

Rapport de Stage

Par **Clément LONGEAC**

Dirigé par **Emmanuel FARHI**

Et par **Fréderic-Emmanuel PICCA**

Présentée et soutenue le : 05/09/2025

SOLEIL SYNCHROTRON

Membre fondateur
Université Gustave Eiffel

Introduction

« En travaillant en étroite collaboration avec les plus grands innovateurs technologiques de notre pays et NVIDIA, nous fournissons aux chercheurs, aux entrepreneurs et aux institutions publiques les outils dont ils ont besoin pour explorer de nouvelles idées, relever des défis complexes et contribuer à façonner l'avenir de l'IA en France. »

Tels sont les mots d'Emmanuel Macron lors d'un communiqué de presse du 11 juin 2025. L'entreprise NVIDIA a annoncé travailler avec les nations européennes et les leaders technologiques et industriels pour construire l'infrastructure d'IA NVIDIA Blackwell qui a pour objectif de renforcer la souveraineté numérique et de positionner le continent comme un leader dans la révolution industrielle de l'IA. NVIDIA est une entreprise américaine spécialisée dans la conception de processeurs graphiques (GPU) et de cartes graphiques. Aujourd'hui, le marché est détenu à 88 % par NVIDIA. Depuis plus de 15 ans, NVIDIA prend de l'avance sur la conception et la mise en place des technologies qui ont permis l'avènement de l'IA et de l'apprentissage profond. L'influence de cette entreprise est telle qu'elle est désormais la première capitalisation boursière au monde, dépassant l'ensemble des GAFAM. Alors que l'IA est en plein boom et que cela nécessite une grande puissance de calcul, les concurrents du géant mondial peinent à rivaliser. Le risque est donc une dépendance effective et inéluctable à NVIDIA qui aujourd'hui augure un monopole certain sur les marchés des futures technologies de l'IA. C'est d'ailleurs ce que l'on entend dans le discours du Président français. Le fait est qu'il est donc ainsi devenu vital pour les sociétés de se détacher de cette dépendance croissante et de développer leur propre IA, de posséder leurs propres datacenters... Là sont tous les enjeux géopolitiques qui se réfugient derrière ce que certains appellent déjà « la Troisième Révolution Industrielle ».

« Inspirer une génération », telle est la devise des Jeux Olympiques de 2012 qui ont permis de mettre à l'honneur les travaux du CERN sur la scène mondiale. Je suis de cette génération inspirée par cette Troisième Révolution industrielle, mais je suis aussi inspiré à comprendre le monde et à repousser toujours plus loin les limites de la science. C'est ainsi, avec cette même inspiration, que j'ai postulé au laboratoire Synchrotron-Soleil pour un stage de 4 mois dans la division Div/Exp GRADES sous la direction de M. Emmanuel FARHI et M. Frédéric-Emmanuel PICCA afin de réaliser des tests d'intégrations de paquets sur Salsa Debian, contribuant à se détacher du monopole scientifique et politique d'NVIDIA sur les cartes graphiques en testant leur compatibilité avec AMD, son concurrent commercial. Et c'est en ce sens que mon stage prend sens, je ne me contente pas d'étudier les dessous de l'informatique et de la data science, j'apprends à contextualiser la science dans un monde économique, aux enjeux géopolitiques. Cela contribue à me former à devenir l'ingénieur polyvalent de demain que je souhaiterais être.

Ce stage rentre donc dans la continuité de mon parcours et de mon projet professionnel, puisque je souhaite poursuivre une spécialité « Datascience & IA » dès l'année prochaine dans la continuité de ma formation au sein de l'ESIEE-PARIS. Je souhaite effectuer mes prochains stages de E4 et de E5 à l'international afin de me former au mieux dans des environnements d'une grande diversité. Ce stage me permet de mettre en pratique les connaissances que j'ai pu acquérir sur les bancs de l'école et d'en acquérir de nouvelles. Je cherche non seulement à apprendre, à m'initier véritablement au monde de la recherche, mais aussi à mon échelle, contribuer à demain. Ce stage est une immense opportunité d'accomplir non seulement un rêve d'enfant mais également d'être formé à devenir un ingénieur de demain, toujours dans la perspective de faire de ma passion, mon métier, et devenir un datascientist polyvalent.

C'est dans ce contexte que ce stage a été initié, les compétences et les connaissances nécessaires font directement appel à la formation qu'ESIEE-PARIS propose. Notamment sur les notions traitants de l'utilisation de Gitlab / GitHUb via le projet de l'unité E3_IGI_3008 - Apprentissage de la programmation avec Python et C , où l'objectif était de coder un jeu de type TBA. Les notions de E3_IGI_3012 - Réseaux , et autres unités utilisant le système Debian m'ont permis d'aborder plus aisément ce stage. L'explication et le détail de mes missions feront l'objet d'une partie. J'y expliquerai en détail et avec rigueur les missions qui m'ont été confiées au cours de ces 4 mois de stages.

Pour ce faire, afin de rendre un rapport qui se veut tant compte rendu que rapport de stage, j'aborderai tout d'abord le contexte de mon stage en expliquant ce qu'est SOLEIL et de manière générale un accélérateur de particule. Ce qui m'amènera dans un second temps à contextualiser l'objectif de mon stage avant d'enfin dans un dernier temps pouvoir détailler les missions qui m'ont été confiées.

Je vous remercie pour le temps que vous accorderez à la lecture de mes travaux.

Abstract

NVIDIA's influence and market share in the graphics card market continue to grow. At the time of writing this report. The American company becomes the very first company to exceed 4 trillion in market valuation. It holds more than 92% of the market as of June 9, 2025 [2]. Unfortunately, this leaves little room for competition, with AMD holding only 8% of the GPU market. It is a major issue to avoid at all costs one company having a monopoly in a market with such technological and geopolitical influence. A company with a monopoly sets its own rules of the game, regardless of the protests of the players, who in this case are the customers. Moreover, having a commercial competitor ensures the evolution of technologies and forces the company to excel in this field. That's why, at Synchrotron-SOLEIL, we are trying to detach ourselves from NVIDIA in order to ensure our sovereignty in light of the challenges raised. The software that Synchrotron – SOLEIL uses to process and acquire data uses Debian packages, which form the building blocks for performing the necessary calculations for data analysis. These include, for example, packages that allow for the calculation of Fourier transforms, or, in specific cases, enable the processing of data from tomographies. In this context, it is important to know if these packages are compatible with GPUs, NVIDIA or AMD. That's why we wanted to test their compatibility by running autopkgtests that use OpenCL code. NVIDIA uses CUDA as its GPU technology, unlike AMD which uses ROCm as its GPU. With a view to moving to AMD, our goal is to test the compatibility of Debian packages with AMD GPUs. For this, our goal is to perform OpenCL autopkgtests on ROCm on the AMD GPU we have, and implement them on the packages we want to test compatibility with. We will ensure that the dependencies necessary for building the package are present and that the environment variables as well as the chroot are correctly configured on Debian. The implementation of the tests is done according to the method established in this report. The modifications are tested by running the "sbuild" command, after taking care to configure them. The build is done in a "podman+rocm" container. Moreover, we first verify that the processed package actually contains OpenCL code and associated tests, whether unit or not. After implementing these OpenCL autopkgtests for ROCm, and testing the build on the different packages to be processed. We were able to equip the packages PyOpenCL, Nabu, Ufo-filters, clpeak, pyvkfft, gpyfft, silx, sasmodels with ROCm autotests. Nevertheless, some selected packages did not contain OpenCL code, which made the implementation of autopkgtests less interesting, particularly packages like genx, examl. Moreover, packages like pysph have intrinsic issues with the source code; the migration from Python 2 to Python 3 has not been completed, making it unusable. The changes are then pushed to the git-salsa repository of the Debian package so that the modifications become effective for the entire community. So, these OpenCL autopkgtests for ROCm are there to test compatibility with AMD GPUs in order to allow as many Debian packages as possible to enable this compatibility and thus facilitate the transition to AMD GPUs. These are therefore the first building blocks enabling this transition from NVIDIA to AMD in order to limit NVIDIA's influence on the GPU market. From now on, it will be easier to make this transition after testing their compatibilities within Synchrotron-SOLEIL. In reality, the work done locally at Synchrotron-SOLEIL benefits the entire international Debian scientific community.

Keywords: Debian, Packaging, Computing, Autopkgtest, ROCm,

Remerciements

Ce stage a rythmé ma vie pendant les quatre derniers mois, et bien que ce travail soit le mien, il n'aurait pas été réalisable sans les contributions directes d'un grand nombre de personnes. Je vais donc faire de mon mieux pour n'oublier personne, mais si j'échoue, ne le prenez pas personnellement s'il vous plaît.

Tout d'abord, j'ai une immense gratitude envers mes deux maîtres de stage au sein de Synchrotron SOLEIL, qui, sans leur présence, tout ceci n'aurait été possible, et qui m'ont accordé leur confiance pour le mener à bien. Je pense tout particulièrement à M.Emmanuel FARHI, chef de la division GRADES, et à M.Frederic-Emmanuel PICCA, développeur Debian au sein de GRADES. Cela a été un réel plaisir de travailler aux côtés d'Emmanuel et de Fred, ils ont été d'un grand soutien tous les deux. Les explications détaillées d'Emmanuel et de Fred m'ont tout de suite permis de comprendre les enjeux éthiques et techniques derrière mes travaux. Malgré le travail effectif, tout ceci a un réel sens dans le cadre de la recherche scientifique. Celle-ci se doit d'être souveraine. Même face à ce sujet sérieux, votre sens de l'humour n'était jamais très loin ! Je n'avais jusqu'alors jamais fait de packaging Debian, mais la pédagogie et la patience d'Emmanuel m'ont permis d'avancer rapidement. En effet, malgré les contretemps techniques, tous deux ont été d'une immense bienveillance à mon égard. Fred et Emmanuel, vous êtes deux excellents encadrants et votre disponibilité, lorsque je passais dans vos bureaux pour 10 minutes qui se transformaient en plusieurs heures, ont été d'une aide inestimable, malgré vos emplois du temps bien remplis. Ce stage est pour moi le début de l'accomplissement d'un rêve, travailler dans un synchrotron. Cela n'est que le début de l'aventure, qui me mènera un jour, je l'espère, à travailler dans le plus grand accélérateur de particules du monde : le Centre Européen de la Recherche Nucléaire (CERN).

Évidemment, je ne peux que remercier mon collègue stagiaire Ethan GUILLOT, pour son soutien, son humour et sa bienveillance. Tu as su me mettre sur la piste de solutions concernant certains de mes problèmes informatiques sur le passage de ces tests. Je tiens tout particulièrement à remercier également pour sa gentillesse et ses précieux conseils, Olga ROUDENKO.

En dehors des équipes de Synchrotron SOLEIL, je voudrais aussi remercier tout particulièrement Monsieur Etienne DURIS, qui a su m'accorder sa confiance et son soutien dans mon projet de stage. En effet, ce stage n'est pas du tout obligatoire et très rare en E3, mais lorsque je vous ai parlé de mon ambition de faire ce stage dans le but de réaliser mon rêve, vous avez tout de suite soutenu mon projet. Vous m'avez également énormément accompagné durant toute la phase administrative, un tant soit peu complexe. Je vous remercie de votre bienveillance et du temps que vous m'avez accordé. Malgré la réponse tardive de la validation de mon stage au sein de Synchrotron SOLEIL, les formalités administratives ont été rapidement bouclées. Je tiens donc également à remercier toute l'équipe administrative en charge des stages pour leur réactivité et leurs explications de qualité sur la manière de procéder, même si cela s'avère en pratique assez technique par endroit. Je voudrais aussi remercier Monsieur Jean-Marc POINTET, mon responsable de filière, pour son soutien et la validation de stage, qui compte tant pour moi, et de l'avoir rendu possible.

Sur le plan personnel, je tiens à remercier ma famille qui a été d'un grand soutien, sur le plan humain et scolaire. Je tiens à remercier Mme Honorine BLAEVOET pour son soutien et son expertise en littérature, je tiens tout particulièrement à te remercier pour ta relecture de ce rapport et les corrections littéraires apportées.

Enfin, je remercie toutes les personnes que j'ai pu croiser sur ma route, pour leurs conseils et leur soutien indéfectible.

Table des matières

Introduction

Abstract

Remerciements

Listes des abréviations

Avant-propos

1. Principe et fonctionnement d'un accélérateur d'électron

2. Les lignes de lumières

Partie 1 : Pourquoi ?

1. Qu'est-ce que Synchrotron SOLEIL en bref et quelle est sa position au sein de la recherche mondiale ?
2. Le stage
3. Fonctionnement Interne de L'entreprise

Partie 2 : Debian

1. Qu'est-ce que le système open source Debian ?
2. Comment Debian fonctionne-t-il ?
3. Les supports matériels CPU et GPU

Partie 3 : Mes missions

1. Les procédures et Setup
2. Les paquets Debian à traiter et leurs traitements avec les problématiques rencontrées

Conclusion

Bibliographie

Annexe

Liste des abréviations

SOLEIL : Source Optimisée de Lumière d'Energie Intermédiaire du LURE

IA : Intelligence artificielle

ROCm : Radeon Open Compute platform

HPC: High Performance Computing

POCL: Portable Computing Language (PoCL)

CGPU: Computing on GPU

CUDA : Compute Unified Device Architecture

Avant-propos

Sommaire

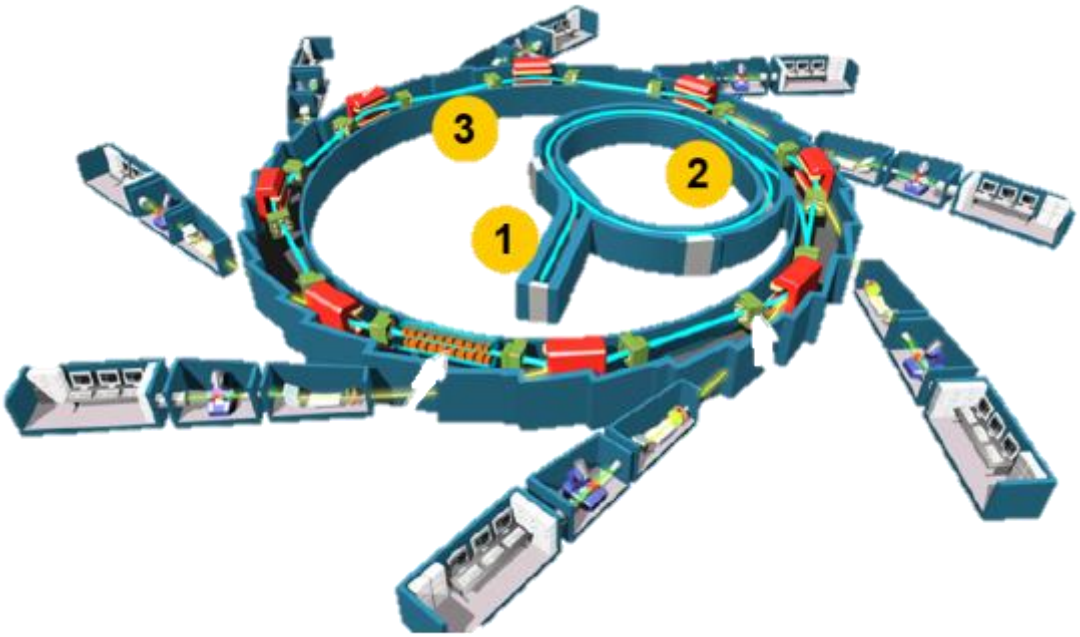
1. Principe et fonctionnement d'un accélérateur d'électron

2. Le synchrotron et les lignes de lumières dans le cas de Synchrotron SOLEIL [3]

2.1 Une ligne de lumière c'est quoi ?

2.2 Localisations des lignes

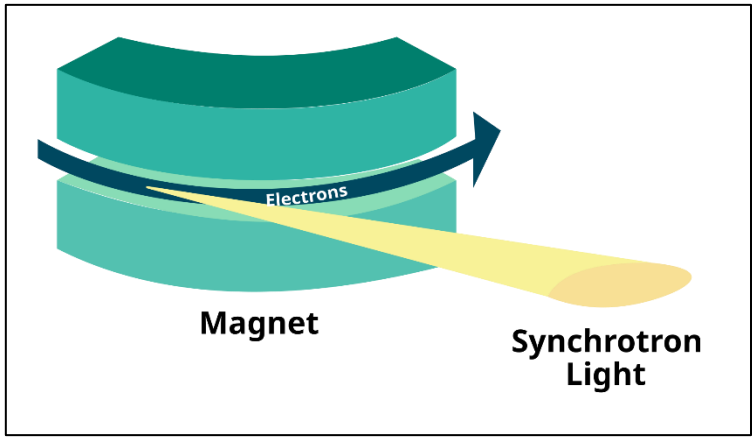
1. Principe et fonctionnement d'un accélérateur d'électron



Titre : Schéma de principe du synchrotron comportant le booster, l'anneau lui-même et les lignes de lumières

Qu'est-ce qu'un accélérateur d'électron (synchrotron) et quel est son principe physique ?

Un accélérateur propulse des particules chargées, telles que des protons ou des électrons, à des vitesses élevées, proches de celle de la lumière. Elles sont ensuite projetées sur une cible ou contre d'autres particules circulant en sens inverse. Dans le cas de Synchrotron SOLEIL, les particules accélérées sont des électrons, ils ne sont quant à eux pas destinés à entrer en collision entre eux. En réalité tout l'intérêt du synchrotron SOLEIL est de permettre le rayonnement synchrotron issu de la courbure du faisceau d'électrons selon différentes gammes de fréquence du spectre électromagnétique allant des rayons UV au visible en passant par les rayons X dur. Par le biais du rayonnement capté par les lignes de lumière, ce rayonnement permet l'étude des matériaux à l'échelle nanométrique.



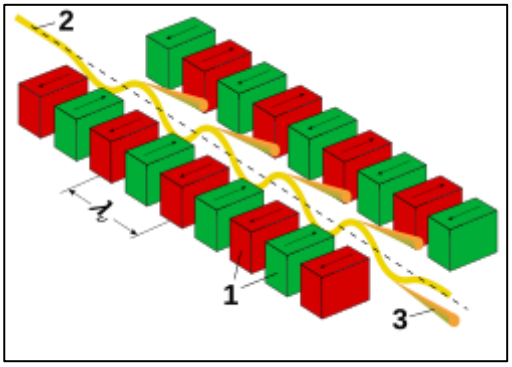
Titre : Rayonnement synchrotron produit par un aimant de courbure [5]

Lorsque les électrons ont suffisamment d'énergie, ici 2.4 Gev (Giga électron Volt), un phénomène inimaginable se produit lorsque le faisceau d'électrons est courbé : les électrons perdent de l'énergie lors de la courbure du faisceau, et se faisant les électrons émettent désormais de l'énergie sous forme de photons et c'est donc ce qu'on appelle le rayonnement synchrotron qui correspond ainsi à l'accélération subie. Le rayonnement produit de cette manière, possède une polarisation caractéristique, et les fréquences générées peuvent s'étendre sur une grande partie du spectre électromagnétique.

Le rayonnement synchrotron est similaire au rayonnement continu de freinage, qui est émis par une particule chargée lorsque son accélération est parallèle à la direction du mouvement. Le rayonnement émis par les particules dans un champ magnétique est le « rayonnement gyromagnétique », pour lequel le rayonnement synchrotron est le cas particulier ultra-relativiste. Le rayonnement émis par des particules chargées se déplaçant de manière non relativiste dans un champ magnétique appelé « rayonnement cyclotron ». Pour les particules dans la gamme légèrement relativiste (≈ 85 % de la vitesse de la lumière), l'émission est appelée « rayonnement gyrosynchrotron » [5].

Titre : Rayonnement synchrotron émis par un ondulateur. [5]

1. Champ magnétique variable
2. Faisceau d'électron
3. Rayonnement synchrotron



La lumière synchrotron est aussi émise quand on fait osciller les électrons à faible amplitude, ce qui revient à une série de petits virages, avec un dispositif magnétique périodique appelé onduleur. La brillance ainsi atteinte peut être très élevée grâce à des interférences constructives.

La puissance des rayonnements émis par une particule chargée subissant une force perpendiculaire à sa trajectoire, généralement engendrée par un champ magnétique (voir force électromagnétique), peut être calculée grâce à la formule de Larmor relativiste :

$$P_\gamma = \frac{q^2}{6\pi\varepsilon_0 c^3} a^2 \gamma^4 = \frac{q^2 c}{6\pi\varepsilon_0} \frac{\beta^4 \gamma^4}{\rho^2},$$

où , ε_0 est la permittivité du vide ; q la charge électrique de la particule ; a l'amplitude de l'accélération ; c la vitesse de la lumière ; γ le facteur de Lorentz ; $\beta = \frac{v}{c}$ le rayon de courbure de la trajectoire de la particule. La force agissant sur l'électron émetteur est donnée par la force d'Abraham-Lorentz-Dirac.

Déjà partiellement évoqué plus haut, son utilisation permet de sonder la matière à des niveaux de détails encore jamais perçus.

Ce phénomène est décrit par la célèbre force d'Abraham-Lorentz :

$$\mathbf{f} = \frac{2}{3} \frac{q^2}{4\pi\epsilon_0} \frac{1}{c^5} \left(c^2 \ddot{\mathbf{v}} - |\mathbf{a}|^2 \mathbf{v} \right).$$

Si le rayonnement est émis par une particule se déplaçant dans un plan, le rayonnement est polarisé linéairement lorsqu'il est observé dans ce plan, et il est polarisé circulairement lorsqu'il est observé à un petit angle. Cependant, si l'on considère la mécanique quantique, ce rayonnement est émis dans des paquets discrets de photons et a des effets significatifs dans les accélérateurs, appelés « excitation quantique ». Pour une accélération donnée, l'énergie moyenne des photons émis est proportionnelle à γ^3 et le taux d'émission à γ .

Et c'est ce rayonnement dit synchrotron que l'on utilise à SOLEIL pour analyser et comprendre l'infiniment petit.

Dans un synchrotron, un champ électrique intense permet d'accélérer un faisceau de particules et un champ magnétique permet de dévier sa trajectoire. Dans le cas d'un synchrotron, ces particules sont généralement des électrons (plus rarement des positrons) et tournent à des vitesses relativistes. Les accélérateurs circulaires produisent toujours un rayonnement synchrotron lorsque les particules sont déviées dans le champ magnétique. Cependant, la quantité et les propriétés du rayonnement dépendent fortement de la nature de l'accélération qui se produit. Par exemple, en raison de la différence de masse (le facteur γ^4 dans la formule de la puissance émise), cela signifie que les électrons émettent de l'énergie à environ 10^{13} fois le taux des protons. La perte d'énergie par rayonnement synchrotron dans les accélérateurs circulaires était à l'origine considérée comme une nuisance, car une énergie supplémentaire devait être fournie au faisceau afin de compenser les pertes. Cependant, à partir des années 1980, des accélérateurs d'électrons circulaires connus sous le nom de « sources lumineuses » ont été construits pour produire délibérément des faisceaux intenses de rayonnement synchrotron pour la recherche.

Désormais, le rayonnement synchrotron étant émis, il est utilisé pour l'analyse de l'infiniment petit, de part d'utilisation d'une ligne de lumière dont nous développerons les caractéristiques dans la partie suivante de l'avant-propos afin de parfaire la compréhension du sujet qui nous incombe.

En effet, ce rayonnement dépend de la vitesse des électrons, mais couvre une très large partie du spectre électromagnétique, de l'infrarouge aux rayons X durs. Il est alors possible soit d'utiliser une gamme spectrale étendue (spectroscopie infrarouge à transformée de Fourier, diffraction de Laüe , spectre de Bragg), soit plus habituellement de monochromatiser ce faisceau blanc pour ne travailler qu'avec une bande très étroite de fréquences lumineuses. Dans le cadre de certaines expériences, absorption de rayons X utilisant la technique EXAFS ou la technique XANES par exemple, la possibilité de faire varier finement l'énergie du faisceau est un atout fondamental et permet de sonder précisément certaines transitions énergétiques. Le rayonnement synchrotron est particulièrement brillant (intense et focalisé), il peut l'être 10000 fois plus que la lumière solaire. Dans la gamme des rayons X durs, la faible divergence du faisceau permet la mise en œuvre de méthodes de micro-imagerie, à l'échelle de quelques fractions de micromètre sur les lignes les plus performantes (comme celles de l'ESRF ou de Synchrotron SOLEIL, respectivement à Grenoble et sur le plateau de Saclay). On distingue également des caractéristiques du rayonnement synchrotron, il est facilement calculable, très stable par rapport aux sources classiques, polarisé, pulsé, et permet de travailler en cohérence de phase.

On distingue dans le monde différents types d'accélérateur : d'électrons comme SOLEIL, de particules, d'ions etc, utilisant des technologies diverses. Nous avons le synchrotron, le cyclotron, le synchrocyclotron, des LINAC, des grands collisionneurs (comme le LHC), etc.

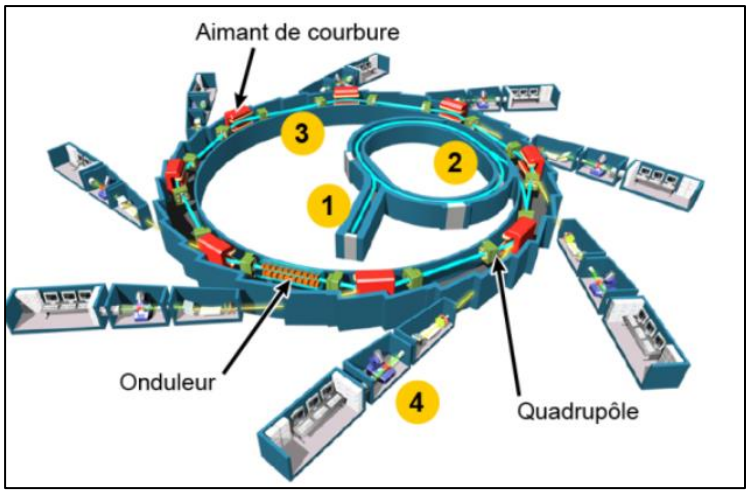
2. Le synchrotron et les lignes de lumières dans le cas de Synchrotron SOLEIL [3]



Titre : Image réelle de la structure d'une ligne de lumière

2.1 Une ligne de lumière c'est quoi ?

Les installations expérimentales d'un centre de rayonnement synchrotron s'appellent des « lignes de lumière ». Il s'agit d'un ensemble de cabanes successives où le rayonnement est recueilli, sélectionné, focalisé et dirigé vers les échantillons à étudier.



