-oriented fast Mersenne Twister: a 128-bit pseudorandom number generator. Dans *Monte Carlo and Quasi-Monte Carlo Methods 2006*. Springer, 607–622.

Salmon, J.K., Moraes, M.A., Dror, R.O., Shaw, D.E. (2011). Parallel random numbers: as easy as 1, 2, 3. Dans *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis*. 1–12.

Schroeder, B., Pinheiro, E., Weber, W.-D. (2009). DRAM errors in the wild: a large-scale field study. *ACM SIGMETRICS Performance Evaluation Review*, 37(1), 193–204.

Schulte, E., Davison, D., Dye, T., Dominik, C. (2012). A multi-language computing environment for literate programming and reproducible research. *Journal of Statistical Software*, 46, 1–24.

Schwab, M., Karrenbach, N., Claerbout, J. (2000). Making scientific computations reproducible. *Computing in Science & Engineering*, 2(6), 61–67.

Senthil Kumaran, S. (2017). *Practical LXC and LXD: linux containers for virtualization and orchestration* Springer.

Shaydulin, R., Marwaha, K., Wurtz, J., Lotshaw, P.C. (2021). QAOAKit: A toolkit for reproducible study, application, and verification of the QAOA. Dans *2021 IEEE/ACM Second International Workshop on Quantum Computing Software (QCS)*. IEEE, 64–71.

Sommerville, I. (2001). Software documentation. *Software engineering*, 2, 143–154.

Stallman, R.M. (1981). EMACS the extensible, customizable self-documenting display editor. Dans *Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation*. 147–156.

Stanisic, L., Legrand, A., Danjean, V. (2015). An effective git and org-mode based workflow for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1), 61–70.

Stodden, V. (2011). Trust your science? Open your data and code. , 2011.

Stodden, V., Borwein, J., Bailey, D.H. (2013). Setting the default to reproducible. *computational science research. SIAM News*, 46(5), 4–6.

Stodden, V., Krafczyk, M.S., Bhaskar, A. (2018). Enabling the verification of computational results: An empirical evaluation of computational reproducibility. Dans *Proceedings of the First International Workshop on Practical Reproducible Evaluation of Computer Systems*. 1–5.

Stodden, V., Leisch, F., Peng, R.D. (2014). *Implementing reproducible research* CRC Press.

Strong, R.W., Alvarez, G. (2019). Using simulation and resampling to improve the statistical power and reproducibility of psychological research. , 2019.

Taufer, M., Padron, O., Saponaro, P., Patel, S. (2010). Improving numerical reproducibility and stability in large-scale numerical simulations on GPUs. Dans *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 1–9.

Taylor, I., Shields, M., Wang, I., Harrison, A. (2007). The triana workflow environment: Architecture and applications. *Workflows for e-science: Scientific workflows for grids*, 2007, 320–339.

Ten Hagen, K.G. (2016). Novel or reproducible: That is the question. *Glycobiology*, 26(5), 429–429.

Topalidou, M., Leblois, A., Boraud, T., Rougier, N.P. (2015). A long journey into reproducible computational neuroscience. *Frontiers in computational neuroscience*, 9, 30.

Torrez, A., Randles, T., Priedhorsky, R. (2019). HPC container runtimes have minimal or no performance impact. Dans *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. IEEE, 37–42.

Utterback, R., Agrawal, K., Lee, I.-T.A., Kulkarni, M. (2017). Processor-oblivious record and replay. *ACM SIGPLAN Notices*, 52(8), 145–161.

Vandewalle, P., Kovacevic, J., Vetterli, M. (2009). Reproducible research in signal processing. *IEEE Signal Processing Magazine*, 26(3), 37–47.

Wang, C., Dryden, N., Cappello, F., Snir, M. (2018). Neural network based silent error detector. Dans *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 168–178.

Wilkinson, M.D., Dumontier, M., Aalbersberg, Ij.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L.B., Bourne, P.E. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data*, 3(1), 1–9.

Wilson, G., Aruliah, D.A., Brown, C.T., Chue Hong, N.P., Davis, M., Guy, R.T., Haddock, S.H., Huff, K.D., Mitchell, I.M., Plumbley, M.D. (2014). Best practices for scientific computing. *PLoS biology*, 12(1), e1001745.

Wu, X., Vijayakumar, K., Mueller, F., Ma, X., Roth, P.C. (2011). Probabilistic communication and i/o tracing with deterministic replay at scale. Dans *2011 International Conference on Parallel Processing*. IEEE, 196–205.

Wu, X., Mueller, F. (2013). Elastic and scalable tracing and accurate replay of non-deterministic events. Dans *Proceedings of the 27th international ACM conference on International conference on supercomputing*. 59–68.

Xavier, M.G., Neves, M.V., Rossi, F.D., Ferreto, T.C., Lange, T., De Rose, C.A. (2013). Performance evaluation of container-based

virtualization for high performance computing environments. Dans *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE, 233–240.

Xue, R., Liu, X., Wu, M., Guo, Z., Chen, W., Zheng, W., Zhang, Z., Voelker, G. (2009). MPIWiz: Subgroup reproducible replay of MPI applications. Dans *Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming*. 251–260.

Yong, C., Lee, G.-W., Huh, E.-N. (2018). Proposal of container-based HPC structures and performance analysis. *Journal of Information Processing Systems*, 14(6), 1398–1404.

Younge, A.J., Pedretti, K., Grant, R.E., Brightwell, R. (2017). A tale of two systems: Using containers to deploy HPC applications on supercomputers and clouds. Dans *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 74–81.

Zhao, Y., Hategan, M., Clifford, B., Foster, I., Von Laszewski, G., Nefedova, V., Raicu, I., Stef-Praun, T., Wilde, M. (2007). Swift: Fast, reliable, loosely coupled parallel computation. Dans *2007 IEEE Congress on Services (Services 2007)*. IEEE, 199–206.

Zhou, N., Zhou, H., Hoppe, D. (2022). Containerization for High Performance Computing Systems: Survey and Prospects. *IEEE Transactions on Software Engineering*, 49(4), 2722–2740.

Zolfagharifard, E., Boland, H. (2020). Coding that led to lockdown 'totally unreliable' and 'a buggy mess', say experts. , Mai 2020. [En ligne]. Disponible à l'adresse : https://www.telegraph.co.uk/technology/2020/05/16/coding-led-lockdown-totally-unreliable-buggy-mess-say-experts/ [Consulté le 1 Janvier 2024].