Titre : Organisation des composants principaux de l'accélérateur d'électron. [3]

Maintenant que nous avons compris comment fonctionne un accélérateur d'électron du point de vue du principe physique, nous nous intéresserons cette fois ci à la réalisation de celui-ci. Comment cela fonctionne-t-il en pratique dans le cas de Synchrotron-SOLEIL ?

① Cette marque désigne là où sont captés les électrons et accélérés

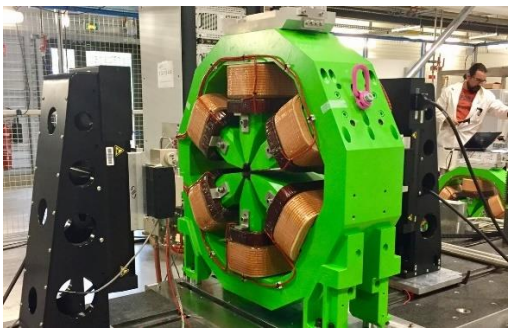
Dans un premier temps, il faut capter les paquets d'électrons pour ensuite les accélérer. Pour cela, plusieurs milliards d'entre eux sont arrachés à une pastille de métal (du tungstène) puis sont accélérés et rassemblés par paquets, de quelques centimètres de long, dans le LINAC (ou accélérateur linéaire). Au bout de cet accélérateur linéaire de 16 m de long, les électrons atteignent un premier niveau d'énergie : 100 MeV. Pour que les électrons soient accélérés, il ne doit pas y avoir d'obstacle sur leur parcours, tout au long de leur odyssée dans les accélérateurs de SOLEIL, ils circulent donc dans une chambre à vide. Il s'agit d'un tube de moins de 10 cm de diamètre dans lequel on maintient, grâce à différents dispositifs de pompage, un vide très poussé.

② Cette marque désigne là où l'énergie des électrons est augmentée jusqu'à 2.75 Gev

Après cette première accélération, le faisceau d'électrons est dirigé vers un deuxième accélérateur appelé **Booster** qui augmente leur énergie tour après tour, jusqu'à la valeur de fonctionnement de SOLEIL soit 2,75 GeV (2,75 milliards d'électronvolts). Pour accélérer, guider et focaliser les électrons, on utilise des champs électriques et des champs magnétiques produits par divers types de dispositifs. Les électrons sont accélérés en passant dans des cavités radiofréquences (RF), guidés en traversant des dipôles (ou aimants de courbure), et maintenus confinés dans un espace comparable à un cheveu grâce à des systèmes magnétiques appelés quadrupôles et sextupôles.



Titre : Image d'un quadrupôle



Titre : Image d'un sextupôle de l'ESRF à Grenoble

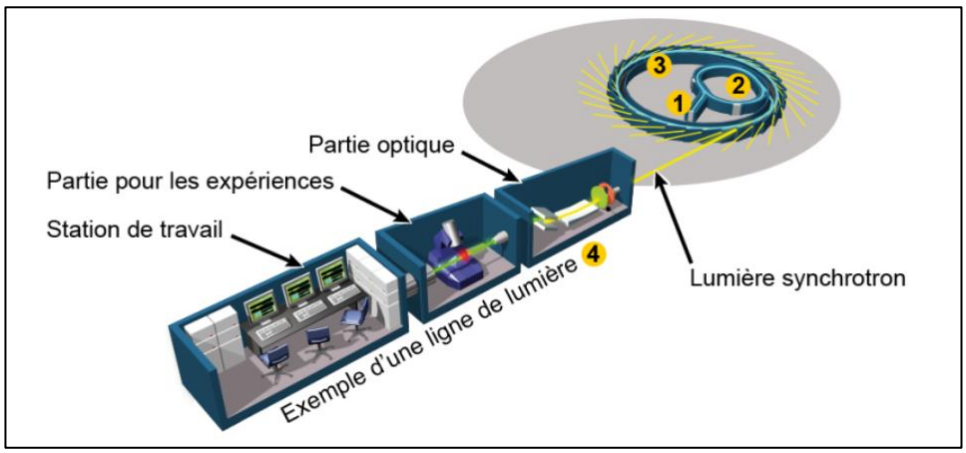
③ Cette marque désigne là où sont ensuite envoyés et stockés les électrons

Les électrons sont ensuite « injectés et accumulés » dans l'anneau de stockage de 354 mètres de circonférence, qui porte bien son nom, puisqu'on fait en sorte que les électrons y circulent le plus longtemps possible : on les stocke. En revanche, ce n'est pas un anneau au sens strict du terme, mais une sorte de polygone (à 32 côtés, soit un dotriacontagone !), où des sections courbes alternent avec des sections droites. À chaque virage, dans chacun des 32 aimants de courbure, l'électron est insensible au champ magnétique et se propage lui aussi dans une chambre à vide vers chacune des lignes de lumière installées autour de l'anneau. Entre deux dipôles, les électrons vont « globalement » tout droit, que ce soit dans les quadrupôles, les sextupôles ou les sections droites sans champ magnétique. Afin de produire encore plus de rayonnement synchrotron, on peut installer dans ces sections droites des « onduleurs » qui sont une succession de dipôles courts avec un champ magnétique alterné : les électrons qui les traversent font de nombreux petits virages en zigzag sur une courte distance et émettent à chaque fois de la lumière, avant de reprendre leur trajet dans l'anneau de stockage à la sortie de l'onduleur. Le nombre de photons émis est ainsi significativement augmenté comparé à celui émis lors de la traversée d'un dipôle. Après chaque tour d'anneau, puisqu'ils ont traversé des dipôles et des onduleurs, les électrons ont perdu une partie de leur énergie initiale. Si on ne fait rien, l'énergie diminue tour après tour, et les électrons finissent par être perdus. Il faut donc compenser leur perte d'énergie à chaque tour pour qu'ils continuent de circuler dans l'anneau de stockage. Grâce à une onde Radio Fréquence (onde RF) puissante, stockée dans une cavité accélératrice traversée par les électrons, ceux-ci regagnent leur énergie perdue et repartent pour un tour. Cette compensation n'est cependant pas suffisante pour garder le nombre d'électrons constant, car des électrons sont également perdus lors de collisions entre eux ou bien avec des atomes du gaz résiduel dans la chambre à vide. C'est pourquoi tout le processus d'accélération dans le LINAC et le Booster est redémarré environ toutes les 2 minutes mais seulement pour remplacer le nombre d'électrons perdus. C'est le « Top-up » qui permet de garantir la stabilité de l'intensité du faisceau d'électrons au cours du temps, essentielle pour les expériences.

④ Cette marque désigne là où se trouve la ligne de lumière :

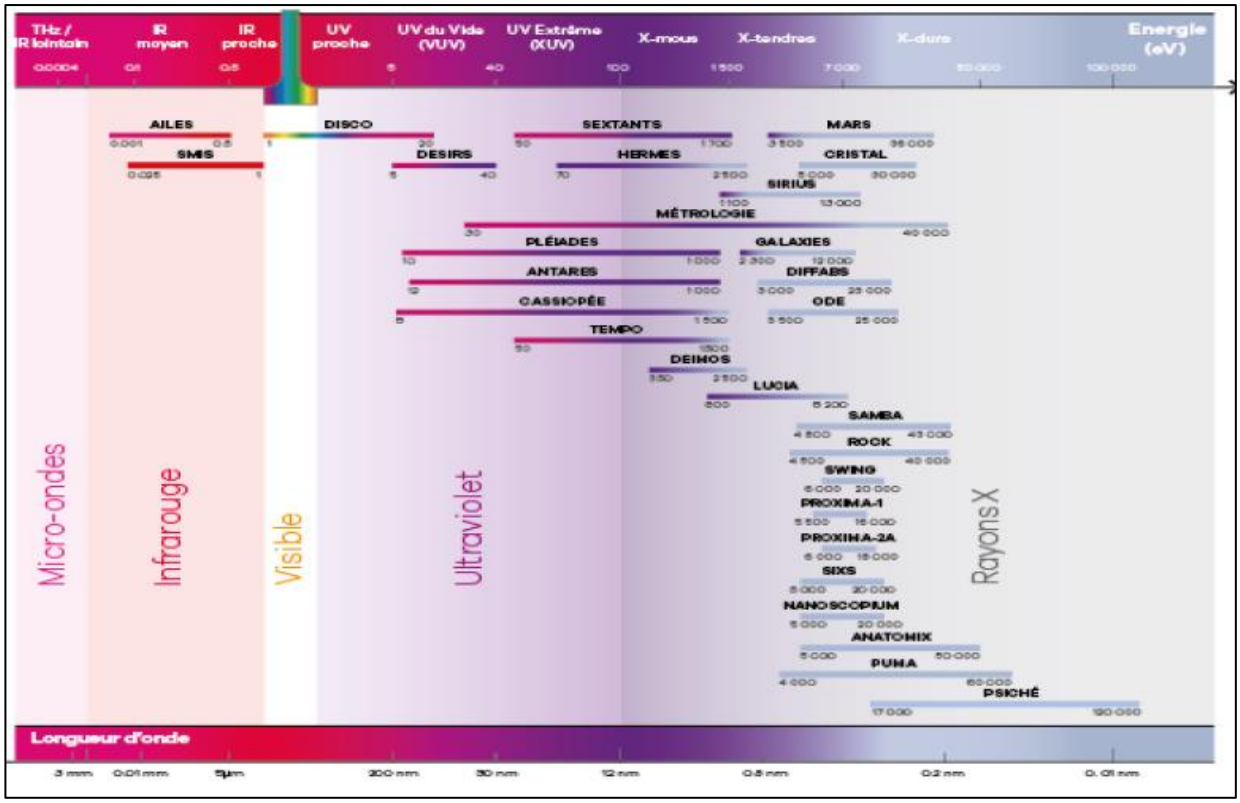
C'est un seul faisceau d'électrons qui était jusqu'à présent stocké dans l'anneau. C'est la déviation de ce paquet d'électrons qui génère le rayonnement, comme expliqué dans la partie précédente de l'avant-propos. Il y a au total 29 lignes de lumières. Ces lignes fonctionnent toutes simultanément. Chaque ligne de lumière se décompose en 3 parties.

1ere partie	Une partie du rayonnement est sélectionnée et conditionnée par des systèmes optiques
2e partie	Expériences d'analyse de la matière
3e partie	Zone de pilotage à distance des différents équipements



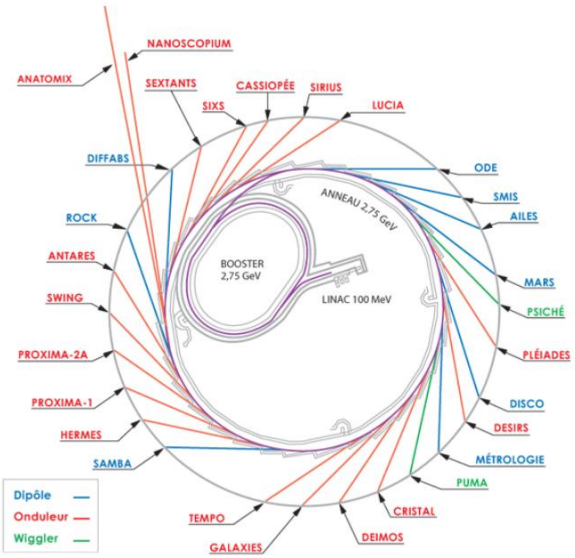
Titre : Schéma de principe d'une ligne de lumière avec ses différentes parties. Il s'agit d'une vue explosée de la partie 4 du schéma précédent

Le type de lumière exploitée par chaque ligne de lumière varie de l'une à l'autre, entre les infrarouges et les rayons X (voir schéma). Chaque ligne de lumière constitue un véritable laboratoire de biologie, ou de chimie, ou de sciences de la Terre, ..., instrumentée pour préparer et analyser les échantillons à étudier à l'aide du rayonnement, et traiter les informations recueillies …
Le bon fonctionnement des accélérateurs et des lignes de lumière s'appuie sur tout un ensemble d'équipes aux compétences variées : électronique, informatique, détecteurs, optique, vide, magnétisme, préparation et caractérisation des échantillons, …



Titre : Répartition du domaine d'étude des 29 lignes de lumières sur le spectre. Notons que la majorité des lignes se situent dans le domaine des rayons X.

2.2 Localisation des lignes



Titre : Plan de l'organisation des lignes de lumières au sein de l'anneau.

Les lignes de lumière sont situées à l'étage 0 du synchrotron. Il y a au total 29 lignes de lumières. Chacune ayant ses spécificités et son domaine d'étude. Le synchrotron se scinde en deux étages, l'un autour de la machine et l'autre au niveau des bureaux. Ceux-ci sont dédiés aux divers groupes constituant SOLEIL. Notons que SOLEIL possède des laboratoires d'analyses, tels qu'un laboratoire d'analyse chimique. Ils sont situés sur la bordure de l'anneau.

Partie 1 : Pourquoi ?

Sommaire

1. **Qu'est-ce que Synchrotron SOLEIL en bref et quelle est sa position au sein de la recherche mondiale ?**

    1.1 Synchrotron SOLEIL en Bref ?

    1.2 Quels types de recherches sont faites à SOLEIL ?

    1.3 L'infrastructure du calculateur

    1.4 Comment sont analysées les données ?

    1.5 Le projet SOLEIL II

2. **Le stage**

    2.1 La raison de ce stage et mon projet professionnels

    2.2 Quelle est ma mission ?

    2.3 Dans quelle mesure cela s'inscrit -il dans le cadre de ma formation à l'ESIEE-PARIS ?

3. **Fonctionnement Interne de l'entreprise**

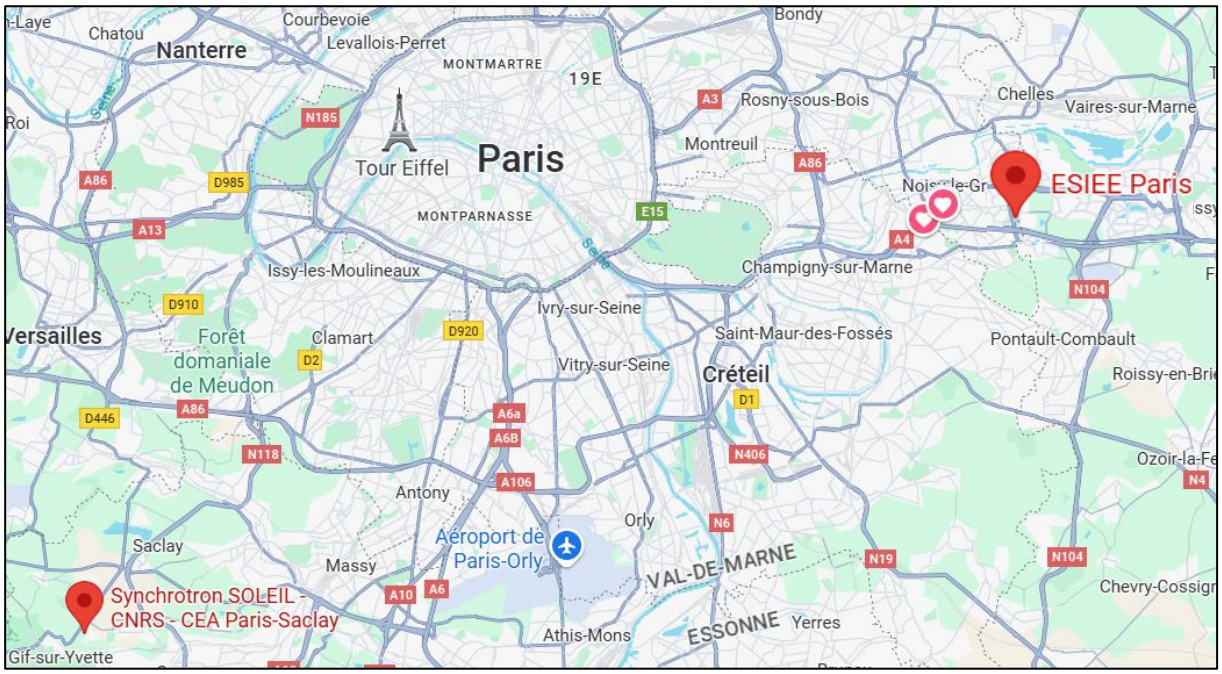


2. Qu'est-ce que Synchrotron SOLEIL en bref et quelle est sa position au sein de la recherche mondiale ?



Titre : Image de l'intérieur de l'anneau où se situe le synchrotron, on voit au 1er plan des cabines de lignes de lumières vues du dessus

1.1 Synchrotron SOLEIL en Bref ?



Crédit : Google Maps

Le laboratoire de recherche Synchrotron SOLEIL est un accélérateur d'électrons, situé sur le plateau de Saclay. Il s'agit d'un synchrotron de 3e génération optimisé dans la gamme des rayons X moyens. Créé en 2006 sous le mandat de Jacques Chirac, il appartient conjointement au CNRS et au CEA, dont ils détiennent respectivement 72% et 28% des parts Cela n'en fait pas pour autant un organisme de l'un ou de l'autre, Synchrotron-Soleil est propre à lui-même et fonctionne comme une entité autonome. Une entreprise composée exactement de 358 scientifiques et administratifs. SOLEIL est lui-même un sous-traitant pour des projets de R&D, néanmoins il en possède également. Les fournisseurs de SOLEIL sont

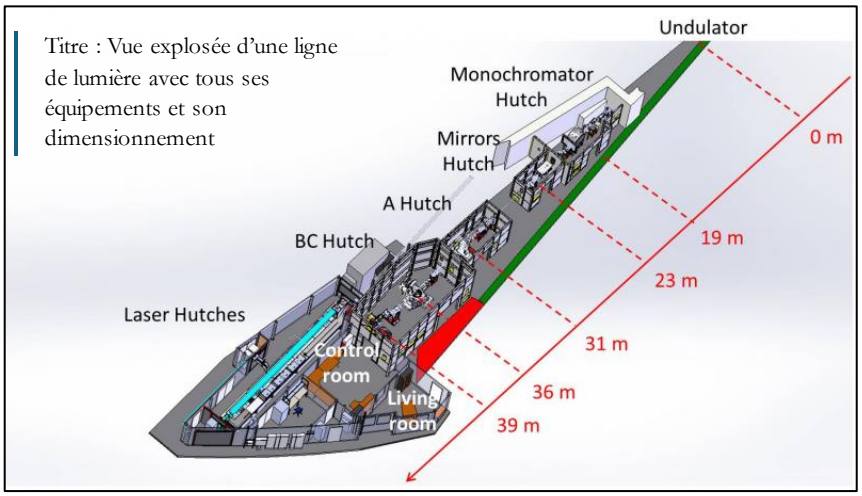
des entreprises françaises spécialisées dans les technologies de pointe, cependant, la liste exhaustive est confidentielle et très difficile à avoir. L'acronyme SOLEIL signifie « Source optimisée de lumière d'énergie intermédiaire du LURE. Le principe de l'accélérateur ayant déjà été évoqué, afin d'éviter toute redite, je suggère de se référer à l'avant-propos. La force de SOLEIL réside dans son domaine d'étude. Beaucoup d'accélérateurs se concentrent sur les hautes et moyennes énergies. Cependant, tout l'enjeu de SOLEIL est de traiter avec précision le cas des basses énergies. Nous avons en France deux grands labos traitant ce sujet, l'ESRF (European Synchrotron Radiation Facility) situé à Grenoble qui utilise un synchrotron de 4e génération (upgrade inauguré en 2018) où sont accélérés des électrons selon le même principe qu'à SOLEIL , à l'aide d'un anneau de 300m de circonférence. Le domaine d'étude va jusqu'à 6Gev [6] . Les chercheurs utilisent le faisceau quasi exclusivement pour des rayons X durs. Ce qui n'est pas le cas de SOLEIL où cette étude, est très en marge de leur domaine. En réalité, le spectre d'études à l'échelle française est assez bien reparti. Personne n'est en compétition avec ses collègues. Synchrotron-SOLEIL étudie les gammes de fréquences allant jusqu'au rayon X durs au maximum. Tandis qu'ESRF les étudie de manière beaucoup plus complète et poussée en raison de sa gamme d'énergie quasiment deux fois supérieure à SOLEIL , 6 Gev. SOLEIL fait également ce type d'étude mais avec des fréquences plus basses, comme le montre le spectre (Avant-propos 2.1 , p22). A quoi sert ces analyses sur différentes fréquences me diriez-vous…

ESRF
The European Synchrotron

1.2 Quels types de recherches sont faites à SOLEIL ?

Synchrotron SOLEIL ne se réduit pas à son synchrotron, ni à ses lignes de lumières. Chaque ligne a une raison d'être, un domaine d'étude précis. Tout d'abord, il faut distinguer la recherche qui est faite en interne, les collaborations et projets de recherche. SOLEIL est lui-même un sous-traitant pour des projets de R&D, néanmoins il en possède également. SOLEIL permet également à des entreprises extérieures de mener leurs projets de recherche et d'innovation industrielle, et c'est d'ailleurs aussi le cas pour les doctorants et post-doc sur les domaines scientifiques et techniques en lien avec les capacités d'études des lignes. Ainsi, cela permet l'étude des domaines des matériaux avancés, la biologie et la santé, la phase gaz, l'astrophysique, les structures électroniques et moléculaires, la photo dynamique, les interfaces solides/gaz/liquides, la matière condensée, la matière sous conditions extrêmes, les nano-objets, mais aussi l'instrumentation, la physique des accélérateurs de particules... Tous sont autant de disciplines qui peuvent être étudiées grâce à un outil comme SOLEIL. Ce sont les utilisateurs qui conduisent ces recherches, mais également le personnel scientifique, au travers de la recherche interne.

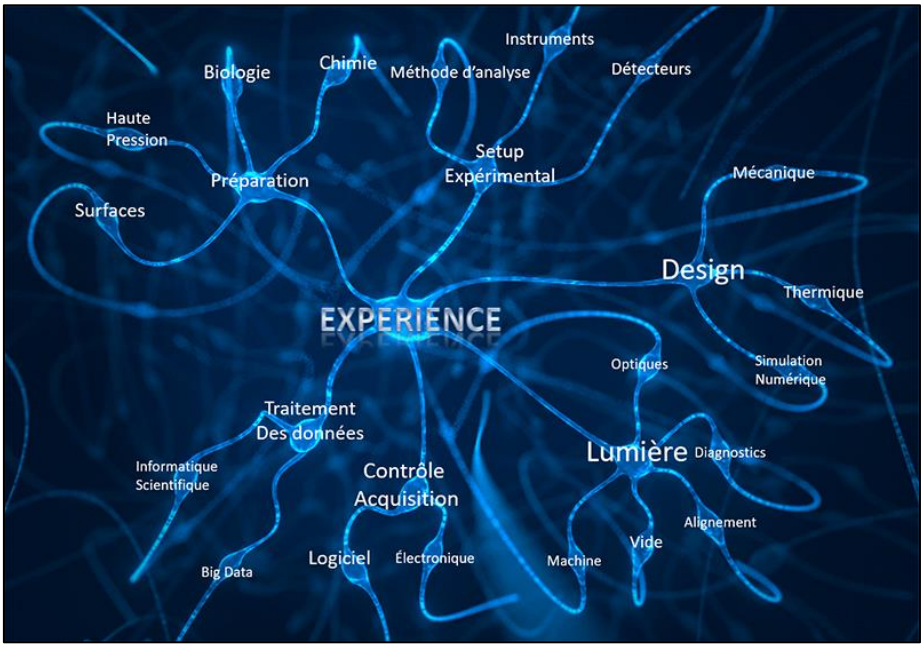




Titre : Vue explosée d'une ligne de lumière avec tous ses équipements et son dimensionnement

L'étude de la biologie, santé, est au cœur de la recherche de SOLEIL. Ces recherches se font notamment via la fluorescence par imagerie aux rayons X qui permet de localiser des atomes et des composés spécifiques dans les tissus animaux et végétaux; l'imagerie et la spectroscopie UV sans marquage permettent de suivre le devenir de composés chimiques dans les cellules vivantes; les expériences de tomographie par rayons X permettent de sonder la structure interne microscopique des échantillons macroscopiques; la diffraction et la diffusion des rayons X fournissent des informations structurelles sur les macromolécules à une résolution atomique.

La section MMI (Méthodes, Instrumentation et Modélisation) assure l'animation scientifique parmi les scientifiques, ingénieurs et techniciens des différents groupes support de SOLEIL.



Titre : Carte mentale de ce que réalise la section MMI au sein de SOLEIL pour permettre la réalisation d'une expérience au sein de SOLEIL

SOLEIL permet également l'étude de la physique atomique et moléculaire, de la science de l'univers. Au sein de la section scientifique « Matière Diluée », les objets étudiés sont des atomes, des molécules, des agrégats, des complexes non-covalents et des nanoparticules qui se trouvent dans un environnement sans solvant et sans substrat. Parce que les collisions intermoléculaires et solvant-molécule peuvent être négligées, ce régime de "matière diluée" offre, par rapport à la matière condensée, l'opportunité d'étudier des systèmes modèles ou réels au niveau élémentaire. Par ailleurs, la matière diluée est optiquement mince, elle peut donc être étudiée en continu sur tout le spectre, et en particulier dans la gamme spectrale couverte par les lignes de lumière de SOLEIL, de l'IR lointain aux rayons X tendres.

DOMAINES DE RECHERCHE	TECHNIQUES D'ANALYSE
Phase gaz • Astrophysique • Photodynamique • Structure électronique et moléculaire Biologie Chimie	Absorption • Spectroscopie IR et THz • Spectroscopie VUV Photo-emission d'électron • Spectroscopie de photo-émission résolue en angle (ARPES) • Spectroscopie de photo-électron et photo-ion en coïncidence (PEPICO) • Études en temps résolu Spectroscopie d'ions • Spectrométrie de masse Diffusion

Domaine scientifique de **l'étude de la physique atomique et moléculaire, de la science de l'univers**

CONDENSED MATTER RESEARCH	Nanoparticles, nano films, nano objects Soft matter / liquid Interfaces (solids/ gas/ liquid) Organised condensed matter (solids) Biomaterial
MAIN TECHNIQUES	IR Spectroscopy (Absorption) Spectroscopy Photoemission (ARPES) (Photoelectron emission) X-ray spectroscopy XAS/XES (Imaging) Diffraction / Diffusion

SOLEIL permet également la recherche pour la section de la matière condensée (chimie et chimie physique), basée sur l'utilisation de plusieurs sondes basées sur le rayonnement synchrotron : spectroscopies de diffraction, d'absorption et de photoémission, pour la description et la compréhension d'une variété de systèmes et de leurs changements dans des conditions bien contrôlées.

LISTE DES COLLABORATIONS ET PROJETS EN COURS	
AIQuAM3D	NEP
CLEXM	NEPHEWS
Destiny	OSTrails
I.FAST	Remade@ARI
IMPRESS	RIANA
LEAPS	SUNSTONE
LEAPS INNOV	UPSaclay STAR φ
	UPWEARS

Il y a également bien d'autres domaines d'études. SOLEIL est un environnement international, pour ces raisons, il entretient des relations fortes avec des entreprises et laboratoires du monde entier. Je pense notamment à l'ESRF au niveau régional, mais également aux autres accélérateurs de particules du monde et européens. La Commission Européenne (CE) finance des projets individuels, collaboratifs et de formation d'excellence et à fort impact scientifique et/ou sociétal. Depuis 2010, SOLEIL a participé à 39 projets collaboratifs de l'UE. SOLEIL est aussi membre du consortium stratégique européen des infrastructures de rayonnement synchrotron et de lasers à électrons libres (LEAPS - the League of European Accelerator-based Photon Sources). SOLEIL signe également des accords-cadres avec d'autres organismes de recherche internationaux. Ces accords constituent la base de futures collaborations scientifiques.

L'Agence Nationale de la Recherche (ANR) lance chaque année plusieurs appels couvrant les thématiques de recherches du Synchrotron SOLEIL. Depuis 2010, 85 propositions ont été financées. SOLEIL a également reçu des financements via les appels LabEx, Equipex, IdeX... du Plan pour les Investissements d'Avenir (PIA). Plusieurs subventions de la région Ile-de-France ont été obtenues, parmi lesquelles on peut citer les projets DIM et SESAME. Pour faciliter et accélérer la mise en œuvre de projets de recherche communs entre les équipes du Synchrotron SOLEIL et d'autres organismes de recherche français, SOLEIL signe des accords-cadres qui constituent la base de futures collaborations scientifiques.

LISTE DES COLLABORATIONS ET PROJETS EN COURS	
B-BEST	MaTerRE
Batteries	Nano-Saclay
CNRS	PALM
DIADEM	PAMIR
IFPEN	QuanTiP
INRAE	SPIN - Spintronique
LUMA	TASE

SOLEIL s'est engagé dans une démarche de mise en place de plateformes de recherche et de services spécialisées par domaine applicatif, pour l'hébergement d'équipes externes, publiques et/ou privées, menant des projets synchrotron à moyen-long terme.

INRA-SOLEIL	IPANEMA, matériaux anciens
NANOIMAGESX, imagerie X 3D à très haute résolution	MICASOL, Matériaux & Nanotechnologies
Cosmetomics, Cosmétique	/

SOLEIL associe les entreprises de l'instrumentation scientifique, majoritairement des PME françaises, à sa recherche technologique, en amont dans le cadre de projets collaboratifs et en aval par la concession de licences permettant l'exploitation industrielle et commerciale de dispositifs innovants conçus par ses équipes. Cela sera développé dans la 3e section de cette partie, lors du traitement de la structure internet de l'entreprise SOLEIL. Néanmoins, il est aisé de faire des expériences, l'enjeu est désormais d'en tirer des conclusions et d'analyser les résultats récoltés par les différents dispositifs, capteurs, etc. Cela passe essentiellement par des logiciels de traitement et autres outils informatiques présents à SOLEIL.

1.3 Comment sont analysées les données ?

Les scientifiques des lignes de lumière traitent leurs données dans le but de modéliser la composition des échantillons graphiquement, de faire la tomographie, de faire des figures de Braag, de révéler des éléments dans le cas d'échantillons à portée archéologique. Cela peut aussi être l'objet de vérification d'un modèle, d'une théorie, ou de modéliser les résultats comme énoncé plus haut. Cependant, pour analyser, modéliser, tracer, il faut des logiciels adaptés à cela. Ce sont des logiciels spécifiques pour le traitement des données issues de ce type d'installation. Pour traiter notamment les données et en faire des figures de diffraction, de fluorescence d'atomes, etc.

Pour l'analyse de données scientifiques, Emmanuel FARHI a créé il y a des années de cela, avec des outils rudimentaires et avec un noyau un peu ancien, la machine virtuelle DARTS. Elle dispose d'un



environnement Linux avec des moteurs de recherches et des API pour l'analyse de données. Il dispose d'outilS supportant CUDA, OpenCL, OpenACC pour la partie GPU.

Plusieurs supports sont disponibles selon les versions de Debian, Debian 11 (BullEye) et Debian 12 (Bookworm) . Il possède un nombre variable de CPU allouable allant de 4 à 64. Et disposant respectivement jusqu'à 256 Gb de RAM. Cela en fait une machine virtuelle complète et utilisable pour l'analyse de données et pour tous les utilisateurs de SOLEIL. En réalité, l'un des seuls défauts que l'on pourrait lui attribuer est que le déplacement de fenêtre à l'aide du curseur lag un petit peu.



Par ailleurs, les équipes de synchrotron SOLEIL souhaitent effectuer une transition de DARTS vers VISA. VISA (Virtual Infrastructure for Scientific Analysis) est une plateforme qui simplifie la création d'instances de calcul sur une infrastructure OpenStack pour permettre aux utilisateurs scientifiques d'analyser leurs données expérimentales à distance à l'aide d'un simple navigateur Web. Ce sont deux visions différentes entre VISA et DARTS. VISA a été entièrement développé dans le contexte de PaNOSC (Photon an Neutron Open Science Cloud), ce qui signifie que cette fonctionnalité est fournie de manière paneuropéenne à tous les scientifiques. L'objectif est de permettre aux instituts d'adapter VISA à leurs infrastructures et de fournir une solution d'analyse des données pour les scientifiques afin qu'ils aient un accès direct à leurs données expérimentales, aux outils nécessaires pour les analyser et permettre des collaborations en temps réel. Avec VISA, les instituts PaN travaillent ensemble pour répondre à un besoin réel de la communauté. VISA visait à apporter une solution à ce problème et à proposer l'Analyse des données en tant que service dans le cloud. [7]

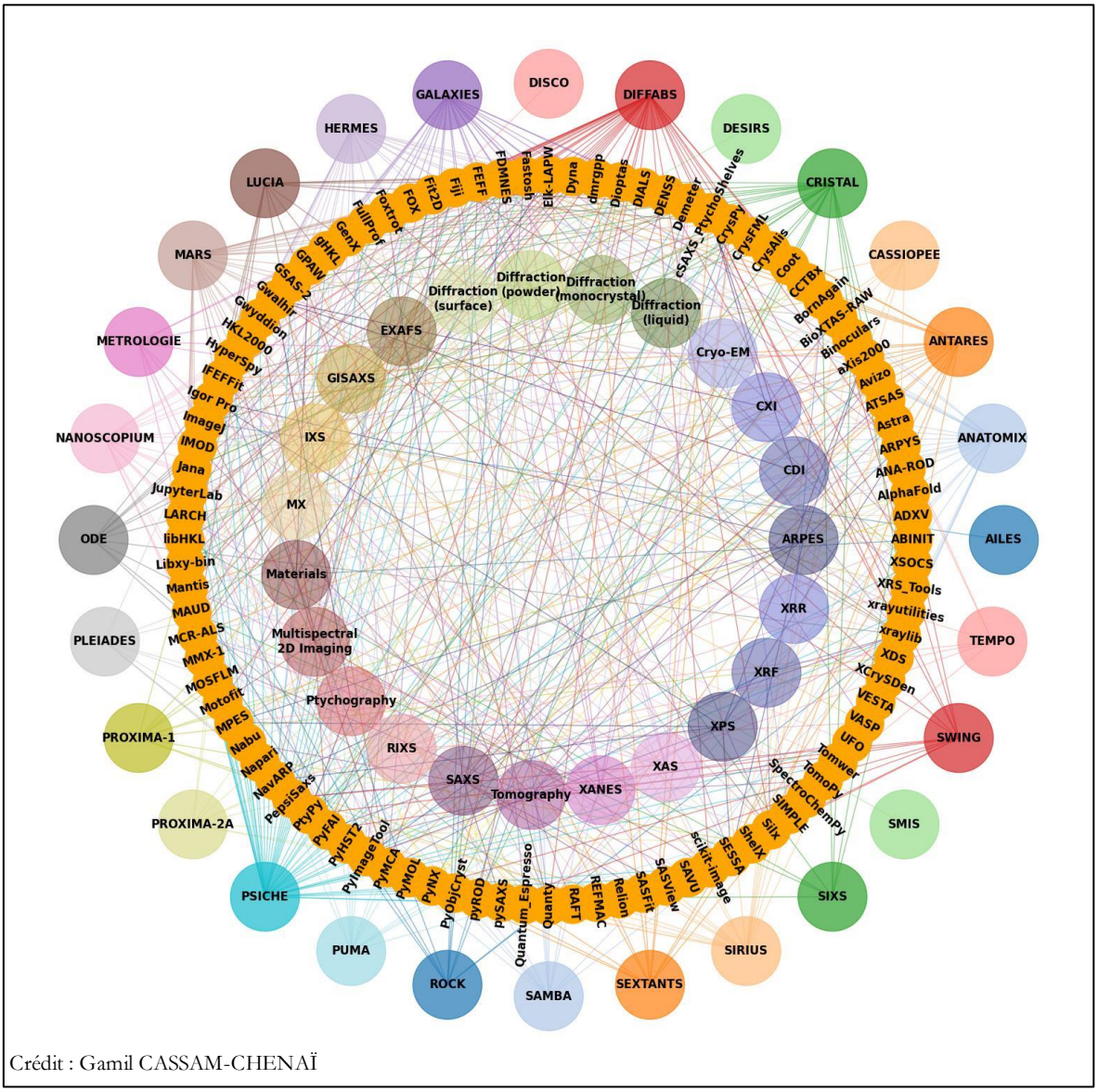
De plus, l'utilisation de notebooks est possible via l'utilisation de Jupiter qui est installé en local. Il y a également à disposition des supports pour le calcul de modélisation des matériaux (estimation des dispersions électroniques et des phonons via DFT/ab-inito, p. ex. fichier CIF requis). Des ensembles de données sont disponibles pour les calculs précédents et suivants : diffraction des poudres/Fluorescence (les estimations, nécessitent un fichier CIF ou une formule chimique). SOLEIL développe également des logiciels scientifiques en interne. En particulier les logiciels Binoculars/libHKL, McXtrace, DARTS, et contribue activement au déploiement de logiciels scientifiques (Debian/Ubuntu). Ces logiciels utilisent des transformées de Fourier, et autres outils mathématiques. Ces logiciels sont construits à partir de paquets Debian, utilisés pour les calculs demandés. Le paquet Pynx, est un outil Python pour la cristallographie des nanostructures (Scripts), et est utilisé dans des logiciels pour faire les calculs dédiés. Les paquets constituent les briques de ces logiciels. Il est donc important et vital de les maintenir pour le passage de Debian aux futures versions. Beaucoup de logiciels utilisent des CPU et des GPU pour accélérer l'analyse de données et la performance des paquets. Aujourd'hui, nous utilisons des GPU et des CPU à Synchrotron-SOLEIL, pour booster les performances et rendre les calculs bien plus rapides. Les données sont issues de la « ruche », c'est ici où sont stockées les données générées par les lignes de lumières en vue de leurs traitements. Alors, dans la continuité de l'analyse des données scientifiques, il devient vital de vérifier la compatibilité des paquets vis-à-vis des architectures en place, amd64, arm , i386 etc et des supports de cartes graphiques. CUDA pour Nvidia et ROCm pour AMD. Cela représente donc un enjeu majeur et a directement un lien avec ma mission, dont je développerai la nature et les enjeux moraux et scientifiques dans la section numéro 2 de cette partie.

Gamil CASSAM-CHENAÏ a réalisé un inventaire des logiciels utilisés par les lignes de lumières à SOLEIL. A l'aide d'un script, il a pu générer un arbre des dépendances entre l'ensemble des logiciels, les paquets et les lignes de lumières. Dont voici le lien vers le git salsa de Synchrotron-SOLEIL.

https://gitlab.com/soleil-data-treatment/soleil-software-projects/sas-draw.git

Titre : Tableau de l'ensemble des paquets et API utilisés à SOLEIL pour l'analyse, la modélisation des données scientifiques.

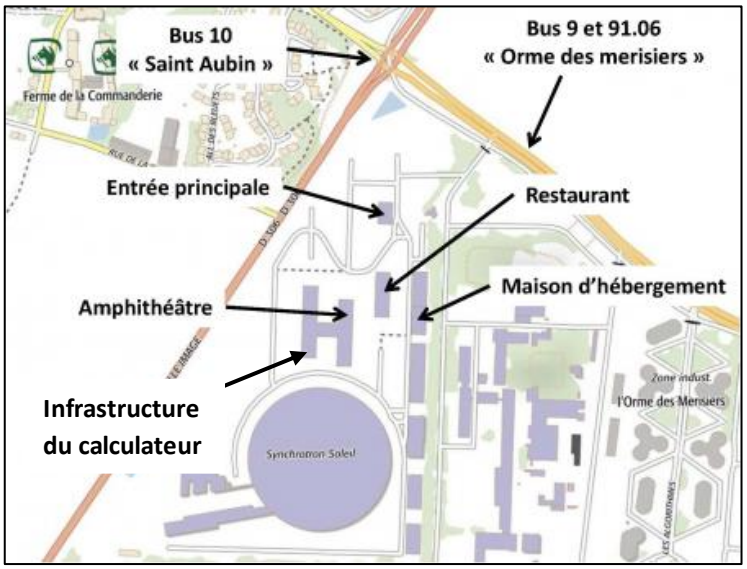
ABINIT	DIALS	HyperSpy	PepsiSaxs	SESSA
ADXV	Dioptas	IFEFFit	PtyPy	SheIX
AlphaFold	dmrgpp	Igor Pro	PyFAI	Silx
ANA-ROD	Dyna	ImageJ	PyHST2	SIMPLE
ARPYS	Elk-LAPW	IMOD	PyImageTool	SpectroChemPy
Astra	Fastosh	Jana	PyMCA	TomoPy
ATSAS	FDMNES	JupyterLab	PyMOL	Tomwer
Avizo	FEFF	LARCH	PyNX	UFO
aXis2000	Fiji	libHKL	PyObjCryst	VASP
Binoculars	Fit2D	Libxy-bin	pyROD	VESTA
BioXTAS-RAW	FOX	Mantis	pySAXS	XCrySDen
BornAgain	Foxtrot	MAUD	Quantum_Espresso	XDS
CCTBx	FullProf	MCR-ALS	Quanty	xraylib
Coot	GenX	MMX-1	RAFT	xrayutilities
CrysAlis	gHKL	MOSFLM	REFMAC	XRS_Tools
CrysFML	GPAW	Motofit	Relion	XSOCS
CrysPy	GSAS-2	MPES	SASFit	
cSAXS_PtychoShelves	Gwalhir	Nabu	SASView	
Demeter	Gwyddion	Napari	SAVU	
DENSS	HKL2000	NavARP	scikit-image	



Crédit : Gamil CASSAM-CHENAÏ

Titre : Graph des dépendances entre les lignes de lumières (Bordure), les paquets Debian utilisés (Orange), et les techniques scientifiques d'analyses utilisés à Synchrotron-SOLEIL (Au centre)

1.4 L'infrastructure du calculateur



Titre : Carte de Synchrotron SOLEIL

Le calculateur est situé dans le bâtiment principal. Il abrite l'ensemble des données de SOLEIL. Au total c'est 4 Pétabits de stockage, cela correspond à 4000 Téra octets, 4 000 000 Go. Les données des expériences et l'ensemble des données sont dupliquées pour pallier une attaque ou un disfonctionnement. L'acquisition et les transferts des données vers le calculateur se font à l'aide de centaines de fibres optiques et d'une grande infrastructure réseau. Pour les lignes de lumières ayant un grand flux de données, il y a plusieurs fibres de 100 Go/s et une immense majorité de fibres en 10 Go/s. Le calculateur est composé de CPU et GPU. Ces derniers sont rangés dans les raques adaptés, au total environ 5 cartes graphiques NVIDIA, des RTX 2080 Ti et autres cartes NVIDIA ainsi qu'une carte AMD, une gfx1034, AMD Radeon RX 6400, la carte testée. Ce ne sont pas moins de 255 Go de RAM disponibles pour une machine disposant de 64 cœurs et 4 Pétabits de stockage. La ruche, les données etc , tout est sur cette infrastructure. Ce faisant, les logiciels précédemment énoncés utilisent les ressources de ce calculateur. Il est donc vital, de vérifier la compatibilité des paquets utilisés par les logiciels sur les cartes graphiques utilisées. La phase de test de la carte AMD s'inscrit dans une volonté de détachement de NVIDIA. Qui vient d'ailleurs de dépasser il y a quelques mois les 1000 milliards de valuations, ce qui en fait la première entreprise à dépasser cette barre symbolique.

1.5 Le projet SOLEIL II

Le projet de mise à jour du Synchrotron Soleil a été initié en 2020 avec de multiples objectifs comme l'amélioration de ses performances, une plus grande rapidité des expériences ou une empreinte écologique diminuée. Le démarrage des accélérateurs de SOLEIL II est prévu pour 2030 , avec une montée en puissance jusqu'en fin 2035 où il fonctionnera à pleine puissance. Le projet Upgrade marque une étape majeure dans l'histoire de SOLEIL : il s'agit de transformer l'installation actuelle en SOLEIL II, une version modernisée pensée pour rester à la pointe des synchrotrons de nouvelle génération. Cette transformation ne répond pas seulement à une ambition technique ; elle reflète un choix stratégique à long terme pour répondre aux besoins scientifiques croissants des chercheurs et préserver la compétitivité internationale de l'infrastructure. Sur le plan scientifique, l'objectif est clair : offrir un faisceau de meilleure qualité, grâce à une nouvelle conception de la maille de la machine. Les améliorations attendues en brillance et en cohérence spatiale permettront de réaliser des expériences plus précises, plus rapides et plus sensibles. Certaines recherches aujourd'hui limitées par les performances de l'anneau actuel pourront ainsi se déployer pleinement. Sur le plan stratégique, l'upgrade vise à assurer la pérennité opérationnelle de SOLEIL pour les vingt à trente prochaines années. Pour y parvenir, le projet intègre dès la conception des technologies modernes et évolutives, tout en réduisant les risques techniques grâce à des phases préparatoires de prototypage et de conception détaillée. Enfin, la mise en œuvre est pensée de façon progressive, afin de limiter autant que possible l'impact sur les activités scientifiques en cours.

La transformation s'organise en deux grandes phases. La première, de 2024 à 2028, correspond à la phase de construction : installation de la nouvelle machine et relocalisation ou modernisation initiale de certaines

lignes de lumière. La seconde, de 2029 à 2033, sera dédiée à la montée en puissance et à l'atteinte des performances nominales de SOLEIL II. Sur le plan technique, le projet est structuré en programmes thématiques, chacun divisé en sous-projets clairement définis.

Accélérateurs	Conception, modélisation, prototypage et validation des nouveaux composants de l'anneau de stockage et de l'injecteur.
Infrastructures	Adaptation et extension des bâtiments, réseaux et installations techniques.
Lignes de lumière et laboratoires (BL2)	Réorganisation de l'implantation, relocalisation de lignes pour s'adapter à la nouvelle géométrie et modernisation instrumentale.
Systèmes d'Information et Traitement des Données	Développement logiciels, gestion des données scientifiques et adaptation des infrastructures informatiques aux nouveaux besoins.

Chaque sous-projet est dirigé par un chef de projet, responsable des livrables, du respect du budget et du calendrier, ainsi que de la gestion des interfaces techniques. Deux outils clés soutiennent cette organisation : ORCHESTRA pour le suivi du planning et des ressources, et MERIDIAN pour la gestion documentaire centralisée. L'organisation humaine s'appuie sur une gouvernance claire. Le directeur général reste l'autorité finale, arbitrant les priorités et validant budgets et objectifs. Le Comité de Pilotage (COPIL), composé des responsables de programmes, de la direction administrative et financière, et d'experts invités, valide les décisions techniques majeures. La Coordination des Programmes et Planification (CPP) joue un rôle transversal, assurant le lien entre programmes et garantissant la cohérence technique et scientifique. La coordination repose sur un rythme régulier de réunions : échanges internes dans chaque programme, réunions bimensuelles CPP et COPIL, ateliers thématiques et revues critiques sur les lignes de lumières. La sécurité, la qualité et le soutien administratif sont intégrés dès la phase TDR. Une gestion des risques structurée est mise en place, centralisée au niveau du COPIL, avec suivi et mises à jour régulières. Le programme des lignes de lumières constitue un chantier particulièrement délicat, car il touche directement à la capacité expérimentale de SOLEIL. Pendant la première phase (2024-2028), six lignes seront relocalisées pour s'adapter à la nouvelle géométrie, tandis que quatre seront modernisées en priorité pour exploiter pleinement les capacités de SOLEIL II. Les 23 autres lignes seront réalignées et adaptées au nouveau faisceau. La seconde phase (2029-2033) verra la modernisation de 19 lignes supplémentaires, pour que l'ensemble du parc bénéficie des nouvelles performances. Afin d'assurer la cohérence scientifique, les lignes sont regroupées par thématiques de recherche. Des ateliers réunissant scientifiques, ingénieurs et techniciens permettent de coordonner les développements instrumentaux, notamment dans l'optique, la détection et le traitement des données, en mutualisant les efforts entre plusieurs lignes. La métamorphose de SOLEIL en SOLEIL II dépasse largement le cadre d'un simple remplacement technique : c'est une refonte complète qui engage l'ensemble de l'infrastructure, de ses équipes et de ses méthodes de travail. Ce projet doit conjuguer innovation et continuité, tout en maîtrisant les risques et en préservant la capacité de recherche pendant la transition. Avec une planification en deux étapes, une gouvernance structurée et des outils de pilotage robustes, SOLEIL II se prépare à offrir à la communauté scientifique un instrument capable de relever les défis des décennies à venir.

2. Le stage

2.1 La raison de ce stage et mon projet professionnels

Ce stage est pour moi le début de l'accomplissement d'un rêve, travailler dans un synchrotron et notamment travailler au CERN en tant que Datascientist ou encore en tant que chercheur en physique fondamentale. Mon projet professionnel se construit autour de mon école d'ingénieurs ESIEE-PARIS, de

mes passions et de mes ambitions pour le futur. Bercé depuis petit par l'astronomie et l'univers, j'ai découvert la physique des particules grâce à l'annonce de la découverte du boson de Higgs en juillet 2012. Cela a été pour moi une véritable révélation et mon appétence s'est accrue au fil des années. C'est cette passion qui m'a poussé à postuler au laboratoire Synchrotron-Soleil pour un stage de 4 mois dans la division Div/Exp GRADES de Synchrotron-Soleil afin de réaliser des tests d'intégration de paquets sur Salsa. En effet, du fait de mes appétences pour le domaine de la physique fondamentale, j'ai choisi de faire une classe préparatoire aux grandes écoles en filière PC, ce qui m'a ensuite conduit à entamer une nouvelle page de mon parcours dans les études supérieures, celle d'intégrer l'école d'ingénieurs ESIEE-PARIS, sur le campus de Champs-sur-Marne. J'ai pu être formé sur le code en Python de manière avancée ainsi que le langage C, j'ai pu également réaliser des projets de Deep Learning et suivre des cours de Data sciences. Pour suivre cette passion qui m'anime depuis tout petit, j'ai décidé de réaliser un stage facultatif de E3 à Synchrotron-SOLEIL. Un centre de recherche exceptionnel avec des collègues aussi passionnés que passionnants et travailler sur des problématiques ayant un grand enjeu scientifique. J'apporte ma brique à l'édifice. Au sein de ESIEE-PARIS, j'ai d'ailleurs pour ambition de prendre la majeure DSIA (Data science & AI) dans le but de me former au mieux dans ce monde qui se façonne à vue d'œil. Je souhaite de réaliser des modèles IA et Machine Learning dans le but de modéliser au mieux les phénomènes physiques qui se passe au sein d'un accélérateur de particules ou encore d'implémenter des modèles prédictifs selon les caractéristiques des données recueillies. Ceci impose deux choses : l'une avoir une connaissance profonde du Machine Learning et de l'apprentissage automatique, de l'IA et de la Datascience. Ce qui correspond pour le moment à la formation que je suis à ESIEE-PARIS, grande école d'ingénieur dans le domaine. L'autre, d'avoir une connaissance parfaite et poussée de la physique des accélérateurs et de la physique fondamentale lorsqu'il s'agit de travailler à de si hautes énergies. On ne peut construire un bon modèle prédictif sans connaître les fondements de ce que l'on cherche, quand bien même serait-on un expert du Machine Learning et de l'IA. Sans connaissances parfaites dans le domaine que l'on traite il est impossible de le réaliser et surtout que celui-ci soit d'une grande qualité. Pour cela, je souhaite réaliser un double diplôme avec l'université italienne Politechnico di Milano dans le but d'accroître mon employabilité et d'acquérir des connaissances indispensables à la réalisation de mon objectif. Puis, afin de compléter ma formation, je souhaite réaliser un master de physique fondamentale et me spécialiser dans le domaine de la physique des particules à Paris-Saclay. En faisant cela, j'ai les éléments indispensables à la réalisation de mon projet professionnel. Cependant, dans le cas où ce premier projet ne fonctionne pas, j'ai pour ambition d'aller travailler chez NVIDIA. La situation géographique de Synchrotron-SOLEIL n'a pas influencé mon choix. Ainsi, ce stage n'est que le début de l'aventure, j'en ai bien conscience, ceci est la première pierre pour atteindre ou toucher du doigt ce rêve de pouvoir un jour travailler dans le plus grand accélérateur de particules du monde.

2.2 Quelle est ma mission et ses enjeux ?

Je travaille dans le département GRADES, Div/Exp, partie violette de l'organigramme, sous la direction de M. Emmanuel FARHI et M. Frédéric-Emmanuel PICCA. GRADES est une équipe d'une dizaine de personnes incluant les stagiaires et post-doc. ROCm™ est la plateforme logicielle open source d'AMD pour le calcul haute performance accéléré par GPU et l'apprentissage automatique. Ma tâche consiste à écrire des scripts exécutants les autopkgtest Opencl et à les exécuter sur le CI ROCm, dont les résultats seront utilisés pour valider les modifications du code source du repository du Git Salsa Debian du paquet. OpenCL (Open Computing Language) est la combinaison d'une API et d'un langage de programmation dérivé du C, proposé comme un standard ouvert par le Khronos Group. OpenCL est conçu pour programmer des systèmes parallèles hétérogènes comprenant par exemple à la fois un CPU multi-cœur et un GPU. Les scripts sont écrits en bash. Bash consiste notamment à automatiser les tâches, des simples manipulations de fichiers aux processus de déploiement complexes.

Le projet Debian est une association de personnes à l'échelle mondiale qui ont pour cause commune de créer un système d'exploitation libre. L'objectif est d'avoir un aperçu de la compatibilité du code avec les différentes cartes graphiques AMD disponibles sur ROCm. Ces travaux ont pour but de se détacher de NVIDIA pour aller vers AMD. Cette décision est régie par l'idée qu'une entreprise ne doit pas avoir le monopole d'une technologie. Il en est donc de même au sein de SOLEIL, NVIDIA ne doit pas constituer l'ensemble des cartes utilisées par SOLEIL. Cela évoque naturellement des questionnements éthiques, que nous aurons l'occasion de soulever dans un second temps. De là, mon travail a pour but de permettre le passage de paquet de phase Instable à la phase Testing dans le but d'inclure ces paquets dans la prochaine version de Debian. Ces paquets sont utilisés dans la conception de logiciels. Il est donc vital de les faire évoluer en phase avec les mises à jour de Debian. La prochaines est Trixie en référence au personnage de Toy Story, qui s'est fait fin juin. Le but est donc de réaliser les autopkgtest faisant intervenir de l'OpenCL sur la carte graphique AMD Radeon RX 6400 ayant le support de compilation, ROCm. En conséquence, comme évoqué dans les sections précédentes. Ces paquets constituent les briques des logiciels. Il est donc vital de vérifier si ces logiciels , et donc les paquets qui les composent fonctionnent bien sur les cartes que l'on possède. Surtout si l'objectif est à terme de se détacher de NVIDIA et donc des Américains le plus possible. Dans une idée de conserver au maximum la souveraineté des institutions scientifiques et de faire du local (Français et Européen). Voici les principaux paquets à équiper après identification. Cf [8]. J'ajoute également le lien vers le wiki que j'écris pour les futurs setups [9]. Pour équiper le paquet ROCm, je vais utiliser le script rocm-test-launcher, pour activer les tests d'installation/désinstallation sur une machine virtuelle vierge, l'autopkgtest. Le google summer of Code sur le sujet et le projet ont tous deux étés créés par un Mentor confirmé : Cordell Bloorand. Pour ce faire, je travaille sur la machine grades-01 où je me connecte en tant que « user ». Dans le cas où les tests ne fonctionnent pas, je dois corriger cela et implémenter les tests pour ROCm.

Voici les principaux paquets a équipé après indentifications :

xrstools, genx, XRT, mccode, hyperspy, freesas, navarp, mantis-xray, freeart, xraylib, tomogui, pyxrd, ufo-tofu, pyvkfft, python-mrcfile, pyhst2, python-opentsne, dioptas, python-tomopy, python-spectral, nabu, clpeak, clblast, pynx, genx, haskell-futhark, pyfai, pocl, ufo-filters, examl

Néanmoins, nous pouvons nous interroger sur les aspects éthiques qui peuvent intervenir dans le cadre de missions au sein de synchrotron-SOLEIL.

Ma mission possède elle-même une dimension éthique dans la mesure où Synchrotron-SOLEIL souhaite effectuer une transition de NVIDIA vers AMD. Le marché de l'IA est en plein boom. Cela nécessite de grande puissance de calcul, permise par la mise en place de cluster. Ces mêmes clusters sont composés de milliers de cartes graphiques. Aujourd'hui, le marché est détenu à 88 % par NVIDIA, une entreprise américaine à but lucrative. Depuis plus de 15 ans NVIDIA prend de l'avance sur la conception et la mise en place des technologies qui ont permis l'avènement de l'IA et de l'apprentissage profond. AMD est son concurrent commercial qui a le défaut d'avoir pris énormément de retard sur le développement de ces technologies en interne. De fait, il ne représente aujourd'hui que 12 % du marché mondial de l'IA. Il y a donc un risque évident de dépendance au géant NVIDIA, ce qui augure un monopole sur le marché des futures technologies de l'IA. Mon rôle est tout justement d'évaluer par le biais de tests, la dépendance effective ou non à NVIDIA. Je cherche effectivement à tester la compatibilité des paquets avec d'autres cartes graphiques que celles déjà en place dans les centres de calculs. Synchrotron-SOLEIL possède exclusivement des cartes NVIDIA compilant avec un langage très peu pratique nommé CUDA. Soleil nous offre donc un cas éthique d'étude : dans quelle mesure la monopolisation de certaines entreprises peut-elle représenter un danger pour l'indépendance des entreprises comme Synchrotron-SOLEIL ? Tout l'enjeu est justement de se détacher du monopole créé par NVIDIA en achetant des cartes AMD. A titre d'exemple, les expéditions de GPU data center de NVIDIA ont augmenté de +42 % en 2023 (de 2,64 M à

3,76 M unités), générant 36,2 Md$ de revenus, contre 10,9 Md$ en 2022. [19] Le maintien d'un concurrent pousse l'entreprise adverse à la compétitivité. Le fait d'avoir une forme d'opposition assure cette forme d'indépendance et d'anti-monopolisation. Dans le cas où cela se produirait, celui-ci fixerait ses conditions et dicterait les lois qui régiraient le marché de ces technologies. Le prix, la manière de compiler, de consommer et l'ensemble de la chaine, de sa conception à son utilisation seraient alors dans les mains d'une seule entreprise. Bien au-delà des simples aspects commerciaux évidents et de la richesse engendrée, cela soulève des questions éthiques et géopolitiques. Nous avons tous appris au travers des livres d'histoire que Franco, Mussolini, Staline, Hitler, ont tous été dictateur et ont imposé leurs idéaux par la force. Si une entreprise d'ordre mondiale détient à elle seule une technologie, il est tout à fait concevable que celle-ci impose par la force ses idéaux et prix par la même occasion. Quoi de plus amer pour une entreprise qu'être condamné à payer une technologie bien trop chère simplement parce que, « l'on ne peut pas faire autrement ils détiennent l'ensemble de cette technologie ». En réalité, au travers de cette question nous touchons du doigt un problème bien plus large, la liberté de choisir ce que l'on veut acheter ou non. Cela porte atteinte en réalité aux droits humains fondamentaux. Celui d'être libre de ses choix et de ses actions. « L'homme est né libre, et partout il est dans les fers. » (*Du contrat social*, Rousseau, 1762). Rousseau met en lumière la tension entre liberté naturelle et contraintes sociales ou économiques. Ainsi, à travers la mission principale qui m'a été attribuée, ce rapport d'éthique prend tout son sens : je participe au maintien d'une concurrence commerciale loyale et à l'indépendance commerciale de Soleil. Nous avons pu constater à travers notre réflexion quels seraient les dangers d'une monopolisation et au-delà de ça nous allons justement voire maintenant que cela pose aussi un problème de souveraineté.

La deuxième question qui s'est imposée à moi est liée à la première. Au-delà de la dépendance du laboratoire, la souveraineté informatique même de l'entreprise est remise en question par le principe même de ce qu'on appelle l'informatique de « confiance ». En effet, le conseil d'administration de Synchrotron-SOLEIL a fait le choix de changer de système de gestion de messagerie, en passant de Zimbra, un système public de messagerie français garantissant la confidentialité des données échangées au sein de SOLEIL et à l'extérieur, à Microsoft OutLook, un système de messagerie détenu par une entreprise privée américaine. Ainsi, on peut se demander s'il est encore possible de garantir la souveraineté informatique des laboratoires de recherches français en particulier lorsqu'il s'agit d'un laboratoire de pointe comme Synchrotron-SOLEIL ? Microsoft est une entreprise américaine à but lucrative, dont le gouvernement Américain à l'autorisation de consulter les données recueillies. Le gouvernement américain peut accéder aux données collectées et les stocker via Microsoft, même si ces données sont hébergées en dehors des États-Unis. Ces données sont recueillies via les courriels envoyés par la messagerie OutLook ou par le système d'exploitation lui-même via des programmes tels que Steps Recorder et autres. Ce pouvoir lui est conféré principalement par le Clarifying Lawful Overseas Use of Data Act, plus connu sous le nom de CLOUD Act, promulgué en mars 2018. Cette loi a été adoptée à la suite d'un litige opposant Microsoft au gouvernement américain, où les autorités demandaient l'accès à des courriels stockés dans un centre de données situé en Irlande dans le cadre d'une enquête pénale. Microsoft avait alors refusé, arguant que le mandat américain ne s'appliquait pas à des données hébergées hors des États-Unis. Le CLOUD Act a été conçu pour lever cette barrière juridique : il autorise explicitement les forces de l'ordre américaines à exiger l'accès à des données, qu'elles soient stockées sur le territoire américain ou à l'étranger, dès lors qu'elles sont détenues ou contrôlées par une entreprise relevant de la juridiction des États-Unis, comme Microsoft. [10][11][12][13]. Le CLOUD Act modifie ainsi la Stored Communications Act de 1986 et renforce l'extraterritorialité du droit américain en matière d'accès aux données numériques. En pratique, cela signifie que si Microsoft reçoit une demande légale (mandat, ordonnance judiciaire) émanant d'une autorité américaine, elle est tenue de fournir les données demandées, même si celles-ci sont hébergées dans un autre pays. Il permet un accès aux données sans nécessairement informer les utilisateurs concernés et sans passer par les tribunaux du pays où les données sont stockées. Enfin, il existe d'autres textes américains relatifs à la surveillance et à l'accès aux données, comme la Section 702 du Foreign Intelligence

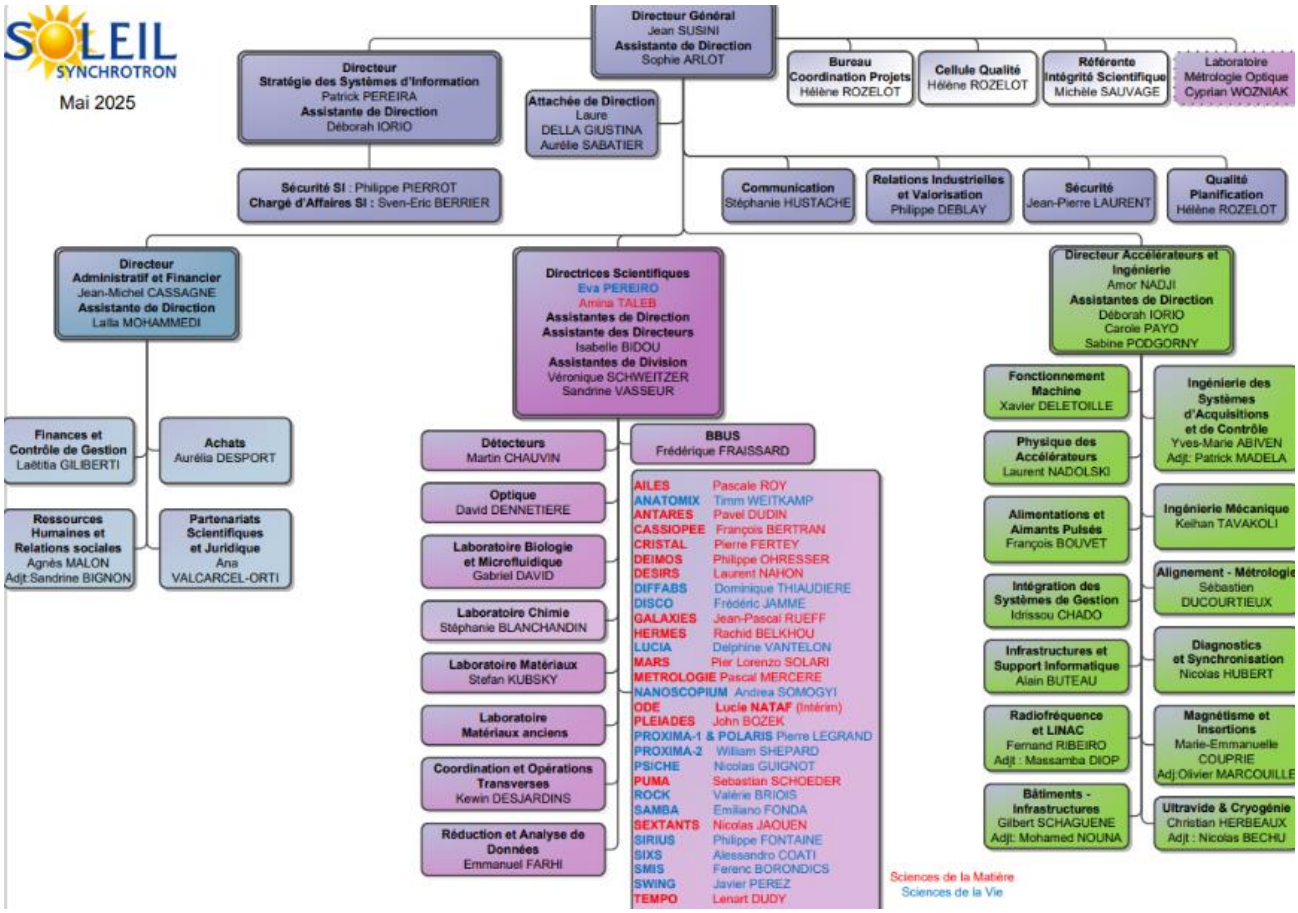
Surveillance Act (FISA), qui autorise la surveillance de communications électroniques pour des raisons de sécurité nationale. [14][15][16][17] En réalité, ce n'est pas question de SOLEIL spécifiquement, sous la sécurité se cache un enjeu glaçant, celui d'une société entièrement surveillée à la manière de *1984 -George Orwell*. De plus, 72% des utilisateurs sont sous Windows, cela signifie que 72 % de la population mondiale est sous surveillance, même l'armée Française est sous Windows.[18] Ce qui pose des problèmes de sécurité évident. Le fait de sous-traité à Windows implique également une fuite du savoir. Si personne ne fait du hardware dans l'entreprise alors le savoir se perd. Ce qui laisse le monopole à Microsoft. Notons que Microsoft peut éteindre des machines à distance. En réalité, Microsoft est bien plus puissant que la bombe atomique, une commande et 72 % des ordinateurs mondiaux s'éteignent. Notons que 66% des données générées par Synchrotrons-SOLEIL vont en Amérique. Cela représente environ 400Go par jour. Si les USA possède les données de Synchrotron-SOLEIL, cela présuppose que ces données peuvent être utilisées à des fin d'entrainement pour les IA générative, cela pose un problème évident car des informations confidentielles issues de l'entreprise peuvent être captée, comme les budgets, les postes à venir etc . De ce fait, l'entreprise est toujours devancée par quelqu'un qui « souffle » les réponses. De cette manière cela prend en compte le fait que Windows collecte les données en tâche de fond. Ces mêmes données collectées peuvent être utilisées pour l'apprentissage d'IA générative comme The ORG qui ressence les arbres de relations des individus et toutes leurs informations professionnelles. Cependant il est encore possible de garantir la souveraineté des laboratoires de recherches Français. Cela passe par la mise de côté de Microsoft et l'utilisation de système d'exploitation comme Linux et des messageries françaises sécurisées et publiques afin que les informations circulent de manière sécurisée sur le territoire.

Ainsi, lors de mon stage et à travers cette section, nous avons pu comprendre les enjeux de ma mission et mettre en lumière des problèmes éthiques évidents appliqués à un cas concret. Mon intégration au sein de Soleil a permis de mettre en lumière ces questions de souveraineté et de dépendance informatique. Ma mission principale a d'ailleurs pour but de contribuer à résoudre des potentielles dépendances commerciales. Ici les pratiques sont en changement, et ma mission est justement d'aider à opérer ce changement de pratique pour une souveraineté informatique et entrepreneuriale durable.

2.3 Dans quelle mesure cela s'inscrit -il dans le cadre de ma formation à l'ESIEE-PARIS ?

Ce stage traite des aspects techniques appris à ESIEE-PARIS, sur divers aspects il s'ancre parfaitement dans la formation dispensée. Les compétences et les connaissances nécessaires font directement appel à la formation qu'ESIEE-PARIS fournit. Notamment sur les notions traitant de l'utilisation de Gitlab / GitHUb via le projet de l'unité E3_IGI_3008 - Apprentissage de la programmation avec Python et C , où l'objectif était de coder un jeu de type TBA. Les notions de E3_IGI_3012 - Réseaux , et autres unités utilisant le système Debian m'ont permis d'aborder plus aisément ce stage. Ce faisant, j'ai pu appréhender plus aisément les notions que ce stage faisait intervenir. Il a néanmoins une multitude d'aspects du packaging Debian qui ne sont pas enseignés à ESIEE-PARIS. C'est une facette très spécifique de Debian, on touche directement à son cœur, à son mode de fonctionnement, à ses rouages. A l'ESIEE-PARIS, nous utilisons un aspect plus haut niveau de Debian, pour de la cybersécurité ou encore de l'informatique réseau. En réalité, ce sont deux aspects qui se complètent. Pour ces raisons, ce stage s'inscrit dans le cadre de ma formation à l'ESIEE-PARIS.

3. Fonctionnement interne de l'entreprise



Titre : Organigramme de Synchrotron -SOLEIL

Fonctionnement de l'Entreprise : Vous argumenterez une réponse sur un des thèmes suivants. Travail en équipe. Lors de votre stage, vous avez travaillé en équipe. Quelle était la taille de l'équipe ? Comment définiriez-vous le travail en équipe ? Quelles différences et similitudes constatez-vous avec vos expériences passées (école, association, sport ...) ? Identifiez les tâches, rôles, responsabilités des membres de l'équipe. Quels atouts du travail en équipe avez-vous constaté ? Quels freins avez-vous observé ? Comment avez-vous été intégré ? Quels talents et qualités avez-vous développé ? Comment un membre de l'équipe (vous ou un collègue) arrive à faire partager ses idées ?

Réponse :

Les unités de Synchrotron-SOLEIL sont interdépendantes, la division GRADES assure le fonctionnement des logiciels utilisés par les lignes de lumière et communique également étroitement avec la direction. Cela est crucial car cela permet de mettre en lumière les besoins et remarques de la division GRADES sur la manière de mener les acquisitions et expériences au sein de l'infrastructure. De plus, le Groupe SI, est chargé de la cybersécurité de l'entreprise etc et communique donc beaucoup avec GRADES. De fait, tout est interconnecté. Je joue également un rôle dans tout cela en tant que stagiaire. En tout, environ 358 employés, incluant internes et externes, sont présents sur site. L'ensemble des équipes communique plus ou moins avec les autres. Cela passe également par un outil que l'on appelle Nibelis et JIRA. JIRA est une plateforme multifonction qui vise à faciliter la gestion de projet en aidant à suivre les tâches, à identifier les blocages et à partager des informations entre les membres d'une équipe. À l'origine, dédiée au développement logiciel, la solution peut aujourd'hui s'adapter à des projets marketing, de finances, RH, etc. Une personne n'ayant pas connaissance d'une procédure ou ayant un problème au

sens large envoie un JIRA. L'équipe GRADES est composée d'une dizaine de personnes. Le travail est fondé sur la confiance et sur l'entraide. Les réunions sont faites de manière hebdomadaire, tous les vendredis à 10h30. Emmanuel FARHI, chef de la division GRADES, fait un retour sur ce qui a été dit lors des réunions de la direction, du CSE etc. Un tour de table des avancées de la semaine est effectué. En conséquence, lorsque j'ai demandé à Emmanuel FARHI mes horaires, il m'a répondu, que je faisais comme je voulais, l'idée est que le travail doit être réalisé et bien fait et que je dois faire mes 35h. Travailler dans un tel environnement est très valorisant car on se sent en confiance et que l'on sait que l'on nous fait confiance. Ce qui implique aussi que l'on ne veut pas décevoir et cela met malgré tout une pression pour moi, stagiaire. C'est une expérience totalement différente de ce j'ai connu à l'ESIEE-PARIS et dans le milieu associatif. J'ai beaucoup plus de responsabilités. Frederic Emmanuel PICCA fait du packaging Debian, il crée et maintient des paquets Debian utilisés par les lignes de lumières. Des collègues travaillent sur des algorithmes de Machines Learning et LLM appliqués aux études de SOLEIL. Nous avons 3 post doctorant dans l'équipe. La confiance est un atout, elle fluidifie toutes les opérations et laisse libre cours aux idées des éléments qui composent l'équipe. La direction peut être un frein selon certains points de vue. Dans le projet SOLEIL II, la direction n'a pas réellement pris en compte les avis de l'ensemble des scientifiques de lignes, ce qui a conduit à des désaccords profonds sur la réalisation du projet. Il aurait été, pertinent de consulter et de faire participer les scientifiques de lignes aux réunions sur les questions intrinsèques du projet SOLEIL II. J'ai été intégré à l'équipe officiellement aux yeux de tous lors de la réunion du vendredi où Emmanuel FARHI a pu me présenter. J'ai également fait ma présentation sur la Debian Science List, où j'ai décrit mes missions et mon intégration dans le projet Debian. J'ai également fait ma présentation à l'ensemble des personnes de SOLEIL via la réunion hebdomadaire SOLEIL qui permet d'annoncer les grands changements et lignes directrices et de présenter les nouveaux éléments. J'ai pu développer mon autonomie et ma résilience face aux tâches complexes et face à des notions qui m'étaient alors inconnues. Lorsque celles-ci se révélaient trop ardues, je consultais mes maîtres de stage et prenais soin d'écouter leurs conseils. J'ai pu développer des connaissances concrètes en packaging Debian et les mettre à profit sur des projets concrets. Nous partageons nos idées lors de la pause déjeuner et également par mail.



Management d'une équipe. Lors de votre stage, vous avez eu un responsable. Quel est son style managérial (directif, participatif, absent, ...) ? Comment avez-vous vécu le lien hiérarchique ? Quelles sont vos attentes d'un manager ? Interviewez votre responsable sur sa vision du management de l'équipe. A l'issue de votre stage, quel type de management souhaiteriez-vous en tant que subordonné ?

Réponse :

Emmanuel FARHI et Frédéric-Emmanuel PICCA m'ont laissé beaucoup d'autonomie et m'ont fait confiance. Cependant, lorsque j'avais besoin d'aide ils ont toujours été très à l'écoute et bienveillants, même si mes questions semblaient triviales à leurs yeux. J'ai très bien vécu ce lien hiérarchique, nous entretenions un lien amical mais très professionnel. Un manager se doit de laisser les talents de ses équipes s'exprimer et faire confiance aux talents qui les composent. Un management trop agressif et basé sur la compétition est très destructeur, cependant, une compétitivité bienveillante peut être synonyme de croissance au sein de l'équipe. A titre personnel, je souhaite entretenir un lien de confiance avec mes supérieurs et collègues. Pour moi l'autonomie et la confiance sont la clef. Cela demande également une grande responsabilité personnelle.

Accomplir une mission. Lors de votre stage, vous avez rempli une mission. Définissez-la. Aviez-vous des objectifs (qualitatifs et/ou quantitatifs) à atteindre ? Avez-vous atteint ou dépassé ces objectifs ? Sinon pourquoi ? Identifiez et expliquez votre plus grande réussite/satisfaction. Quels sont les moyens mis à votre disposition ? Citez un échec auquel vous ou votre équipe avez été confronté(e) ? Qu'en avez-vous appris pour l'avenir ? A quelles difficultés avez-vous été confronté au cours de votre mission ? Quelles suggestions d'améliorations, initiatives pouvez-vous proposer ?

Réponse :

Lors des précédentes sections j'ai pu évoquer le contenu et la portée de ma mission. J'ai d'abord et avant toutes choses des objectifs qualitatifs à atteindre. A l'heure où j'écris ce rapport, il y a bon nombre de contretemps, de bugs en tout genre sur les paquets Debian sur lesquels je travaille et qui sont intrinsèques aux codes sources. Je développerai les raisons et les bugs/problèmes que j'ai pu rencontrer dans la partie 3. Il y a également des contraintes matérielles plutôt du côté hardware. Pour le moment ma mission est sur le point d'être atteinte, mais ne l'est pas encore. Tous les moyens sont mis à ma disposition pour y arriver, internet, mes maîtres de stages, collègues, l'IA, intranet etc. Je pense notamment au paquet Pynx, il se trouve qu'après avoir résolu tous les bugs provenant de l'absence des dépendances dans le fichier debian/tests/control, le paquet présentait encore et toujours des erreurs. Ce n'étaient pas des erreurs classiques, de dépendances et d'erreurs des tests mis en place dans le paquet mais des erreurs de codes. Malheureusement, étant donné que le paquet ne m'appartient pas, je dois faire des patchs, etc. Mais lorsque l'on fait des patchs à répétitions, il y a souvent des problèmes de fusions et de merge. Cependant, il est infiniment plus simple que le détenteur du code fasse lui-même les modifications. Nous avons donc décider d'opter pour un bugs report à l'upstream dans l'espoir qu'il résolve les bugs. Je n'ai pas d'amélioration spécifique à apporter, Synchrotron-SOLEIL est un environnement plaisant et sains dans lequel on évolue en confiance.

Relation client/fournisseur. Si lors de votre stage, vous avez été très souvent en relation avec des clients ou des fournisseurs internes/externes, quelles stratégies sont utilisées pour le satisfaire (pour lui faire préciser son besoin, quelle est la logistique pour répondre à son besoin...) ? Quelles méthodes/techniques sont utilisées pour gérer un conflit ? Quels aspects de la relation avec un client appréciez-vous? Quels sont ceux que vous n'appréciez pas? Argumentez. Qu'avez-vous appris de cet aspect relation client (ou fournisseur) pour votre futur métier d'ingénieur ?

Réponse :

Je n'ai pas du tout eu de lien avec l'aspect Relation Client/fournisseur à mon échelle et dans la division dans laquelle je suis. Je ne peux donc pas répondre à ces questions avec justesse. Mais lors de mon stage, j'ai rapidement compris que les ingénieurs de SOLEIL évoluent en permanence dans une logique de relation client-fournisseur, aussi bien en interne (avec les chercheurs et utilisateurs scientifiques) qu'en externe (avec les industriels et prestataires). Pour répondre aux besoins exprimés, j'ai observé qu'une première stratégie consiste à clarifier la demande par des réunions, où le scientifique précise ses contraintes expérimentales et où l'ingénieur reformule en termes techniques (performances, délais, coûts). J'ai réalisé que cette phase est cruciale pour transformer une attente souvent générale en cahier des charges vraiment exploitable. Sur le plan logistique, j'ai découvert que SOLEIL suit des processus très rigoureux : planification des étapes, rédaction d'appels d'offres pour les achats, suivi de livraison et de conformité. L'objectif que j'ai perçu est d'assurer à la fois la qualité technique et le respect des délais. En cas de conflit (par exemple divergence sur les spécifications ou retard fournisseur), la méthode privilégiée reste le dialogue structuré : identification claire du problème, recherche de compromis techniques, et si nécessaire, arbitrage au niveau du chef de projet ou du comité de pilotage. Les aspects que j'ai vraiment appréciés dans cette relation sont la dimension collaborative et l'échange constant avec des interlocuteurs très variés : chercheurs, ingénieurs, techniciens et industriels. Cela crée un environnement particulièrement dynamique et enrichissant que j'ai

trouvé stimulant. En revanche, j'ai ressenti comme plus contraignantes les étapes administratives (procédures d'achat, validations multiples), qui bien qu'indispensables, ralentissent parfois le processus d'innovation et peuvent être frustrantes. De cette expérience concrète, j'ai surtout retenu que la relation client-fournisseur est véritablement au cœur du métier d'ingénieur : il ne s'agit pas seulement de maîtriser des compétences techniques, mais aussi de développer une réelle capacité d'écoute, de communication et de gestion de compromis. Pour mon futur métier, cette immersion m'a fait comprendre l'importance de savoir traduire clairement un besoin en spécifications techniques concrètes, tout en gardant une posture ouverte et résolument orientée solution face aux contraintes ou désaccords inévitables.

Localisation géographique. Si vous avez choisi ce stage en raison de sa localisation géographique, détaillez les raisons de votre choix. En quoi le stage confirme-t-il ou infirme-t-il l'idée que vous vous étiez faite de ce pays/lieu ? Argumentez en vous appuyant sur des exemples concrets. Quel impact cela a-t-il sur votre projet personnel et professionnel ?

Réponse :

Je n'ai pas choisi ce stage en raison de sa situation géographique. Cela confirme assez bien l'idée que je me suis fait de cette entreprise, j'ai été cependant très surpris et touché de la bienveillance et de la confiance qui m'ont été accordées lors de ce stage. On m'a accordé les droits de super-utilisateur sur le serveur alors qu'étant que stagiaire je peux commettre des erreurs. Cela révèle que l'on place en moi une grande confiance. C'est une entreprise à taille humaine, même si elle s'avère très vaste sur le plan de la superficie. De manière générale, ce stage a eu une influence directe sur mon projet professionnel car cela permet de mettre un pied dans le monde des accélérateurs de particules. Je souhaite pour la suite poursuivre dans un autre accélérateur en Europe ou dans le monde ou même au CEA. Cela me permet d'avoir une expérience concrète dans ce type d'installation, et les mettre en valeurs pour mes futurs stages. Le packaging Debian est assez répandu dans le monde scientifique car cela à l'assurance d'avoir des éléments open source et indépendants. En réalité, c'est très prisé car cela offre une grande souveraineté à celui qui l'utilise, aucune dépendance à aucune institution, ni de siphonage de données sur le système Debian.

Ingénieurs. Si votre entreprise d'accueil emploie des ingénieurs, identifiez (en interviewant par exemple des ingénieurs) des métiers d'ingénieurs rencontrés au sein de l'entreprise : missions, fonctions, responsabilité d'équipe, de budget, avec qui sont-ils en relation, ... ? Quelles sont les compétences nécessaires et les qualités requises pour être ingénieur dans cette entreprise ? Quelles sont les évolutions de carrière au sein de cette entreprise ? Quelles tâches trouvez-vous intéressantes? Quelles sont celles qui vous déplaisent ? Argumentez en vous appuyant sur des exemples concrets. Quel est l'impact sur votre projet personnel et professionnel ?

Réponse :

SOLEIL emploi des ingénieurs, des scientifiques de lignes ayant pour missions de régler, rendre les données recueillies de meilleures qualités, en améliorant les optiques permettant l'analyse de l'échantillon au sein de la machine. De plus, la machine qui servira dans la cabine est bien souvent montée et pensée à SOLEIL par ces mêmes scientifiques et ingénieurs. À SOLEIL, j'ai pu découvrir plusieurs profils d'ingénieurs qui contribuent au fonctionnement et à l'évolution de l'installation. Les ingénieurs en électronique et systèmes numériques conçoivent et intègrent les cartes d'acquisition rapides (ADC, FPGA, MicroTCA) pour les diagnostics faisceau et le timing. Les ingénieurs en acquisition et intégration de détecteurs installent et adaptent les détecteurs des lignes de lumières, développant les chaînes de lecture en lien avec les chercheurs et les fournisseurs. Les ingénieurs CVC/Fluides assurent la conception et la maintenance des infrastructures thermiques et de refroidissement indispensable à la stabilité du faisceau. J'ai aussi rencontré des assistants et apprentis ingénieurs, qui participent aux arrêts machine, au support logiciel et au suivi technique [5]. Enfin, une partie des ingénieurs travaille sur le contrôle-commande TANGO et les logiciels

scientifiques, développant les outils de supervision, d'archivage et de traitement de données en collaboration avec le consortium international. Ensemble, ces métiers couvrent l'électronique, l'instrumentation, les infrastructures et le numérique, avec des responsabilités partagées entre conception, exploitation et innovation au service de la recherche. Ces métiers exigent des compétences techniques pointues (électronique, FPGA, contrôle-commande, optique rayons X, programmation scientifique) mais aussi la gestion de projets et d'achats publics. J'ai constaté que des qualités humaines sont indispensables : rigueur, curiosité scientifique, communication claire avec des interlocuteurs variés, et capacité de travail en équipe dans un environnement international. J'ai appris que les ingénieurs peuvent évoluer vers une expertise technique reconnue, vers des postes de management d'équipes ou encore vers le pilotage de projets stratégiques (comme l'Upgrade SOLEIL II). La mobilité entre spécialités et lignes de lumières est fréquente et valorisée. D'après mes observations, les tâches les plus motivantes sont la mise en service de nouveaux équipements (diagnostics, détecteurs) et la résolution de problèmes en temps réel sur le faisceau, car elles combinent innovation et impact immédiat. En revanche, les procédures administratives d'achats et la maintenance logicielle de long terme sont plus contraignantes, bien que nécessaires. Ce stage a confirmé mon intérêt pour l'ingénierie scientifique au service de la recherche. Je souhaite m'orienter vers un rôle mêlant données, contrôle-commande et instrumentation, dans un contexte international comme DESY/SOLEIL ou le CERN. Les ingénieurs de SOLEIL m'ont montré l'importance de combiner expertise technique et esprit collaboratif, compétences et l'aspect de souveraineté scientifique que je veux développer davantage dans mon parcours et ma vie professionnelle.

Conclusion de la partie 1 :

Dans cette partie, nous avons pu comprendre les enjeux et la nature de ce qui est fait à Synchrotron-SOLEIL. Notamment sur l'aspect mondial et la nature des recherches qui y sont effectuées. C'est un cluster, il concentre la recherche et permet de réaliser des avancées conséquentes à l'aide des instruments mis en service. La compréhension de l'infrastructure, que ce soit au niveau du calculateur ou du synchrotron en lui-même et de ses évolutions est un point clef pour appréhender au mieux les fondements de mes missions. La portée personnelle et professionnelle de ce stage a pu être comprise. Avec ces éléments, l'ensemble de mes missions ont pu être comprises et intégrées dans le cadre de ma formation à l'ESIEE-PARIS. Nous avons jusqu'à présent analysé la partie technique et technologique de ce stage, cependant l'aspect humain est très important et doit être pris en compte. C'est pourquoi, l'analyse du fonctionnement interne de l'entreprise permet une analyse plus poussée du management et des liens sociaux qui y sont entretenus.

Partie 2 : Le système Debian

Sommaire

1. Qu'est-ce que le système open source Debian ?
2. Comment Debian fonctionne-t-il ?
    - 2.1 Les versions et le concept de Stabilité d'un paquet
    - 2.2 Debian maininglist de Debian Science
    - 2.3 Debian Tracker
    - 2.4 Debian package
    - 2.5 Debian Salsa Git
    - 2.6 Debian wiki
    - 2.7 Les Patch et Bugs reports chez Debian
3. Les supports matériels CPU et GPU
    - 3.1 NVIDIA
    - 3.2 AMD



1. Qu’est-ce que le système Open source Debian ?

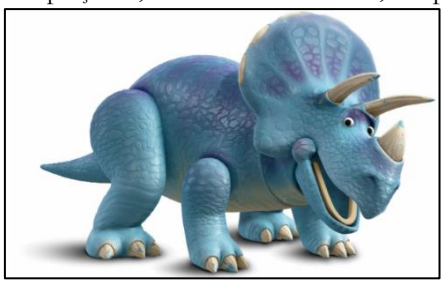
Debian est un système d'exploitation open source et libre, qui repose principalement sur le noyau Linux. Cependant, il existe également des versions expérimentales basées sur d'autres noyaux tels que GNU Hurd ou FreeBSD. Cette distribution, élaborée en partenariat avec une communauté mondiale de volontaires, se caractérise par sa solidité, sa sûreté et son dévouement aux valeurs du logiciel open source. À la différence de distributions commerciales telles que Red Hat ou Ubuntu (qui est une version de Debian), Debian repose sur un modèle communautaire et bénéficie du soutien d'une organisation sans but lucratif. Logiciel d'intérêt public. Créé en août 1993 par Ian Murdock, qui était alors étudiant à l'Université Purdue, Debian doit son nom à l'union des prénoms Ian et Debra, sa compagne de l'époque. Depuis sa création, le projet s'est affirmé comme un modèle dans l'univers du logiciel libre, particulièrement en adoptant en 1997 le Contrat Social Debian et les Directives Debian sur les Logiciels Libres (DFSG), qui établissent son dévouement éthique. La définition officielle de l'open source par l'Open Source Initiative (OSI) a d'ailleurs été influencée par ces principes. Au cours de son histoire, Debian a franchi des jalons importants, tels que l'introduction de dpkg en 1996 (précurseur d'APT), le déploiement de sa première version stable majeure (Debian 2.0 « Hamm ») en 2000, ou encore la montée en popularité d'Ubuntu en 2005, une distribution dérivée qui a rendu Linux accessible au grand public. Actuellement, Debian demeure l'une des distributions les plus influentes, avec un rythme de publication stable d'environ deux ans. Debian est administré selon un modèle démocratique et décentralisé, où les décisions sont déterminées soit par consensus, soit par scrutin parmi les développeurs officiels. Ces individus, connus sous le nom de Développeurs Debian (DDs), doivent suivre une procédure stricte pour obtenir l'autorisation de gérer les paquets officiels. Le projet est supervisé par un Leader Debian (DPL), élu de manière annuelle, et appuyé par plusieurs équipes techniques, telles que l'Équipe de Gestion des Versions (Release Team), l'Équipe de Sécurité Debian (mises à jour de sécurité) ou les Administrateurs Système Debian (DSA, gestion de l'infrastructure). À l'inverse de plusieurs autres distributions, Debian ne repose pas sur une société, mais s'organise autour d'une structure associative et communautaire. Du point de vue technique, Debian se démarque grâce à son système de gestion de paquets sophistiqué, incluant APT (Advanced Package Tool) et dpkg, facilitant l'installation et la mise à jour des programmes. Elle répond à une multitude de besoins, allant des serveurs aux ordinateurs de bureau et incluant les systèmes intégrés, avec plus de 60 000 paquets à sa disposition. Les utilisateurs ont la possibilité de sélectionner parmi plusieurs axes de développement : Stable (version hautement fiable, actualisée tous les deux ans), Testing (prochaine version stable) et Unstable (version constamment en cours d’évolution, destinée aux contributeurs chevronnés). Debian est aussi compatible avec un large éventail d'architectures matérielles, y compris amd64, arm64 et ppc64el, ce qui en fait une option polyvalente pour divers usages. Debian, au-delà de ses caractéristiques techniques, représente une philosophie solide axée sur la liberté des logiciels et la coopération communautaire. Le Contrat Social assure que le système demeurera entièrement libre, sauf pour quelques firmwares propriétaires optionnels. Cette philosophie a largement influencé l'univers du logiciel libre, touchant des distributions telles qu'Ubuntu, Linux Mint ou Kali Linux, ainsi que des entreprises et des entités publiques. En définitive, Debian est davantage qu'une simple distribution : c'est un projet novateur qui conjugue qualité technique et principes humanistes, demeurant après trente années une pierre angulaire de l'univers open source. [19][20]

2. Comment Debian fonctionne-t-il ?

2.1 Les versions et le concept de stabilité d’un paquet

Debian gère un ensemble conséquent de paquets, environ 60 000 comme énoncé dans la section précédente. Chaque paquet possède 3 versions activement retenue, *« stable » , « testing » et « unstable »* . La distribution stable contient la dernière distribution officiellement sortie de Debian. Le fonctionnement de Debian repose sur un système de versions bien structuré qui permet d’équilibrer innovation et fiabilité.

Contrairement à certaines distributions qui fonctionnent en « rolling release » et intègrent en continu les dernières nouveautés, Debian suit une organisation en plusieurs branches distinctes, qui correspondent à différents niveaux de stabilité des paquets. Cette organisation permet d'atteindre un haut niveau de qualité, tout en répondant aux besoins d'une communauté très diverse, allant des utilisateurs débutants aux administrateurs système. Tout commence avec la branche dite *unstable*, aussi appelée « Sid ». C'est là que les nouveaux paquets sont d'abord introduits. Ces logiciels sont les plus récents disponibles, parfois même encore en développement. Leur intégration est rapide, ce qui permet de tester rapidement de nouvelles versions. Mais comme leur stabilité n'est pas garantie, ils ne sont pas encore considérés comme prêts pour une utilisation quotidienne sur des systèmes sensibles [21]. Une fois qu'un paquet a été intégré dans *unstable*, il peut migrer vers la branche *testing*. Cette branche joue un rôle de transition. Pour qu'un paquet soit accepté dans *testing*, il ne doit pas avoir provoqué de problèmes critiques dans *unstable*, et toutes ses dépendances doivent être satisfaites. Autrement dit, *testing* ne reçoit que des paquets qui semblent fonctionner correctement et qui sont techniquement compatibles avec le reste du système. C'est cette branche qui, à terme, deviendra la prochaine version stable de Debian. Elle bénéficie donc d'une attention particulière en amont de chaque publication officielle [22]. Lorsque l'équipe Debian décide qu'une nouvelle version stable est prête à être publiée, *testing* entre dans une période dite de gel. Pendant cette phase, seuls les correctifs essentiels sont encore acceptés. Cela permet de stabiliser complètement l'ensemble du système avant la sortie officielle. Une fois le gel terminé, *testing* est promue en tant que nouvelle version stable, et un nouveau cycle recommence. En moyenne, Debian publie une version stable tous les deux ans. Par exemple, la version 12, nommée « Bookworm », est sortie en juin 2023, et sa remplaçante, Debian 13 « Trixie », est prévue pour août 2025 [23]. Actuellement, la distribution stable de Debian est la version 12, nom de code *Bookworm*. Elle a été initialement publiée en tant que version 12.0 le 10 juin 2023 et sa dernière mise à jour, version 12.11, a été publiée le 17 mai 2025. Pour anecdote les noms de code font références à des personnages de Toy Story. La prochaine version, Trixie, prévue pour le 9 août 2025, fait référence au dinosaure de Toy Story. (ci-contre) . Dans la branche *stable*, ce terme prend tout son sens. Une fois qu'un paquet est intégré à la version stable, il ne change plus ou du moins, il ne reçoit plus de mises à jour majeures. Les seules modifications autorisées concernent les failles de sécurité ou des bugs critiques. Cela permet d'éviter les mauvaises surprises et garantit que le système reste identique au fil du temps, ce qui est indispensable dans les environnements de production. Lorsqu'un correctif est nécessaire, les développeurs effectuent ce qu'on appelle un *backport* : ils adaptent la correction à la version déjà publiée du logiciel, sans introduire les autres changements de la version plus récente. Cette rigueur permet à Debian stable d'offrir un niveau de fiabilité remarquable [24]. Tous les quelques mois, Debian publie une « révision » de la version stable, appelée *point release*. Ces versions ne sont pas de nouvelles moutures à proprement parler, mais elles regroupent les correctifs de sécurité et autres mises à jour critiques accumulés depuis la dernière publication. Par exemple, Debian 12.11 est une révision de Bookworm qui inclut tous les correctifs jusqu'en mai 2025, sans changer la base du système. Cela permet aux utilisateurs d'installer un système à jour dès le départ, sans avoir à télécharger ensuite toutes les mises à jour [25]. À côté de ces trois branches principales (*unstable*, *testing* et *stable*) , il existe également une branche appelée *experimental*. Comme son nom l'indique, elle sert à tester des versions très instables ou des transitions majeures. Elle n'est pas destinée à un usage classique, mais elle est utile aux développeurs qui souhaitent tester des fonctionnalités encore en chantier. En général, les paquets d'*experimental* doivent être explicitement demandés et ne s'installent pas automatiquement [26]. Ce système de branches est au cœur de la philosophie Debian. Il permet d'assurer un équilibre constant entre évolution technologique et robustesse. Chacune de ces branches répond à des besoins différents : ceux qui veulent profiter des toutes dernières nouveautés peuvent se tourner vers *unstable* ou *testing*, tandis que ceux



qui privilégient la stabilité et la sécurité choisiront sans hésiter *stable*. Grâce à cette architecture claire et maîtrisée, Debian offre une grande liberté d'usage tout en maintenant une rigueur exemplaire dans la gestion des logiciels [27].

2.2 Debian maininglist de Debian Science

Le projet Debian Science a été initié pour répondre à un besoin croissant : celui de fournir aux chercheurs et scientifiques un environnement Debian enrichi de logiciels adaptés à leurs disciplines. Cette initiative vise à intégrer des outils scientifiques libres au sein de la distribution Debian, facilitant ainsi leur installation et leur utilisation. Le projet s'articule autour de plusieurs composantes, dont la création d'une mailing list dédiée : debian-science. L'idée de créer une équipe Debian spécifiquement orientée vers les sciences est née lors de la DebConf 5 à Helsinki, en 2005. Cette rencontre a mis en évidence le besoin d'une structure au sein de Debian capable de regrouper les efforts autour des logiciels scientifiques. Ainsi, la mailing list *debian-science@lists.debian.org* a été lancée pour centraliser les discussions et les contributions relatives à ce domaine. Elle sert de plateforme pour échanger sur l'utilisation de Debian dans les recherches scientifiques, partager des informations sur les paquets disponibles, et discuter des problèmes spécifiques rencontrés par les utilisateurs scientifiques de Debian. La mailing list debian-science a pour objectifs principaux de :

- Faciliter les échanges entre les utilisateurs et les développeurs de logiciels scientifiques.
- Discuter des améliorations à apporter à Debian pour mieux répondre aux besoins des chercheurs.
- Partager des informations sur les paquets scientifiques disponibles et leur utilisation.

Elle est ouverte à tous, sans modération préalable, permettant ainsi une communication fluide et transparente. Les messages envoyés à cette liste sont archivés et accessibles publiquement, offrant une ressource précieuse pour la communauté scientifique . Le projet fonctionne selon le modèle des Debian Blends, une approche qui consiste à regrouper des paquets logiciels autour d'un thème ou d'un domaine d'application spécifique. Debian Science agit comme une couverture, couvrant les domaines scientifiques qui n'ont pas encore de projet dédié. Pour assurer une gestion efficace des communications, Debian Science utilise deux listes de diffusion distinctes :

- *debian-science@lists.debian.org* : destinée aux discussions générales, aux retours d'expérience et aux échanges entre utilisateurs et développeurs.
- *debian-science-maintainers@lists.debian.org* : réservée aux notifications automatiques liées à la maintenance des paquets, telles que les rapports de bugs ou les mises à jour du système de suivi des paquets (BTS)

Cette séparation permet de maintenir une communication claire et structurée au sein du projet, en évitant la surcharge d'informations non pertinentes pour les participants aux discussions. La création de la mailing list debian-science et du projet Debian Science dans son ensemble répond à un besoin essentiel : celui de fournir aux chercheurs un environnement Debian adapté à leurs besoins spécifiques. En centralisant les discussions, en structurant les efforts autour de métapaquets thématiques et en favorisant la collaboration

au sein de la communauté, Debian Science contribue à enrichir l'écosystème Debian et à soutenir la recherche scientifique avec des outils libres et performants. [27][28][29]

2.3 Debian Tracker

Le Debian Tracker est un outil indispensable au sein de l'écosystème Debian, conçu pour suivre, gérer et garantir la qualité des paquets logiciels qui composent cette distribution [30]. Il se compose principalement de deux services complémentaires : le Debian Package Tracker et le Package Tracking System . Ces deux systèmes travaillent de concert pour offrir une vision claire et transparente de l'état des paquets tout au long de leur cycle de vie [31]. Le Debian Package Tracker est une plateforme web qui permet aux mainteneurs, développeurs et utilisateurs avancés de suivre l'évolution des paquets. Grâce à lui, il est possible de consulter des informations détaillées, de vérifier les journaux de construction, et de recevoir des alertes en cas de mises à jour ou de problèmes détectés. Ce service repose sur un logiciel développé par la communauté Debian pour centraliser la gestion des paquets et simplifier leur suivi [32]. Le Package Tracking System, quant à lui, complète ce dispositif en fournissant un historique précis des paquets sources. Il s'avère particulièrement utile pour les co-mainteneurs et les équipes de contrôle qualité, qui peuvent ainsi vérifier la santé et la stabilité des paquets sur les différentes versions de Debian [33]. Les données utilisées par le Debian Tracker proviennent de plusieurs sources clés : le système de suivi des bugs Debian, qui recense toutes les remontées d'erreurs et les demandes d'amélioration ; les journaux de construction, qui détaillent le processus de compilation des paquets ; et les dépôts de code source, où sont stockées les différentes versions des logiciels. Toutes ces informations sont ensuite regroupées et présentées de manière claire et accessible sur les interfaces web du Tracker et du Package Tracking System. Les utilisateurs peuvent ainsi chercher un paquet, consulter son état actuel, s'abonner à des notifications personnalisées, et accéder à toutes les données utiles en un seul endroit [30][33]. L'intérêt principal du Debian Tracker est de faciliter la gestion et la maintenance des paquets, tout en garantissant leur qualité et leur sécurité. Il aide les équipes à identifier rapidement les paquets qui posent problème, à suivre les mises à jour, et à assurer une meilleure collaboration entre les développeurs. En rendant le processus transparent, il encourage aussi la participation de la communauté et renforce la confiance des utilisateurs dans la distribution Debian [30][31]. En résumé, le Debian Tracker joue un rôle fondamental dans le maintien de la robustesse et de la fiabilité de Debian. En centralisant les informations, en automatisant leur suivi et en facilitant la communication entre tous les acteurs du projet, il contribue à faire de Debian une distribution stable, évolutive et en constante amélioration [30][31].

2.4 Debian package

Le système de gestion des paquets Debian est au cœur de l'administration d'un système Debian, assurant une gestion cohérente, sécurisée et efficace des logiciels. Les paquets Debian, au format .deb, sont des archives contenant les fichiers nécessaires à l'installation d'un logiciel, ainsi que des métadonnées décrivant son fonctionnement et ses dépendances. Chaque section a un objectif spécifique : la section main contient des logiciels libres conformes aux directives du logiciel libre de Debian, tandis que les sections contrib et non-free contiennent des logiciels libres dépendant de logiciels non libres ou ayant des licences restrictives. La gestion des paquets Debian repose sur plusieurs outils. dpkg est un outil de bas niveau permettant d'installer, de configurer et de supprimer des paquets locaux. Il gère directement les fichiers .deb et est essentiel pour les opérations locales. Cependant, dpkg ne résout pas les dépendances entre paquets. C'est là qu'intervient APT (Advanced Packaging Tool), une interface de plus haut niveau qui gère les installations à partir de dépôts distants, résout les dépendances et effectue les mises à jour de manière transparente. APT offre des outils en ligne de commande tels que apt-get, apt-cache et apt, qui simplifient la gestion des paquets. Le suivi des paquets est également une composante essentielle du système. Le Debian Package permet de suivre l'activité d'un paquet source, d'être informé des mises à jour et de contribuer au projet. Cela assure une transparence et une collaboration efficaces au sein de la communauté

Debian. Le système de gestion des paquets Debian est conçu pour assurer une administration efficace, sécurisée et cohérente des logiciels sur cette distribution Linux. Il repose sur des outils robustes et une organisation claire des dépôts, permettant aux utilisateurs et aux administrateurs de gérer efficacement les logiciels sur leurs systèmes Debian.

2.5 Debian Salsa Git

Debian Salsa est la plateforme GitLab officielle du projet Debian, disponible à salsa.debian.org, conçue pour héberger des dépôts Git collaboratifs, gérer des équipes de mainteneurs et coordonner les workflows de packaging pour les paquets Debian [34]. Chaque paquet Debian ou documentation est hébergé dans un dépôt Git sur Salsa. Les utilisateurs et équipes (groupes) peuvent créer, forker, cloner, et collaborer autour de ces dépôts. Une interface GitLab permet de gérer les accès, les permissions et les membres des groupes [36]. Le script salsa, fourni par le paquet devscripts, facilite la création, la configuration et la gestion de projets et de groupes sur cette plateforme, en automatisant des tâches comme l'ajout d'utilisateurs ou le suivi des pipelines CI [35]. Le nom « Salsa » est un jeu de mots sur « source » → « sauce » → « salsa », en accord avec la tradition Debian de choisir des noms originaux pour ses services [36]. Apparue à la suite d'un sprint Alioth en août 2017, la plateforme est restée en version bêta jusqu'en janvier 2018 avant de remplacer définitivement l'ancien service FusionForge alias Alioth [37]. Un élément central de cette infrastructure est Salsa CI, le pipeline d'intégration continue intégrée à Salsa. À chaque commit Git, il déclenche jusqu'à 17 jobs pour compiler le paquet sur plusieurs architectures (amd64, i386, ARM…), exécuter les tests automatiques autopkgtest, et analyser la qualité du paquet avec Lintian, afin de détecter rapidement les régressions avant tout déploiement officiel [38]. Ce système a permis de réduire un cycle de vérification qui prenait jusqu'à plusieurs jours en quelques minutes [38]. L'adoption de Salsa CI est désormais massive : en 2023, plus de 9 000 projets Debian l'utilisait, et au début de 2025, ce nombre dépassait 27 000 paquets source. Certains contributeurs Ubuntu commencent également à l'adopter pour améliorer la qualité des paquets avant publication [39]. La plateforme a ainsi renforcé la fiabilité de la migration des paquets de Debian Unstable vers Testing, tout en réduisant les erreurs humaines [38]. Salsa répond aux enjeux clés du développement Debian moderne : elle fournit une collaboration centralisée pour les mainteneurs, une traçabilité complète via Git, une qualité logicielle améliorée grâce à des tests automatisés, et une facilitation des contributions externes dans une infrastructure sécurisée et transparente. Une fois que les modifications locales fonctionnent, les modifications sont soit push directement sur le git salsa du paquet ou bien l'on fait un merge request afin de demander à l'upstream si l'on peut mettre les modifications sur le git.

2.6 Debian wiki

Les wikis Debian constituent une plateforme communautaire basée sur le moteur *MoinMoin* ,permettant à chacun de créer, modifier et enrichir des pages documentaires, des tutoriels, des guides techniques ou d'actualité. Ils sont conçus pour offrir une expérience intuitive grâce à des fonctionnalités comme la recherche intégrée, l'historique des modifications, l'indexation et l'accès aux pages récentes wiki.debian.org. Les contributions sont ouvertes à toute personne, favorisant ainsi une documentation toujours à jour et collaborative. De plus, toute modification postérieure au *24 juillet 2025* est publiée sous licence *CC BY-SA 4.0*, à moins qu'une autre licence ne soit explicitement indiquée. Ces wikis remplissent plusieurs rôles essentiels : ils servent avant tout de référentiel technique et pédagogique pour des sujets comme l'installation, la configuration, l'usage des paquets ou les environnements de bureau, tout en offrant un lieu de support communautaire où les utilisateurs peuvent échanger des astuces ou corriger des contenus. Ils permettent également de diffuser des actualités et annonces. Debian Science s'organise autour d'un wiki central qui sert de point de coordination pour les activités du projet. Ce wiki contient des informations sur les métapaquets scientifiques, les domaines d'application, les bonnes pratiques de

packaging, et les tâches en cours. Il permet aux contributeurs de suivre l'évolution du projet et de s'impliquer dans les différentes initiatives.

J'ai d'ailleurs créé un wiki qui décrit concrètement comment intégrer des *autopkgtests* pour des logiciels OpenCL tels que ROCm ou pocl. Il détaille la création de fichiers debian/tests/control, la configuration des dépendances (Depends, Restrictions, Test-Command), l'ajout de test-name ROCm, et l'usage de conteneurs avec podman ou unshare pour exécuter les tests [43]. Ces tests s'appuient sur des mécanismes fondamentaux. Un *chroot* est un environnement isolé utilisé par *sbuild* pour construire les paquets Debian dans un contexte minimal, garantissant ainsi l'absence de dépendances externes ou de pollution du système hôte [41]. L'outil *sbuild* orchestre ces builds dans des *chroots* fournis par schroot, unshare, ou via l'option --chroot-mode=autopkgtest, ce qui permet d'utiliser des containers LXC, QEMU, Podman ou des hôtes distants pour les tests [42]. *Autopkgtest*, quant à lui, exécute des tests runtime dans ces environnements isolés. Il se base sur la spécification DEP-8, avec des fichiers debian/tests, et prend en charge des modes d'isolation comme *isolation-container* ou *machine* pour des tests plus complexes nécessitant un conteneur ou une VM dédiée [40]. Le système détecte automatiquement les conteneurs, ou QEMU selon la configuration, et permet de filtrer les tests avec des options comme --test-name [40]. Le wiki OpenCL illustre ces principes de manière opérationnelle : Il montre comment configurer *sbuild* pour construire des paquets dans des chroots propres, puis comment lancer des tests spécifiques à OpenCL (via ROCm, pocl, etc.) en mode podman+rocm ou unshare, lancer via autopkgtest le test nommé (--test-name=opencl-ROCm), gérer les dépendances (pkg-rocm-tools, clinfo) et rendre les tests skippables si besoin [43]. Ainsi, mon wiki est un exemple concret de workflow Debian : création d'un chroot via sbuild, build isolé, automatisation des tests runtime avec autopkgtest, et utilisation de conteneurs pour garantir un environnement fidèle et sûr. Cela montre la force des wikis Debian pour diffuser des workflows techniques avancés, tout en restant éditables, interconnectés avec la documentation globale, et en constante évolution.

2.7 Les Patch et Bugs reports chez Debian

Les patchs :

Dans Debian, les patchs sont des fichiers de corrections (.patch ou .diff) générés avec diff et appliqués via quilt, afin de modifier le code source upstream sans altérer directement l'archive d'origine. Ils sont principalement utilisés pour corriger des bugs, adapter le logiciel aux standards Debian ou ajouter des fonctionnalités spécifiques avant publication [44]. Le format standard actuel est 3.0 (quilt) : chaque patch est stocké dans le dossier debian/patches et la liste des patchs à appliquer figure dans debian/patches/series. Cette organisation modulaire respecte la norme DEP-3 pour inclure une documentation claire dans chaque patch, ce qui facilite la relecture, la maintenance et la soumission aux développeurs upstream [45][46].Pour créer un patch, on installe et on configure quilt (souvent via un alias dquilt) de sorte à ce qu'il travaille dans debian/patches. On utilise ensuite quilt new nom.patch pour démarrer un patch, quilt add ou quilt edit pour modifier les fichiers concernés, puis quilt refresh pour enregistrer les changements. On complète enfin l'en-tête via quilt header -e afin de respecter DEP-3. Si l'on doit réviser un patch existant, l'approche consiste à « popper » jusqu'à ce patch, corriger le contenu, et effectuer un refresh suivi d'un nouveau header [47] [48].Lors de la construction du paquet avec dpkg-source ou dpkg-buildpackage, l'application automatique de tous les patchs listés dans series intervient avant la compilation, avec une éventuelle suppression après compilation si besoin. En cas de modifications imprévues, dpkg-source peut échouer ou générer un patch automatiquement selon les options utilisées (--auto-commit, etc.) [49].Ce système de patchs Debian offre trois avantages principaux : il permet de maintenir les modifications séparément des sources upstream, il assure une traçabilité claire et une gestion modulaire des correctifs, et il simplifie les mises à jour ou la contribution vers l'upstream (pull request ou

merge request), ce qui permet de retirer un patch une fois que la correction est intégrée officiellement [50].

En pratique voici la procédure pour les patchs que j'ai appliqué :

	Dans un premier temps vérifié que l'on est sur la bonne branche et des vérifications :
gitk –all	
	En principe on se place sur la branche master, si ce n'est pas le cas :
git checkout master	
	Passage sur la branche patch queue : pq pour patch queue
	Faire les modifications nécessaires
gbp pq import	
git commit	
gbp pq rebase	
gbp pq export	Pour sortir de la patch queue
gbp pq rebase	
Sbuild	

Les Bugs reports :

Dans l'univers de Debian, signaler un bug est bien plus qu'un simple acte technique : c'est une contribution directe à l'amélioration d'une distribution utilisée par des millions de personnes à travers le monde. Lorsqu'un utilisateur rencontre un dysfonctionnement ou un comportement inattendu dans un paquet logiciel, il peut le faire remonter via un système dédié appelé le BTS – pour *Bug Tracking System*. Ce système, à la fois ouvert, transparent et collaboratif, permet de suivre chaque problème signalé, d'en discuter, et surtout de le corriger. L'idée n'est pas ici de demander de l'aide personnelle, mais bien de participer à la stabilité et à la qualité de Debian pour l'ensemble de la communauté [50][51]. Concrètement, tout commence par une vérification : est-ce que le bug que j'ai rencontré a déjà été signalé ? C'est une étape clé, car cela évite de créer des doublons inutiles. Pour cela, l'utilisateur peut consulter le site bugs.debian.org ou utiliser l'outil *reportbug*, qui est spécialement conçu pour guider pas à pas dans ce processus [51][52]. Si le problème semble inédit, alors vient le moment de rédiger un bugreport. Celui-ci doit être le plus clair et complet possible : quelle version du paquet est concernée, dans quelles conditions le bug se manifeste, à quoi on s'attendait-on à voir, et qu'a-t-on observé à la place ? Si possible, on décrit les étapes précises pour reproduire le problème, on ajoute les messages d'erreur, et on reste factuel mais précis [53][54]. Pour envoyer ce rapport, deux méthodes s'offrent à la personne qui le signal. La première, plus directe, consiste à envoyer un email structuré à l'adresse *submit@bugs.debian.org*, avec quelques en-têtes obligatoires comme *Package:, Version:,* et éventuellement Severity: pour indiquer la gravité du bug. La seconde – et souvent la plus simple – consiste à passer par *reportbug*. Cet outil interactif guide l'utilisateur à travers une série de questions, récupère automatiquement les informations techniques nécessaires (comme la version du paquet ou les journaux d'erreurs), et génère le message final. En quelques minutes, le rapport est envoyé, et une confirmation est envoyée [50][52].Une fois le bug signalé, il reçoit un numéro unique dans le BTS et peut être consulté, suivi et commenté. Le mainteneur du paquet est immédiatement notifié.

Celui-ci peut alors qualifier le bug, répondre s'il a besoin de précisions, ou planifier sa correction. Selon les cas, le bug peut être jugé "non reproductible", nécessiter plus d'informations ("moreinfo"), ou bien être corrigé dans une mise à jour future. Une fois corrigé, le bug est fermé – souvent automatiquement – dès que la version corrigée du paquet est publiée. Le processus est ainsi entièrement traçable, du signalement à la résolution [51][53]. Ce qui fait la force du système de bug Debian, c'est qu'il repose sur la participation volontaire et ouverte. Nul besoin d'être développeur ou expert : toute personne utilisant Debian peut signaler un bug. Et plus les rapports sont nombreux et bien construits, plus la distribution gagne en robustesse. Cette démarche s'inscrit dans l'esprit du logiciel libre : un écosystème où chacun peut contribuer à sa mesure. En somme, rapporter un bug dans Debian, c'est contribuer activement à la qualité et à la pérennité d'un projet communautaire d'envergure mondiale [50][51][52][53][54].

En pratique quelles sont les commandes à utiliser pour réaliser un bug report :

Tout d'abord installer reportbug : **apt install reportbug** Et le configurer en suivant les étapes données. Une fois installé et après avoir identifié le bug : **reportbug <pkg>** On énonce le numéro de la version, puis, on indique à l'aide d'un titre ce qu'il retourne du bug. On suit ensuite la procédure en indiquant la sévérité, le contexte autour du bug, les commandes qui ont été faites pour y arriver. Les logs d'erreurs et ce qu'on est censé obtenir dans le cas où le bug est résolu.

3. Les supports matériels AMD et NVIDIA

3.1 AMD ROCm

La plateforme ROCm (*Radeon Open Compute*), développée par AMD, est un environnement open-source pensé pour exploiter pleinement les capacités de calcul hétérogène des GPU modernes [55]. Contrairement à des solutions propriétaires, ROCm mise sur la transparence du code et sur une architecture modulaire qui facilite la personnalisation et l'adaptation aux besoins spécifiques de la recherche. Son cœur technologique repose sur HIP (*Heterogeneous-Compute Interface for Portability*), une interface qui permet de réutiliser et de porter facilement du code CUDA grâce à l'outil *hipify* et au compilateur hipcc [56]. En parallèle, ROCm embarque un runtime OpenCL (rocm-opencl-icd) [57], qui assure la compatibilité avec les applications OpenCL standards. L'écosystème est enrichi par plusieurs bibliothèques optimisées : rocBLAS pour les opérations matricielles, rocFFT pour les transformées de Fourier, et MIOpen pour les primitives de deep learning, offrant ainsi une alternative ouverte à cuDNN [55]. Côté outils, AMD fournit tout le nécessaire pour diagnostiquer et surveiller l'état des GPU : rocminfo pour obtenir la liste et les caractéristiques des agents GPU présents sur le système [58], rocm-smi pour suivre en temps réel la consommation énergétique, la température et la fréquence du processeur graphique [59], ou encore rocm-test-launcher et le paquetage *pkg-rocm-tools*, qui regroupent des utilitaires de validation et de benchmarking [59]. Tous ces éléments reposent sur le pilote amdgpu du noyau Linux, complété par le composant KFD (*Kernel Fusion Driver*), qui gère la communication et la synchronisation entre CPU et GPU [55]. Le support matériel de ROCm reste sélectif. Du côté des datacenters et du HPC, AMD garantit une compatibilité complète avec ses accélérateurs Instinct : MI100, MI200/MI250X et MI300 [60][61]. Certaines cartes professionnelles comme les Radeon Pro W6800 (RDNA2) et W7900 (RDNA3) bénéficient aussi d'un support officiel, mais celui-ci reste en cours de consolidation [62]. Pour les GPU grand public basés sur RDNA2 ou RDNA3, le support est uniquement expérimental : il peut fonctionner dans un contexte de développement personnel, mais n'est pas considéré comme stable pour un environnement de production [60]. Sur le plan stratégique, ROCm présente plusieurs atouts. Son caractère ouvert en fait une solution privilégiée pour les projets exigeant transparence, auditabilité et indépendance technologique [55]. De plus, il s'intègre avec des standards ouverts comme SYCL ou oneAPI, ce qui permet d'écrire des codes capables de s'exécuter sur différentes architectures (Intel, AMD, NVIDIA) [63]. Cependant, certaines limites persistent. Le support officiel est encore réservé aux GPU haut de gamme, ce qui limite son adoption par les utilisateurs équipés de cartes grand public [60]. Les

frameworks d'IA comme PyTorch ou TensorFlow ne sont pas toujours compatibles sans ajustements spécifiques, et la documentation communautaire reste inégale sur ce point [64]. Enfin, ROCm ne dispose pas d'un support Windows natif complet : l'essentiel de son utilisation se fait sous Linux, avec un support expérimental via WSL2 qui n'offre pas encore des performances comparables à un environnement Linux natif [64].

3.2 NVIDIA CUDA

CUDA (*Compute Unified Device Architecture*), lancé par NVIDIA en 2006, est devenu la référence mondiale du calcul parallèle sur GPU [65]. C'est une plateforme propriétaire qui permet aux développeurs d'exploiter pleinement la puissance des GPU NVIDIA dans des domaines comme le deep learning, la simulation scientifique ou la finance [66]. Le modèle de programmation repose sur CUDA C/C++, avec des interfaces pour Python et Fortran [67]. Les programmes sont traduits en PTX, un langage intermédiaire propriétaire optimisé par les pilotes NVIDIA [68]. L'écosystème est riche : cuBLAS pour l'algèbre linéaire, cuFFT pour les transformées de Fourier, cuDNN pour le deep learning, TensorRT pour l'inférence IA [69], ainsi que des outils comme Nsight Systems pour profiler les performances [70]. CUDA bénéficie d'un support matériel étendu : de l'architecture Kepler (2012) aux GPU actuels A100 et H100, en passant par les GeForce et RTX [71]. Les Tensor Cores, intégrés depuis Volta, accélèrent fortement les calculs IA [72][73]. Cette maturité logicielle et matérielle explique pourquoi PyTorch et TensorFlow offrent un support optimisé CUDA dès leur installation [73]. Mais ce succès a un revers : la dépendance. Avec plus de 80 % de parts de marché dans le calcul scientifique et l'IA [75], NVIDIA a créé un verrouillage technologique (*vendor lock-in*). Le code CUDA est difficile à porter sur d'autres architectures, même via HIP [77], et les bibliothèques HPC/IA sont souvent développées pour CUDA en priorité. Les prix des GPU haut de gamme peuvent dépasser 30 000 €, et la forte demande en IA amplifie ce phénomène [76]. Ce monopole freine l'adoption de standards ouverts comme SYCL ou OpenCL [75], limitant la flexibilité et la souveraineté technologique des laboratoires et entreprises. La concentration du marché autour d'un seul acteur pose un risque stratégique, notamment en cas de tension sur la production ou d'évolution des politiques commerciales.

Conclusion de la partie 2 :

En conclusion de cette partie, l'ensemble des notions nécessaires à la compréhension de mes missions ont été développées, expliquées dans cette partie. Notamment sur Debian, ce qu'est Debian, ses enjeux sociaux et éthiques etc. Dans cette partie, il a également été développé les notions et enjeux autour de AMD ROCm, point central au sein de ma mission au sein de Synchrotron-SOLEIL. Mais également ceux de NVIDIA CUDA, qui, sans cette contradiction, ne mettraient pas en lumière la nécessité de tester la compatibilité des paquets Debian utiles à SOLEIL sur les GPU AMD. De plus, les versions et le concept de stabilité d'un paquet, la Debian mailing list de Debian Science, Debian Tracker, Debian package ,Debian Salsa Git , Debian wiki et enfin les patch et bugs reports chez Debian ont pu être développés au profit de la compréhension complète des notions qui interviennent au sein de mes missions.

Partie 3 : Mes missions

Sommaire

1. Les procédures et setup
    - 1.1 Mise en place de dgit - -gbp sbuild
    - 1.2 Mise en place du chroot pour sbuild
    - 1.3 Mise en place de la procédure spécifique pour les autopkgtest
    - 1.4 Mise en place des containers podman+rocm
    - 1.5 Les procédures de patchs
    - 1.6 Les procédures de bugs report
    - 1.7 Les procédures de correction à l'upstream
2. Les paquets Debian à traiter et leurs traitements avec les problématiques rencontrées
    - 2.1 Le langage Opencl
    - 2.2 Les paquets faisant de l'OpenCL
    - 2.3 Les paquets ne faisant pas d'OpenCL



Lorsque je suis arrivé à Synchrotron-SOLEIL, Emmanuel et Fred m'ont expliqué l'environnement Debian et toutes ses spécificités, procédures, etc., que j'ai pu retranscrire dans toute la partie 2 intitulée Debian. Afin de comprendre tous les enjeux et les notions nécessaires à la compréhension précise de cette partie, je pense qu'il est vital d'avoir lu la partie 2. J'ai eu pendant plusieurs semaines du mal à appréhender l'infrastructure Debian. Cela faisait énormément d'informations à mettre en pratique. J'ai tout d'abord fait l'inventaire des paquets à implémenter, présents dans le Git salsa de la division Grades de Synchrotron-SOLEIL. L'une de mes premières missions a été d'une part de les regrouper au sein d'un tableau et de chercher si ceux-ci faisaient de l'OpenCL et surtout s'ils possédaient des autopkgtest ainsi que des tests unitaires OpenCL, voire plus rarement des autopkgtest pour AMD ROCm. Frédéric-Emmanuel PICCA et Emmanuel FARHI, m'ont donné pour mission de réaliser un wiki qui contiendra l'ensemble des procédures, mises en pratique, paquets implémentés, etc. et ce qui a été fait durant ces 4 mois. J'ai me suis tout d'abord inscrit sur la Debian-Science-Team et le Debian wiki ainsi que la Debian-pan-maintainers. J'ai dans un premier temps fait un mail sur la mailing list de Debian-Science pour présenter la mission qui m'a été confiée. J'ai ensuite commencé mes travaux en commençant par l'établissement des procédures nécessaires à la réalisation de cette mission. Ces procédures ont été mises en place au fur et à mesure de ma mission. Par exemple les procédures portant sur la mise en place de bug report à l'upStream ou encore de requête de fusion, ont été mises en place que vers le milieu ou vers la fin de mon stage car je n'en n'avais pas encore eu besoin. A la différence des procédures de Setup comme celles de *sbuild, dgit – gbp sbuild* , *commandes d'autopkgtest* et enfin des commandes sur le container podman+rocm vitales à la réalisation de la mission. Cette idée de construction du setup et des procédures nécessaires à la réussite de ma mission fait partie intégrante de mon travail et est donc considérée comme telle. La phase de recherche et de mise en place de ces environnements est très chronophage et peut s'avérer assez technique pour des éléments n'étant pas familiers avec le packaging Debian (raison pour laquelle j'insiste sur la lecture de la partie 2). De plus, la phase de recherche des tests OpenCL et des paquets faisant de l'OpenCL ainsi que l'alimentation du wiki sont des parties importantes de mes travaux. La première partie des éléments que j'ai pu réaliser concerne les procédures et setup, je fournis des explications sur l'implémentation de celles-ci, indispensables à sa compréhension. La deuxième concerne l'implémentation directe des paquets Debian, qu'ils fassent de l'OpenCL ou non, cela constitue la majorité de mon temps de travail. La deuxième partie impose des explications et une mise en contexte. Dans la pratique, cela s'accompagne d'un grand travail de recherche de mon côté pour appréhender les logs d'erreurs. Même si le premier mois a été largement consacré à la première partie et à sa réalisation.

1. Les procédures et setup

1.1 Mise en place de dgit - -gbp sbuild

La commande *dgit --gbp sbuild* sert à construire un paquet Debian directement à partir des sources Git, en utilisant gbp pour gérer les branches. Elle combine dgit (outil pour travailler avec Debian et Git) et sbuild (pour construire le paquet dans un environnement isolé). Dans le cadre de mes missions, c'est un point central car il permet de réaliser les autopkgtests et de construire le paquet et de veiller à son bon fonctionnement. C'est très utile car cela permet de vérifier si l'implémentation des autopkgtest est bonne, de voir la nature des potentielles erreurs. De valider les modifications qui ont été faites. Cependant il n'y a pas que ce type de commande pour réaliser une construction du paquet. Cependant, il faut pour cela mettre en place un chroot que je développerai dans la partie suivante. Le but ici est d'établir la procédure à suivre afin que cela fonctionne pour la construction de l'ensemble des paquets. La mise en place de cette procédure m'a pris plusieurs semaines car je n'avais encore jamais fait de packaging Debian. J'ai été confronté à l'ensemble de l'infrastructure Debian, mes maîtres de stages étaient très à l'écoute et bienveillants sur ces aspects. Car ils s'avaient que le packaging Debian était encore méconnu et encore obscur pour moi. Le problème avec la commande *dgit –gbp sbuild* est qu'elle exige une version de travail dite propre avec aucun élément non commité , ni push. Cela signifie qu'avant chaque lancement de la

commande, l'utilisateur doit, soit push ses modifications sur l'upstream (ce qui est malvenu s'il y a des erreurs), soit faire un commit dans lequel il y a les modifications apportées. Donc, d'un point de vue pratique, cela pose quelques soucis car, cela impose de grandes contraintes. Ce qui n'est en réalité pas un mal car cela oblige l'utilisateur à travailler proprement.

Par exemple dans le cas de *ufo-filters*, si nous voulons vraiment lancer la via la commande *dgit --gbp sbuild* alors il faut nettoyer et reset ce qu'on a fait car, comme dit précédemment, *dgit* souhaite des répertoires propres (sans modifications locale) et c'est un concept qui, lorsque l'on ne connait pas la commande, peut être assez perturbant.

dgit --gbp sbuild *dgit clone ufo-filters (cela permet de cloner le répertoire du pkg)* *dpkg -l | grep ufo-filters* *sudo apt update (met à jour l'ensemble des paquets)* *sudo apt upgraded*	Pour nettoyer tout et tout reset pout le dgit si jamais cela est gênant : *git clean -xdf* *git reset --hard* *dgit --gbp sbuild*

1.2 Mise en place du chroot pour sbuild

Le chroot est utilisé comme un environnement de travail. Le but est de construire le paquet dans un environnement temporaire dans lequel on a fixé les paramètres. Le container que l'on utilise, les options des autopkgtest, les proxys, etc. Nous l'utilisons en tant qu'environnement de développement isolé afin que nous puissions développer avec différentes versions de bibliothèques sans affecter le système global. Nous pourrons installer des dépendances spécifiques à un projet sans altérer le système principal. Certaines options sont nécessaires car nous utilisons des dockers. Et pour cela il faut les configurer. Tout d'abord il faut créer un fichier .config dans lequel un dossier *sbuild* est créer. Ensuite, dans ce même dossier est créé un fichier config.pl. Le .pl signifie que le fichier est codé en Perl. Il s'agit d'un langage de programmation créer en 1987, ce langage, interprété, s'inspire des structures de contrôle et d'impression du langage C, mais aussi de langages de scripts *sed, awk et shell* (sh). Ce qui m'a posé le plus de problème est le proxy de Synchrotron-SOLEIL. En effet, le paquet ufo-filters nécessitait l'utilisation d'un proxy pour collecter des données via un lien vers le CNRS. La construction du fichier config.pl m'a pris plusieurs semaines en raison de recherches sur la manière de procéder pour contourner le proxy. Chaque fois que je lançais le test cela le faisait échouer. J'ai d'abord essayé de le désactiver manuellement à chaque lancement de test. Puis de le désactiver globalement, sans succès. Avant que je me rende compte que la configuration du proxy devait se faire dans le fichier config.pl. En effet le fichier est en Perl, et ne sachant pas coder en Perl, le premier mur fut la syntaxe. J'ai cherché, échangé avec Frédéric-Emmanuel PICCA pour comprendre en détail comment construire ce fichier si important pour la suite. Nous avons chacun nos missions, je ne pouvais pas le harceler avec mes questions. Il fallait être autonome même si cela prenait plus de temps. Après avoir construit 80% du fichier mais restant encore et toujours le proxy à gérer, Fred m'a donné la syntaxe et la manière d'implémenter la configuration du proxy dans le fichier. Le maître mot est l'autonomie, et la compréhension par soi-même. C'est lorsque que l'on cherche par sois même que l'on comprend et que l'on apprend le mieux.

Voici le code complet :

Fichier config.pl

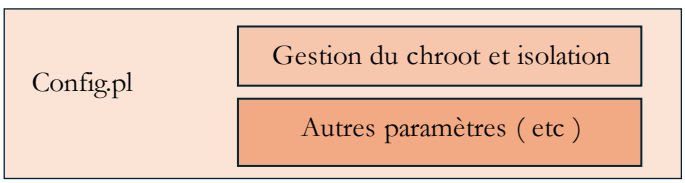
```
$chroot_mode = "unshare";
$unshare_mmdebstrap_keep_tarball = 1;
$run_autopkgtest = 1;
$external_commands = {
  "chroot-update-failed-commands" => [ [ '%SBUILD_SHELL' ] ],
  "build-deps-failed-commands" => [ [ '%SBUILD_SHELL' ] ],
  "build-failed-commands" => [ [ '%SBUILD_SHELL' ] ],
};
$unshare_mmdebstrap_extra_args = [
  "*" => [ '--aptopt=Acquire::http { Proxy "http://195.221.0.35:8080"; }',
           '--aptopt=Acquire::https { Proxy "http://195.221.0.35:8080"; }',
           '--customize-hook=echo "export http_proxy=http://195.221.0.35:8080" > "$1"/etc/environment',
           '--customize-hook=echo "export https_proxy=http://195.221.0.35:8080" > "$1"/etc/environment',
]
];
$autopkgtest_opts=['--shell-fail', '--apt-upgrade', '--env=https_proxy=http://195.221.0.35:8080', '--', 'podman+rocm', 'localhost/rocm/debian:%r']
```

Je vais expliquer la construction de ce fichier et la raison pour laquelle sans documentation il peut être assez ardu de le construire à la main et ce pour la première fois. Il spécifie d'abord *$chroot_mode = "unshare";* optant pour une isolation sans beaucoup de contraintes plutôt qu'un chroot traditionnel, offrant ainsi une meilleure sécurité et reproductibilité sans la surcharge d'une virtualisation complète. La directive *$unshare_umdebstrap_keep_tarball = 1;* préserve l'archive (*tarball*) du chroot après sa création, évitant de le reconstruire intégralement à chaque exécution et accélérant ainsi les builds ultérieurs – un *tarball* étant une archive compressée (comme .tar.gz) regroupant fichiers et répertoires pour une distribution ou une sauvegarde efficace.

Les réglages réseau sont vitaux, c'est d'ailleurs ces mêmes réglages qui m'ont autant posé de problème, *$unshare_umdebstrap_extra_args* injecte systématiquement un proxy (*195.221.0.35:8080*) via des options APT (Acquire::http/https) et des *hooks* personnalisés qui écrivent les variables *http_proxy* et *https_proxy* dans */etc/environment*, garantissant que toutes les opérations (téléchargements de dépendances, tests) respectent les contraintes de votre infrastructure d'entreprise. La gestion des erreurs est optimisée par *$external_commands*, qui ouvre un shell interactif (%SBUILD_SHELL) en cas d'échec aux étapes critiques (mise à jour du chroot, résolution des dépendances ou build), facilitant le débogage sans recréer l'environnement. Enfin, *$run_autopkgtest = 1* active une validation automatisée via autopkgtest après chaque build réussi, avec des options ciblant les conteneurs Podman (podman+rocm) et les images GPU personnalisées (localhost/rocm/debian:%r), incluant une mise à jour des paquets (--apt-upgrade) et la propagation du proxy pour les tests. J'expliquerai dans la partie dédiée pourquoi il a choisi d'utiliser *podman+rocm* comme containers plutôt que *qemu*

À titre personnel, j'ai eu du mal à faire la différence entre un chroot et un fichier de config.pl pour le sbuild. Je trouve important de bien préciser les choses pour bien comprendre ce que j'ai fait au tout début de mon stage. Un chroot (change root) est un mécanisme d'isolation qui crée un système de fichiers virtuel séparé pour l'exécution de processus, limitant leur accès au reste du système. On crée un environnement virtuel pour la construction du paquet et des autopkgtest avec certains containers en places. A la différence, le config.pl est un fichier de configuration qui pilote le comportement d'outils comme sbuild, incluant des paramètres pour la création et la gestion des environnements chroot (ou autres modes d'isolation comme unshare).

Voici un bref schéma pour se représenter la situation :



Cette configuration sert mes missions en standardisant les builds de paquets Debian avec intégration transparente du proxy, en automatisant les tests dans des environnements GPU (ROCm), et en fournissant des outils de diagnostic. Tout ceci est optimisé pour l'infrastructure dans laquelle j'évolue.

1.3 Mise en place de la procédure spécifique pour les autopkgtest

L'objectif est de faire fonctionner cette commande quoi qu'il arrive :

```
/usr/bin/autopkgtest -B ufo-filters --test-name=reconstruct -- unshare --release unstable --arch amd64   (1)
```

Procédure pour faire fonctionner (1)

```
sudo apt install  mmdebstrap

sudo nano /etc/subuid

longeac:100000:65536

sudo nano /etc/subuid

longeac:100000:65536

/usr/bin/autopkgtest -B ufo-filters --test-name=reconstruct -- unshare --release unstable --arch amd64
```

Cette commande lance *autopkgtest* sur le paquet *ufo-filters*. J'utilise l'option *-B* pour préciser que le paquet ne doit pas être reconstruit, mais uniquement testé dans sa version binaire. L'option *--test-name=reconstruct* me permet de cibler un test particulier, ici nommé « reconstruct », au lieu de lancer toute la suite de tests. L'environnement d'exécution est fourni par le backend *unshare,* qui utilise les namespaces Linux ainsi que les mappings que j'ai configurés précédemment. L'option *--release unstable* indique que le mini-système Debian utilise sera basé sur la version *unstable*, et *--arch amd64* précise l'architecture matérielle. Voici quelques explications plus approfondies sur la raison de l'utilisation de cette procédure pour obtenir cette commande. J'utilise cette commande pour éviter d'utiliser *chroot* avec toutes ses contraintes. *Unshare* est beaucoup plus flexible là où *chroot* est rigide.

Cependant, je ne peux lancer cette commande sans les prérequis suivants. La première étape consiste à installer *mmdebstrap* avec la commande *sudo apt install mmdebstrap*. C'est un bootstrapper Debian moderne qui sait créer un *chroot* (ou un tarball *rootfs*) rapidement et sans privilèges. C'est exactement ce dont a besoin *autopkgtest* lorsqu'il utilise le backend *unshare* : il doit disposer d'un mini-système Debian « jetable » dans lequel les tests vont être exécutés. Ici, ce système minimal sera basé sur la distribution Debian *unstable*. Ensuite, je modifie le fichier */etc/subuid* en l'ouvrant avec un éditeur comme nano. Ce fichier configure la gestion des user namespaces dans Linux. En effet, le backend *unshare* repose sur ce mécanisme pour isoler les tests. Pour me permettre de me comporter comme un utilisateur root à l'intérieur de l'environnement isolé, il faut établir un mapping entre des identifiants virtuels (UID) internes au namespace et de vrais identifiants sur mon système hôte. C'est pourquoi j'ajoute la ligne suivante : *longeac:100000:65536*. Cette configuration signifie que je pourrais utiliser une plage de 65 536 identifiants d'utilisateurs dans le namespace, qui commenceront à partir de l'UID 100000 sur le système hôte. Concrètement, à l'intérieur du namespace, l'utilisateur root (UID 0) sera mappé à l'UID 100000 sur ma machine réelle. Le mapping se poursuit jusqu'à 165536. Cela permet une isolation sécurisée : même si un processus pense être root dans le namespace, en réalité il ne l'est pas sur mon système. Il est recommandé de faire la même opération dans /etc/subgid pour les groupes, afin que les identifiants de groupes soient également correctement mappés. On peut voir ça comme un dictionnaire de traduction entre deux langues. De manière très

imagée, à l'intérieur du namespace, tout le monde parle une langue simplifiée où root est 0 et les autres utilisateurs suivent à partir de 1, 2, 3… Mais pour que le système hôte comprenne et contrôle ce qui se passe, il faut une traduction. Le fichier */etc/subuid* joue le rôle de dictionnaire : il dit par exemple que « root (0) » dans cette langue spéciale correspond en réalité à « utilisateur 100000 » sur l'hôte. Ainsi, même si le processus croit être root, il est en fait enfermé dans une cage bien définie, sans aucun privilège supplémentaire sur la vraie machine.En fait, */etc/subuid* et */etc/subgid* servent à établir une correspondance sécurisée entre les identifiants virtuels des namespaces et ceux de l'hôte. Ce mécanisme est indispensable pour que des outils comme autopkgtest puissent utiliser le backend unshare et lancer des tests dans un environnement isolé, sans jamais mettre en danger le système principal. En résumé, ces étapes mettent en place un environnement isolé basé sur Debian *unstable* grâce à *mmdebstrap* et aux user namespaces. J'obtiens alors un espace de 65 536 identifiants pour simuler un système complet à l'intérieur du namespace. Enfin, autopkgtest y lance le test « reconstruct » fourni par le paquet ufo-filters, de manière sécurisée et indépendante du système principal. IL s'agit de construire un environnement virtuel tout entier.

1.4 Mise en place des containers podman+rocm

```
$ sudo rocm-podman-create rocm/debian:unstable.img

##To check the image use :
$ podman images

##To check the container :
$ podman ps

##Configure the system for GPU-in-container use
$ rocm-podman-setup -u <user>

##To run autopkgtest :

# Run rocrand's autopkgtests (from the official archive) in that image
$ autopkgtest -B <pkg> -- podman+rocm rocm/debian:unstable.img
```

Procédure pour la création et l'utilisation du container podman+rocm

Eléments à ajouter dans config.pl pour le fonctionnement du container podman+rocm :

```
$autopkgtest_opts=['--shell-fail', '--apt-upgrade', '--env=https_proxy=http://195.221.0.35:8080', '--', 'podman+rocm', 'localhost/rocm/debian:%r']
```

Voici quelques explications plus approfondies sur la raison de l'utilisation de cette procédure. Désormais, après avoir mis en place les environnements virtuels nécessaires à l'exécution des autopkgtest, il est nécessaire d'offrir un container optimisé pour ROCm. Alors *podman+rocm* est parfaitement adapté à notre besoin. Il faut donc établir et expliquer la procédure et nos choix qui le feront fonctionner sans accroc. Pour tester des paquets Debian nécessitant un accès au GPU AMD dans un environnement isolé, j'utilise Podman avec le backend *podman+rocm* qui est une approche recommandée. Cette méthode est particulièrement adaptée pour les autopkgtest ou la construction de paquets, qui dépendent de ROCm (Radeon Open Compute), le framework d'AMD pour le calcul parallèle sur GPU. La procédure commence par la création d'une image Debian instable configurée pour ROCm à l'aide de la commande suivante : *sudo rocm-podman-create rocm/debian:unstable.img*. Cette commande génère une image Podman prête à l'emploi, intégrant les composants nécessaires pour utiliser le GPU AMD dans un conteneur. Le terme *podman+rocm* fait référence à l'utilisation de Podman avec des conteneurs configurés pour accéder directement aux GPU AMD via ROCm. Ce n'est pas un Docker, mais le conteneur créé par Podman fonctionne de manière très simi-

laire à un conteneur Docker : il est isolé, portable et contient tout ce dont l'application a besoin pour s'exécuter, y compris l'accès au GPU. En résumé, Podman+ROCm = conteneur Podman optimisé pour ROCm, tandis que Docker serait une alternative générique mais moins adaptée pour l'accès direct au GPU rootless. L'outil *rocm-podman-create* fait partie du paquet *rocm-podman-support*, qui contient également des utilitaires comme *rocm-podman-setup* pour configurer le système hôte afin de permettre l'accès au GPU depuis les conteneurs. Une fois l'image créée, il est possible de vérifier sa présence avec la commande : *podman images*. Cette commande liste toutes les images disponibles localement. Pour vérifier les conteneurs en cours d'exécution, on peut utiliser *: podman ps*. Avant d'exécuter des tests nécessitant le GPU, il est essentiel de configurer le système hôte pour autoriser l'accès au GPU depuis les conteneurs. Cela se fait en exécutant : *rocm-podman-setup -u <user>*. Cette commande ajuste les permissions et configure les dispositifs nécessaires, tels que */dev/dri* et */dev/kfd*, pour l'utilisateur spécifié. Pour lancer les tests autopkgtest dans l'image Podman configurée pour ROCm, j'utilise la commande suivante : *autopkgtest -B rocrand -- podman+rocm rocm/debian:unstable*. Il faut également configurer certaines options dans le fichier *config.pl* pour personnaliser le comportement des tests :

$autopkgtest_opts = ['--shell-fail', '--apt-upgrade', '--env=https_proxy=http://195.221.0.35:8080', '--', 'podman+rocm', 'localhost/rocm/debian:%r'];

- *Commande d'utilisation de podman+rocm en commande line:*

Autopktest -B <pkg> -- podman+rocm rocm/Debian:image.img

Ces options permettent de gérer les mises à jour des paquets, de définir des variables d'environnement et de spécifier le backend à utiliser pour les tests. Mon choix de Podman avec le backend *podman+rocm* plutôt que d'autres solutions comme QEMU ou Docker repose sur plusieurs avantages spécifiques à l'utilisation du GPU avec ROCm. Podman, en mode rootless, permet d'exécuter des conteneurs sans nécessiter de privilèges élevés, ce qui réduit les risques de sécurité. Le terme *podman+rocm* fait référence à l'utilisation de Podman avec des conteneurs configurés pour accéder directement aux GPU AMD via ROCm. Ce n'est pas un Docker, mais le conteneur créé par Podman fonctionne de manière très similaire à un conteneur Docker : il est isolé, portable et contient tout ce dont l'application a besoin pour s'exécuter, y compris l'accès au GPU. Le backend *podman+rocm* est spécialement conçu pour intégrer les dispositifs GPU nécessaires au fonctionnement de ROCm, tels que */dev/dri* et */dev/kfd*, directement dans le conteneur. Cela facilite l'accès au GPU sans configurations complexes supplémentaires. En revanche, QEMU, bien qu'étant une solution de virtualisation puissante, ne prend pas en charge nativement l'accès direct au matériel GPU dans un environnement rootless. Cela nécessiterait des configurations supplémentaires et pourrait ne pas offrir les performances optimales pour les applications utilisant ROCm. Mon utilisation de Podman avec le backend *podman+rocm* offre une solution pour tester des paquets Debian nécessitant un accès au GPU AMD, en simplifiant la configuration et en assurant une intégration transparente avec le système hôte.

1.5 Les procédures de patchs

```
$gitk --all

On est sur la branche master actuellement
Passage sur la branche patch queue

$gbp pq import

(fait les modifs)
$git commit

$gbp pq export
$git gui

Commit les fichiers

$sbuild (pour relancer la construction)
```

La procédure du patch est expliquée dans la Partie 2 Section 2.7

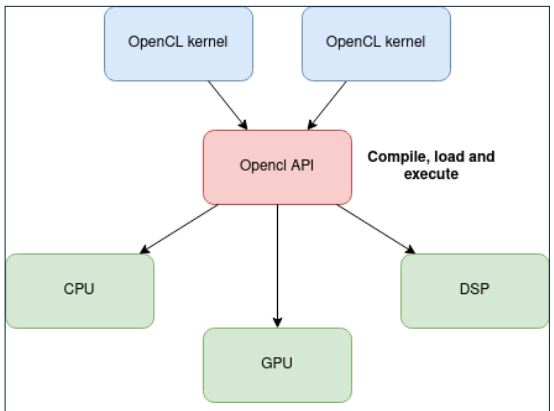
1.6 Les procédures de correction à l'upstream

En fait, *upstream branch* correspond à la branche qui contient le code source original du logiciel, tel qu'il est développé par les auteurs du projet, en dehors de toute modification spécifique à Debian. Elle représente donc l'état "pur" du logiciel, avant son adaptation ou son empaquetage pour la distribution. Son utilité est de servir de référence : en tant que mainteneur Debian, ils peuvent facilement comparer le code qu'ils empaquettent avec la version amont, suivre les évolutions et mises à jour du projet, et intégrer les correctifs ou nouvelles fonctionnalités publiés par les développeurs originaux du logiciel.

Je ne suis pas Debian mainteneur, je contribue ponctuellement à des projets. (ceux décrits dans le tableau dans la section 2.2). Pour des raisons de sécurité, seules les personnes autorisées à faire des modifications peuvent le faire. Les auteurs eux-mêmes, les Debian Maintainers, Debian Dev. Alors, à mon échelle, je ne peux pas push mes modifications implémenter en local sur la version que j'ai récupérée en faisant un *debcheckout* <*pkg*>. Je dois donc faire un *merge request*. Soit une requête de Fusion à l'Upstream pour qu'il m'autorise à psuh mes modifications sur la branche master. En pratique, sur le git-salsa de Debian. Nous avons un onglet, Merge request, nous recherchons le paquet que nous souhaitons et suivons ensuite les étapes. Une fois celui-ci envoyé, il faut que le Maintainer du paquet ou l'auteur lui-même accepte la demande de merge (Fusion), pour qu'elle soit intégrée sur le Git.

2. Les paquets Debian à traiter et leurs traitements avec les problématiques rencontrées

2.1 Le langage Opencl



Titre : Schéma de fonctionnement de l'API Opencl

Qu'est-ce que le langage OpenCL et à quoi sert-il ? OpenCL (Open Computing Language) est la combinaison d'une API et d'un langage de programmation dérivé du C, proposé comme un standard ouvert par le Khronos Group. OpenCL est conçu pour programmer des systèmes parallèles hétérogènes comprenant par exemple à la fois un CPU multi-cœur et un GPU. OpenCL propose donc un modèle de programmation se situant à l'intersection naissante entre le monde des CPU et des GPU, les premiers étant de plus en plus parallèles, les seconds étant de plus en plus programmables. OpenCL distingue le processeur hôte (processeur central faisant office de chef d'orchestre) des périphériques (CPU, GPU, ou autre) dont la mission est d'exécuter des noyaux de calcul intensifs. OpenCL distingue donc d'un côté l'application tournant sur le processeur hôte (et qui va appeler l'API OpenCL), et de l'autre côté les noyaux qui sont programmés en OpenCL-C (et dont la vocation est d'être exécutés sur les périphériques). [78]

2.2 Les paquets faisant de l'OpenCL

Lorsque je suis arrivé à Synchrotron SOLEIL, il m'a tout d'abord été demandé de faire l'inventaire des paquets que l'on pouvait implémenter et qui faisaient de l'Opencl ainsi que des autopktest Opencl. Cette liste m'a été donnée via le git salsa de Synchrotron-SOLEIL (Package SOLEIL). Frédéric-Emmanuel

PICCA m'a permis de créer mon propre wiki sur la liste Debian Wiki, portant sur l'OpenCL. Il y est décrit les paquets que je vais implémenter avec ceux qui font de l'Opencl et les autres non. Les procédures pour réaliser ces implémentations, portant sur les containers etc . Décrit dans la partie 1. Pour cela, j'ai notamment utilisé l'outil **grep -rn « txt »** en command line pour rechercher directement l'ensemble des paquets. Cette commande est une recherche avancée et permet de me donner la ligne et dans quel fichier se trouve le texte cherché, ici des commandes caractéristiques de la présence de code OpenCL.

En outre, avec ce procédé, j'ai pu identifier les paquets à traiter. Dont voici la liste exhaustive. Nous nous sommes aperçus que certains paquets ne faisaient en réalité pas d'OpenCL. Même si tout s'y prêtait au premier abord. Ces paquets sont barrés. Le tableau suivant répertorie, en faisant un résumé très grossier, l'ensemble des avancés durant ces 4 mois. J'expliquerai cas par cas ce qui a été fait, les freins majeurs qui ont pu être rencontrés.

Package	OpenCL unit tests	OpenCL autopkgtest	ROCm autopkgtest	Upstream	Etat
ufo-filters	O	O PASS	PASS ROCm-autopkgtest	Upstream-Ufo-Filters	**PUSHED**
nabu	O	O PASS	PASS	Upsream-Nabu	**NOT PUSHED**
pynx	O	O Added In local but Errors	Added In local but Errors	Upstream-Pynx	
~~pocl~~				~~Upstream-Pocl~~	**NO OPENCL CODE**
~~examl~~	PASS for Other tests NO OPENCL	PASS for Other tests NO OPENCL	PASS for Other tests NO OPENCL	~~Upstream-Examl~~	**NO OPENCL CODE | NOT PUSHED**
clblast	O	O Added POCL autopkgtest	Not pushed because of remaining errors.		**POCL PASS**
~~genx~~				~~Upstream-genx~~	**NO OPENCL CODE | NOT PUSHED**
clpeak		PASS	O PASS url du debci rocm	Upstream-Clpeak	**NOT PUSHED**
haskell-futhark	O	Added In local but Errors	O autopkgtest driver for ROCm		
sasmodels	O	O PASS	PASS	Upstream-Sasmodels	**NOT PUSHED**
hashcat	O	Added In local but Errors	Added In local but Errors	Upstream-Hashcat	
libhmbeagle	O	PASS	PASS	Upstream-Libhmbeagle	**PUSHED**
pyopencl	O	PASS	PASS	Upstream-Pyopencl	**NOT PUSHED**
compyle	O	PASS	PASS	Upstream-Compyle	**NOT PUSHED**
gpyfft	O	PASS	PASS	Upstream-Gpyfft	**PUSHED**
pyvkfft	O	PASS	PASS	Upstream-Pyvkfft	**PUSHED**
python-dtcwt	O	Added In local but Errors	Added In local but Errors	Upstream-Python-dtcwt	
~~xpra~~					**NO OPENCL CODE**

pysph	O	Added In local but Errors	Added In local but Errors	Upstream-Pysph	
pyfai	O	Added In local but Errors	Added In local but Errors	Upstream-Pyfai	
silx	O	PASS	PASS	Upstreal-Silx	**PUSHED**
python-xrt	O	Added In local but Errors	Added In local but Errors	Upstream-Python-xrt	
pytools					**NO OPENCL CODE**
Légende : O PASS L'autopkgtest était présent, à l'origine. Nous avions des erreurs. Puis, après l'implémentation il passe.	PASS L'autopkgtest était absent. Puis, après l'implémentation il passe.	O Présence du tests à l'origine faisant intervenir OpenCL	**NOT PUSHED** Les tests passent mais il n'est pas encore poussé	 Paquets ne faisant pas d'OpenCL	

Nous traiterons les paquets linéairement afin d'avoir une compréhension globale de ce qui a été fait.

Ufo-filters :

Dans un premier temps, nous copions le chemin du paquet, disponible sur son git-salsa. Pour le copier, nous utilisons la commande : *git clone <link>*. Pour débuter nous nous plaçons dans le répertoire de travail du paquet, *cd ufo-filters* . Lorsque l'on récupère un paquet il manque toujours le fichier *origtargz* . Il s'agit d'une archive contenant le code source original d'un logiciel avant les modifications que nous allons faire. Il est utilisé lors du processus d'empaquetage : il sert de base de référence pour distinguer le code de l'Upstream des changements ajoutés par Debian (patchs, autres etc) En pratique, l' *origtargz* garantit que l'on conserve toujours une copie intacte du code source d'origine, ce qui facilite les mises à jour, le suivi des différences et la conformité aux règles de packaging Debian. Afin de vérifier que le paquet fonctionne correctement avant toute implémentation, il faut le construire. Afin d'avoir une base propre j'utilise les commandes : *git clean -xdf ; git reset --hard ; dgit --gbp sbuild* rappelons que la commande dgit est très exigeante puisqu'elle ne fonctionne que si le répertoire local est propre (sans modifications non push ou fichiers indésirables).

Après quelques minutes le résultat de *sbuild* est donné, contenant les autopkgtest à implémenter.

```
Résultat :
        no-ocl                  PASS
        autopkgtest [11:14:18]: @@@@@@@@@@@@@@@@@@@@ summary
        reconstruct             PASS
        no-ocl                  PASS

        E: Autopkgtest run was successful
```

```
+------------------------------------------------------------------------------+
| Cleanup                                   Fri, 16 May 2025 09:14:18 +0000 |
+------------------------------------------------------------------------------+

Purging /build/reproducible-path
Not cleaning session: cloned chroot in use
```

Les tests sont marqués comme PASS, cela signifie qu'ils passent, et sont donc un succès. Dans un summary**,** le Lintian est un outil d'analyse pour les paquets Debian qui détecte automatiquement des erreurs, avertissements et mauvaises pratiques. Il sert à vérifier la qualité et la conformité des paquets par rapport aux standards Debian. Un Lintian Warn ou fail ne veut pas dire que le paquet ne s'est pas bien construit ou que les autopkgtest échouent. Cela signifie juste que le code n'est pas forcément de bonne qualité ou autre, mais que cela n'entrave pas son fonctionnement. Je débute l'implémentation des autopkgtest en allant dans le répertoire $*ufo-filters/debian/tests/* . Il y a le fichier *Control* ,avec différentes parties *Test-Command (indique la commande à lancer pour le test)* , *Depends (liste les paquets nécessaires au test)* ,

Architecture (précise sur quelles architectures le test peut tourner), Features (pour le nom de l'autopkgtest) , *Restrictions (définit des limites ou conditions spéciales d'exécution.)* et le fichier *OOpencl,*fichier de commande nécessaire à l'exécution des autopkgtest. Dans le fichier, *OpenCl,* je décommente les lignes d'exécution des tests .py provenant d'un dataset du CNRS.

```
3. Création fichier opencl

Dans control :

        Tests: opencl
        Depends:
         @,
         clinfo,
         pocl-opencl-icd,
         python3-all,
         python3-pytest,
         python3-pytest-mock,
         python3-pytest-xvfb,
         xauth,
         xvfb,
        Architecture: !i386
        Restrictions: allow-stderr,flaky, skip-not-installable
```

Fichiers sans Modifications

```
Opencl:
        #!/bin/sh -e

        # Check that OpenCL isn't totally broken (note that it isn't totally working either)
        # Uses device 0 platform 0, i.e. to use a real GPU manually install its opencl-icd before running this
        # Mark the test has flaky, the important part is the CPU computation.

        export PYOPENCL_COMPILER_OUTPUT=1

        pynx-test

        # cp bootstrap.py run_tests.py pyproject.toml version.py README.rst "$AUTOPKGTEST_TMP"

        # for py in $(py3versions -r 2>/dev/null)
        # do cd "$AUTOPKGTEST_TMP"
        #    echo "Testing with $py:"
        #    xvfb-run -a --server-args="-screen 0 1024x768x24" $py run_tests.py -v -m --installed
        # done
```

```
Résultat du test :

        autopkgtest [13:21:27]: test opencl: ----------------------]
        autopkgtest [13:21:28]: test opencl:  - - - - - - - - - - results -
        opencl               FLAKY non-zero exit status 132
        autopkgtest [13:21:29]: @@@@@@@@@@@@@@@@@@@@ summary
        command1             PASS
        opencl               FLAKY non-zero exit status 132
```

Il y a donc un problème, le test est marqué comme *Flaky* (instable). Je constate dans les logs d'erreurs que le *siphash24* n'est pas installé et qu'il suggère de le faire, je pense que c'est la source de l'erreur. Je l'installe via *sudo apt install pipx; pip install siphash24 –break-systeme-packages.* Résultat identique. L'erreur est donc ailleurs. *Command1* PASS mais pas *Opencl.* J'exécute seul les Test-Command avec *bash -n control; bash -n no-opencl; bash -n reconstruct*. Je rencontre l'erreur 128 qui indique que le TLS de Curl fonctionne mais pas forcément celui de GIT est différent de Curl , Curl utilise OpenSSL (ou GenTLS, selon la bulld), mais il gère beaucoup mieux certains comportements ambigus de proxy. GIT, sur Debian/Ubuntu.Utilise souvent libCurl compilé avec GenTLS. Ce backend est plus strict et échoue si le proxy ne respecte pas parfaitement la norme TLS. J'ai été bloqué pendant plusieurs jours, voire plusieurs semaines à ne pas savoir pourquoi mes tests échouaient. Je savais que le proxy et le protocole réseau de SOLEIL étaient le problème. J'étais d'une part bloqué par l'appréhension du système Debian lui-même que j'ai mis un peu de temps à appréhender et d'autre part par la complexité du problème lui-même. Le signal réseau parvenait jusqu'au site internet contenant le DataSet, il parvenait à passer le proxy à l'aller. Ce qui est validé par la validation du protocole TLS. Cependant, en pratique, le signal était bloqué à son entrée dans le réseau SOLEIL, c'est-à-dire lorsqu'il revenait. La question était la suivante : comment permettre son passage à son retour ? J'ai tout simplement essayé de désactiver le proxy en passant par la désactivation de l'IPV6 :

```
sudo sysctl -w net.ipv6.conf.all.disable_ipv6=1
sudo sysctl -w net.ipv6.conf.default.disable_ipv6=1
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
sudo apt-get -o Acquire::ForceIPv4=true update
```

```
sudo apt-get -o Acquire::ForceIPv4=true upgrade
Acquire::ForceIPv4 "true";
ping -4 deb.debian.org
echo 'Acquire::ForceIPv4 "true";' | sudo tee
/etc/apt/apt.conf.d/99force-ipv4
sudo apt-get update
```

Cette tentative s'est révélée sans succès. J'ai également vidé les caches. Le problème restait identique. Alors j'ai demandé à Fred et Emmanuel, comment intégrer le protocole réseau dans le fichier de config.pl de sbuild afin que les autopkgtest puissent utiliser le proxy lorsque cela est nécessaire. J'avais créé un fichier config.pl qui prenait en charge le chroot, etc. Cependant, il ne prenait pas en compte correctement le proxy, les protocoles HTTP et HTTPS. Fred m'a ensuite montré comment rendre mon fichier de config.pl propre et complet. En effet, je fonctionne en autonomie et mes maitres de stage me font confiance même s'il arrive parfois que je rencontre des difficultés techniques. Je tente de les surmonter quoiqu'il arrive, et sinon je vais poser des questions. Car c'est en persévérant que l'on apprend le plus. Alors, je n'ai plus qu'à remettre en forme le fichier config.pl afin qu'il prenne correctement en compte ces protocoles (cf fichier config.pl P3 1.2). Une fois ce problème réglé est apparue une autre problématique, celle de permettre un

accès complet au GPU via un container compatible avec ROCm. Après plusieurs jours de recherche et l'aide précieuse de Fred. J'ai pu mettre en place le container podman+rocm, car celui-ci prend en charge et configure automatiquement dev/dri et dev/kfd qui nous posaient tant de problèmes et l'ajouter au fichier config.pl. Cependant cela intervient bien plus tard. Voici les modifications apportées aux fichiers *Control*

Fichiers avec Modifications

```
CONTROL :

        Tests: opencl
        Depends: ufo-filters, libufo-bin, git, pocl-opencl-icd,
         ca-certificates, clinfo,python3-pyopencl,python-pyopencl-doc, python3-pytest,
         python3-pytest-mock, python3-pytest-xvfb,xauth,xvfb,curl, unzip, ufo-core
        Architecture: amd64, arm64, armel, armhf, i386
        Restrictions: allow-stderr, needs-internet, skip-not-installable,needs-root

        Tests: no-ocl
        Depends: ufo-filters, libufo-bin
        Restrictions: allow-stderr, skip-not-installable

        Test-Command: rocm-test-launcher  reconstruct
        Depends: ufo-filters, libufo-bin, git, pkg-rocm-tools, rocm-opencl-icd, ca-certificates
        Features: test-name=reconstruct-rocm
        Architecture: amd64, arm64, armel, armhf, i386
        Restrictions: allow-stderr, needs-internet, skip-not-installable,skippable
```

```
autopkgtest [16:11:46]: test opencl: [-----------------------
Cloning into 'ufo-testing'...
===== clinfo output =====
clinfo: aucun périphérique OpenCL détecté – test ignoré
autopkgtest [16:11:48]: test opencl: -----------------------]
autopkgtest [16:11:49]: test opencl:  - - - - - - - - - - results - - -
opencl               FAIL non-zero exit status 77
```

Sur l'hôte, clinfo voit bien le GPU AMD et OpenCL fonctionne. Mais dans le schroot utilisé par autopkgtest, le test OpenCL échoue car aucun périphérique OpenCL n'est détecté, c'est à dire que le chroot ne voit pas les cartes. C'est un problème classique d'isolation de chroot : le schroot ne voit pas le GPU (et/ou les bibliothèques OpenCL), même si l'hôte fonctionne parfaitement.

Après l'ajout de podman+rocm au fichier de config.pl et sa mise en place selon la procédure il reste encore des erreurs. Notons que la config du container est faite pour l'ensemble des paquets. Nous avons ajouté la dépendance *libclang-common-17-dev* et réécrit les commandes des tests avec leurs chemin *ex :debian/tests/opencl* . Après cela les tests sont tous marqué comme PASS. Fred à pu push les modifications sur le git salsa de Ufo-filters.

Nabu:

J'ai procédé de la même manière pour le lancement de Nabu (sans aucune modification). Cependant, il se trouve que la dépendance *pocl-opencl-icd* n'est pas reconnue.On vérifie la présence de la dépendance à l'aide la commande suivante : *apt cache show pocl-opencl-icd* dont voici le résulat , on peut notamment y voir la version.

```
Résultat :

        Package: pocl-opencl-icd
        Source: pocl
        Version: 6.0-6
        Installed-Size: 47
        Maintainer: Debian OpenCL Maintainers <pkg-opencl-devel@lists.alioth.debian.org>
        Architecture: amd64
        Provides: opencl-icd, opencl-icd-1.1-1, opencl-icd-1.2-1
        Depends: libpocl2t64 (= 6.0-6)
        Description-en: PoCL ICD
        Portable Computing Language is an open source implementation of the OpenCL
        standard which can be easily adapted for new targets. One of the goals of the
        project is improving performance portability of OpenCL programs, avoiding the
        need for target-dependent manual optimizations. A "native" target is included,
        which allows running OpenCL kernels on the host (CPU).
        .
        This package allows one to use PoCL as an installable client driver (ICD) for
        OpenCL.
        Description-md5: a2a9ed5c155c582d02308f5380e633cd
        Multi-Arch: same
        Homepage: http://portablecl.org
        Tag: role::shared-lib
        Section: libs
        Priority: optional
        Filename: pool/main/p/pocl/pocl-opencl-icd_6.0-6_amd64.deb
        Size: 27100
        MD5sum: 277a0ea37bf0be5d1b66886917be70fb
        SHA256: 75a364f99809f7134fe0fd1dde4cb8c9d690c89bae348abe53f8fec0e2c89297
```

Enfin, je mets à jour le chroot afin qu'il reconaisse *pocl-open-icd* . Pour cela on utilise la commande *sudo sbuild-update -ud unstable-amd64-sbuild* et lance sbuild à nouveau pour vérification. Il se trouve que cela a eu l'effet escompté.

Puisque, l'on a toujours, un log d'erreur le chroot n'était peut-être pas bien reconnu. J'utilise alors la commande *ls /srv/chroot* . Et le fait également via la commande *schroot -l*. En réalité du fait de ma prise en charge du système Debian, j'avais encore quelque difficulté et me heurtais encore à des soucis de compréhension fort heureusement mes maîtres de stages étaient là pour m'expliquer certaines notions qui étaient encore floues pour moi, notamment les patchs et les bugs reports.

```
longeac@grades-01:~/nabu$ sbuild
dpkg-source: info: using options from nabu/debian/source/options: --extend-diff-ignore=^[^/]+.(egg-info|dist-info)/
dpkg-buildpackage: info: source package nabu
dpkg-buildpackage: info: source version 2024.2.1-1
dpkg-buildpackage: info: source distribution unstable
dpkg-buildpackage: info: source changed by Roland Mas <lolando@debian.org>
dpkg-checkbuilddeps: error: unmet build dependencies: pocl-opencl-icd
dpkg-buildpackage: error: build dependencies/conflicts unsatisfied; aborting
dpkg-buildpackage: hint: satisfy build dependencies with your package manager frontend
E: Failed to clean source directory /home/experiences/grades/longeac/nabu (/home/experiences/grades/longeac/nabu_2024.2.1-1.dsc)
I: use sbuild --no-clean-source to skip the cleanup
```

Lancement après : *sudo sbuild-update -ud unstable-amd64-sbuild*

```
Résultat :

        longeac@grades-01:~/nabu$ schroot -l
        chroot:UNRELEASED
        chroot:UNRELEASED-amd64-sbuild
        chroot:UNRELEASED-amd64-sbuild-source
        chroot:UNRELEASED-source
        chroot:bookworm-amd64-sbuild
        chroot:bookworm-backports-amd64-sbuild
        chroot:experimental-amd64-sbuild
        chroot:haskell
        chroot:sid
        chroot:sid-source
        chroot:testing-amd64-sbuild
        chroot:unstable-amd64-sbuild
        source:UNRELEASED
        source:UNRELEASED-amd64-sbuild
        source:bookworm-amd64-sbuild
        source:bookworm-backports-amd64-sbuild
        source:experimental-amd64-sbuild
        source:haskell
        source:sid
        source:testing-amd64-sbuild
        source:unstable-amd64-sbuild
```

Il est important de penser à faire un commit lorsque l'on utilise les commandes *git clean -xdf ; git reset --hard* car cela supprime tous les fichiers non commit. **Attention les commandes ne sont pas réversibles, *git clean -xdf ; git reset --hard* donne un résultat différent que si l'on fait *git reset --hard ; git clean -xdf* .** Cette nuance pourtant pas du tout évidente m'a fait perdre près d'une semaine de travail car je n'avais pas saisi toute l'importance de la réversibilité des commandes. En effet, dans cet exemple précis, c'est assez contre intuitif. Pour vérifier la différence en l'upstream et notre implémentation il est possible de faire la commande *git diff*. Je vais aborder cette fois ci l'implémentation qui a été effectuée. La commande que j'utilise pour lancer les tests n'est plus *dgit --gbp sbuild* car celle-ci impose un environnement propre. Je n'étais jusque là pas encore autorisé à faire des commit. Sous réserve de ces contraintes, j'utilise désormais *sbuild --no-clean-source* qui est beaucoup plus flexible et autorise de lancer des constructions même si le paquet est en cours d'implémentation. Les tests sont marqués comme PASS sans implémentation. De ce fait, je passe dans le directory *nabu/debian/tests/* puis *nano control* . J'ai tout d'abord implémenté deux fichiers un fichier *nabu-opencl* déjà présent et un fichier *nabu-rocm*. Ce n'était pas la bonne façon de procéder. Afin de fournir un travail propre Frédéric-Emmanuel PICCA m'a indiqué qu'il valait mieux regrouper cela dans un seul fichier d'autopkgtest que l'on nommera *nabu-pocl* et ensuite d'ajouter des *features* dans le fichier afin de distinguer les autopkgtest ROCm et POCL. Les dépendances *HIPCC* et *libclang-common-17-dev* ont été ajoutées pour les faire fonctionner.

Fichier *control* implémenté

```
Test-Command: debian/tests/nabu-pocl
Depends:
 clinfo,
 pocl-opencl-icd,
 python3-all,
 python3-nabu,
Features: test-name=nabu-pocl
Architecture: amd64 arm64
Restrictions: allow-stderr


Test-Command: rocm-test-launcher debian/tests/nabu-pocl
Depends:
 clinfo,
 rocm-opencl-icd,
 pkg-rocm-tools,
 hipcc,
 python3-all,
 python3-nabu,
 libclang-common-17-dev,
Features: test-name=nabu-rocm
Architecture: amd64 arm64
Restrictions: allow-stderr,skippable
```

Fichier *Nabu-ROCm* implémenté

```
#!/bin/bash

set -efu

# Check that OpenCL isn't totally broken (note that it isn't totally working either)
# Uses device 0 platform 0, i.e. to use a real GPU manually install its opencl-icd before running this

# Mark the test has flaky, the important part is the CPU computation.

# display the OpenCL platform info

export HOME="$AUTOPKGTEST_TMP"

clinfo

# display opencl devices seens via nabu.

# some tests are expected to fail
# so list tests to skip in array variable SKIP_TEST_LIST

declare -a SKIP_TEST_LIST

SKIP_TEST_LIST=(${SKIP_TEST_LIST[@]} \
                test_2D_lossless \
                test_2D_lossy \
                test_3D \
                test_halftomo_right_side[config1] \
                test opencl unsharp)
SKIP_TESTS=""
list_initialised=0
for t in ${SKIP_TEST_LIST[@]}; do
    if [ ${list_initialised} = 0 ]; then
        SKIP_TESTS=$t
        list_initialised=1
    else
        SKIP_TESTS="${SKIP_TESTS} or $t"
    fi
done

if [ "x${SKIP_TESTS}" != "x" ]; then
    SKIP_TESTS="not ( ${SKIP_TESTS} )"
fi

echo "skipping tests with SKIP_TEST_LIST=${SKIP_TEST_LIST[@]}"

for py in $(py3versions -r 2>/dev/null)
do cd "$AUTOPKGTEST_TMP"
   echo "Testing with $py:"
   $py -c "import sys; import nabu.opencl.utils; print(nabu.opencl.utils.get_opencl_devices('all')); sys.exit(0)" 2>&1
   echo "$py -m pytest --pyargs nabu -v -k ${SKIP_TESTS}"
   $py -m pytest --pyargs nabu -v -k "${SKIP_TESTS}"
done
```

Même si le Lintian reste fail cela n'entache pas le fait que tous les tests sont marqués comme PASS, sans *Podman+ROCm*. Il ne reste donc qu'à le PUSH. Je commit. *Git add . ; git commit* . Lorsque je commit j'utilise toujours ce titre que Fred m'a conseillé afin d'être propre et clair : *d/t/control : Added ROCm and POCL autopkgtest* . Le résultat est identique avec *Podman+ROCm*. Cependant en tentant de vérifier si tous les tests étaient bien exécuté je me suis aperçu que tous les tests étaient en réalité SKIPPED. J'ai alors pensé que c'était en raison du manquement d'un dataset. J'ai envoyé un mail au mainteneur afin qu'il m'en fournissen

un. En me plongeant dans le code je me suis aperçu que le développement de la partie en OpenCL était encore en phase de test et que par conséquent, à chaque fois ils étaient SKIPPED. Ce qui est très étonnant car OpenCL est disponible sur nos machines. Les tests devraient réussir. Il y donc un problème de reconnaissance de la disponibilité d'OpenCL via *pyopencl*. On a donc des tests qui sont skipped volontairement. En fait, c'est assez opaque car les tests sont skip silencieusement, certaines parties sont en « prototype ». Exemple, toute la classe du fichier *test_fftshift.py* faisant intervenir *pyopencl* est skip car elle est en prototype. Après avoir fait des recherches sur le sujet, il est possible que même si l'on a un CPU et un GPU disponible, *pytest* préfèrera toujours le CPU. Pour régler ce problème nous devrions mettre une condition entre les deux afin que pytest n'ait plus le choix. En somme il semble y avoir pas mal de soucis avec ça. Je vais lancer les tests en commentant les *pystest.marked.skip* pour voir si cela échoue. De plus le passage mathématique des réels aux complexes ne semble pas supporté. En outre cela un serait un problème connu entre *vkfft* en *Pocl*. L'objectif est désormais de faire un patch afin que l'on puisse constater l'échec ou le passage des tests. J'ai eu énormément de mal à réaliser des patchs car je n'avais pas la procédure exacte mais après que Fred et Emmanuel m'aient bien expliqué j'ai pu mettre cela en production. Cependant j'avais pas mal de problème lors du rebasage et des merges de branches. Car j'avais le plus souvent des problèmes de fusion en raison des différences entre les branches, se qui rendait la mise en pratique fastidieuse. Après de nombreux patchs fait au corps du paquets. Je lance le build et les tests ne s'affiche plus dans le log de construction même en mode verbeux. Cette fois-ci je n'ai plus d'indication sur leur passage ou non. Chaque test où la ligne de skip a été commenté n'apparait plus dans les logs. Pour résoudre ces problèmes de fusion j'utilise l'outil Meld. Les patchs m'ont fait perdre énormément de temps en raison de ces problèmes. C'est une interface graphique permettant de gérer en problème.de Merge. Pour le passage entre *la patch-queue* et la branche *master* on utilise la commande line : *git checkout <nom de la branche>* .J'ai ajouté ceci dans le fichier *nabu/processing/tests/test_padding.py* pour vérifier la détection de Opencl, sans succès. Le problème ne bouge pas car cela n'a pas fait ressortir de logs. J'indique

```
Pour le script :  nabu/processing/tests/test_fft.py , à la ligne 164  on a :

# FIXME - vkfft + POCL fail for R2C in one dimension
if config["backend"] == "opencl" and r2c and ndim == 1:
    if self.cl_ctx.devices[0].platform.name.strip().lower() == "portable computing language":
        pytest.skip("Something wrong with vkfft + pocl for R2C 1D")

   @pytest.mark.skipif(
        not (__has_skcuda__ and __has_pycuda__), reason="Need pycuda and (scikit-cuda or vkfft) for this test"
    )

Ou encore le passage réel -> complexe non supporté...

Dans  test_fftshift.py :

On a :
@pytest.mark.skip(reason="OpenCL fftshift is a prototype")
Donc skipped , car il est mis avant les checks pour ROCm et POCL .

On a aussi :
@pytest.mark.skipif(not (__has_pyopencl__), reason="Need pyopencl for this test")

Dans test_transpose.py
 Des @pytest.mark.skip
```

Identification de la cause dans les scripts de tests

```
n/p/t/test_fft.py Comment @pytest.skip

n/p/t/test_medfilt.py Comment @pytest.skip
```

Titre des patchs

```
import pyopencl as cl

try:
    # Liste des plateformes
    platforms = cl.get_platforms()

    if not platforms:
        print("Aucune plateforme OpenCL trouvée !")
        exit(1)

    print(f"PyOpenCL {cl.VERSION_TEXT} - {len(platforms)} plateforme(s) trouvée(s):")
for i, platform in enumerate(platforms):
    print(f"\nPlateforme {i+1}:")
    print(f"  Nom: {platform.name}")
    print(f"  Vendor: {platform.vendor}")
    print(f"  Version: {platform.version}")

# Liste des devices
devices = platform.get_devices()
print(f"  {len(devices)} device(s):")

for j, device in enumerate(devices):
    print(f"    Device {j+1}: {device.name}")
    print(f"      Type: {cl.device_type.to_string(device.type)}")
    print(f"      Mémoire globale: {device.global_mem_size//1024**2} MB")
```

Partie ajoutée au fichier de test *nabu/processing/tests/test_padding.py* pour la tentative de détection de la plateforme Opencl

alors l'ensemble de mes problèmes à Frédéric-Emmanuel PICCA. En conclusion, les tests sont tous implémentés et cela ne pose pas de problème pour le PUSH puisqu'ils sont marqués comme PASS. Mais cela n'est pas satisfaisant car les tests Opencl ne réussissent pas et je ne parviens pas à résoudre le problème efficacement même avec l'aide de mes maitres de stages.

Pynx:

La procédure de lancement sans aucune modification est identique aux paquets précédents. L'autopkgtest *command1* est marqué comme PASS mais l'autopkgtest faisant intervenir la partie OpenCL est marqué comme SKIP. Il n'y a pas de partie ROCm. Je décommente les lignes du fichier *OpenCL* afin de garantir le bon fonctionnement des autopkgtest. J'implémente les autopkgtest, notamment dans le fichier *Control*. J'installe siphash 24. Parfois Xvfb plante et il est nécessaire de le fermer à la main : *ps aux | grep Xvfb ; kill <n° process>* . Voici le fichier *control* implémenté.

Fichier *control* implémenté pour POCL et ROCm et l'architecture amd64

```
Test-Command: rocm-test-launcher debian/tests/opencl
Depends:
 @,
 clinfo,
 rocm-opencl-icd,
 pkg-rocm-tools,
 python3-all,
 python3-pytest,
 python3-pytest-mock,
 python3-pytest-xvfb,
 python-pycuda-doc,
 xauth,
 xvfb,
 hipcc,
 rocm-device-libs-17,
 libclang-common-17-dev,
Architecture: i386 amd64 arm64
Features: test-name=opencl-rocm
Restrictions: allow-stderr,flaky, skip-not-installable, skippable

Test-Command: debian/tests/opencl
Depends:
 @,
 clinfo,
 pocl-opencl-icd,
 python3-all,
 python3-pytest,
 python3-pytest-mock,
 python3-pytest-xvfb,
 python-pycuda-doc,
 xauth,
 hipcc,
 xvfb,
Features: test-name=opencl-pocl
Architecture:  i386 amd64 arm64
Restrictions: allow-stderr,flaky,  skip-not-installable, skippable

Test-Command: xvfb-run -s "-screen 0 1024x768x24 -ac +extension GLX +render -noreset" sh debian/tests/gui
Depends:
 debhelper,
 mesa-utils,
 pocl-opencl-icd,
 pynx,
 python3-all,
 @,
 xauth,
 xvfb,
 python-pycuda-doc,
Architecture:  i386 amd64 arm64
Restrictions: allow-stderr, flaky,  skip-not-installable, skippable
```

```
ROCM :
ERROR: test_save_nxtomo (pynx.holotomo.test.test_holotomo.TestHoloTomo.test_save_nxtomo)
ERROR: test_save_chunk_hdf5 (pynx.holotomo.test.test_holotomo.TestHoloTomo.test_save_chunk_hdf5)
ERROR:  test_pycuda_reduction (pynx.processing_unit.test.test_cuda.TestCUDA.test_pycuda_reduction)
ERROR: test_cuda_reverse_block (pynx.processing_unit.test.test_cuda.TestCUDA.test_cuda_reverse_block)
ERROR: test_cuda_processing_unit (pynx.processing_unit.test.test_cuda.TestCUDA.test_cuda_processing_unit)
ERROR: test_cuda_gpu_device (pynx.processing_unit.test.test_cuda.TestCUDA.test_cuda_gpu_device)
ERROR: test_cuda_elementwise (pynx.processing_unit.test.test_cuda.TestCUDA.test_cuda_elementwise)
```

Les autopkgtest sont tous implémentés et près a être testé. Cependant, je fais en sorte que cette fois ci les tests ne soient pas marqués comme SKIP mais plutôt que lorsqu'il y a une erreur dans un test, celui-échoue directement et stop le build. L'autopkgtest *command1* passe toujours. La partie POCL et ROCm ne rencontre, hélas, pas ce succès puisqu'il y a énormément d'erreur dans les tests. Deux choix s'offrent à moi. Soit, je les skip et implémentant le fichiers *rules* contenu dans le répertoire *debian/*, soit je corrige tous les tests à la main, ce qui risque de prendre énormément de temps et d'énergie. Je chois donc de les corriger à la main. Voici les fichiers de test contenant les erreurs que ce soit avec POCL ou ROCm. Il faut rendre cela plus verbeux afin d'avoir une vision précise.

Lorsque l'on touche au corps d'un paquet et non à sa partie Debian, nous sommes obligés de faire des patchs. Je me suis aperçu en lisant les logs beaucoup plus verbeux que la source du problème ne provenait pas du tout de la partie OpenCL en elle-même ni de la disponibilité de POCL ou de ROCm. Cela provenait de la partie CUDA qui, lorsque je passais sur GPU me donnait en sortie une erreur.

En résumé sur l'ensemble des tests qui échoue la quasi-totalité sont dues à CUDA, que soir via les dépendances *pycuda, MPI* ou encore selon qu'il n'y ait pas de support CUDA et MPI. Cela est le cas pour POCL et pour ROCm , j'ai d'ailleurs fait un bug report à ce sujet [Bugs report Pynx] .

```
ERROR occurred on: dbdbe71fd9d2 [Linux-6.12.32-amd64-x86_64-with-glibc2.41]
AssertionError: 1 is not false : Failed command-line: pynx-cdi-id01 --data /tmp/tmpbz181iir/TestCDIigua1upg.cxi
 --nb_raar 50 --nb_hio 50 --nb_er 50 --support_update_period 20 --support_post_expand=-1,2
PyNX processing unit: select OpenCL language from PYNX_PU environment variable
Loading data:  /tmp/tmpbz181iir/TestCDIigua1upg.cxi
  CXI input: Energy =     8.27keV
  CXI input: detector distance =     1.00m
  CXI input: detector pixel size =    55.00um
  CXI input: loading iobs
  CXI input: loading mask, with 41753 pixels masked ( 1.991%)
Finished loading iobs data, with size: 2097152
CDI runner: preparing processing unit
Searching available OpenCL GPU [ranking by global memory bandwidth]:
Ignoring Portable Computing Language (POCL) platform by default
ERROR occurred on: dbdbe71fd9d2 [Linux-6.12.32-amd64-x86_64-with-glibc2.41]

FAILED (failures=72, errors=55, skipped=54, expected failures=1)
autopkgtest [12:14:42]: test opencl-pocl: ----------------------]
autopkgtest [12:14:42]: test opencl-pocl:  - - - - - - - - - - results - - - - - - - - - - -
opencl-pocl          FLAKY non-zero exit status 1

** ROCm :

ERROR: test_cuda_processing_unit (pynx.processing_unit.test.test_cuda.TestCUDA.test_cuda_processing_unit)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/usr/lib/python3/dist-packages/pynx/processing_unit/test/test_cuda.py", line 59, in test_cuda_processing_unit
    from pynx.processing_unit import cu_processing_unit
  File "/usr/lib/python3/dist-packages/pynx/processing_unit/cu_processing_unit.py", line 13, in <module>
    from .cuda_device import cu_drv, cua, has_cuda, has_vkfft_cuda, has_skcuda, VkFFTApp, cu_fft
ImportError: cannot import name 'cua' from 'pynx.processing_unit.cuda_device' (/usr/lib/python3/dist-packages/pynx/processing_unit/cuda_device.py)

 File "/usr/lib/python3/dist-packages/h5py/_debian_h5py_serial/_hl/filters.py", line 297, in fill_dcpl
    raise ValueError("Unknown compression filter number: %s" % compression)
ValueError: Unknown compression filter number: 32004

AttributeError: 'NoneType' object has no attribute 'Device'

AssertionError: False is not true : /usr/lib/python3/dist-packages/pytools/persistent_dict.py:52: RecommendedHashNotFoundWarning:
Unable to import recommended hash 'siphash24.siphash13', falling back to 'hashlib.sha256'. Run 'python3 -m pip install siphash24' to install the recommended hash.
  warn("Unable to import recommended hash 'siphash24.siphash13', "
/usr/lib/python3/dist-packages/pyopencl/cache.py:496: CompilerWarning: Built kernel retrieved from cache. Original from-source build had warnings:
Build on <pyopencl.Device 'gfx1034' on 'AMD Accelerated Parallel Processing' at 0x72b9ed0> succeeded, but said:

warning: argument unused during compilation: '-I /usr/lib/python3/dist-packages/pyopencl/cl' [-Wunused-command-line-argument]
1 warning generated.

  _create_built_program_from_source_cached(
```

Logs d'erreur des tests pour POCL et ROCm sur CUDA plus verbeux

Afin d'éviter ces logs d'erreurs je souhaite les skips car cela fait intervenir CUDA, ce qui est dans mon cas pas intéressant. J'implémente donc une solution dans le fichier */debian/rules* afin de skip ces tests. En réalité cela provient du fait que les tests cherchent des GPU NVIDIA pour s'exécuter mais cela ne fonctionne pas pour une raison que j'ignore car nous en possédons et l'accès est à priori bien configuré car je n'ai pas eu de problème sur les autres paquets à ce sujet. Dans le but de résoudre les problèmes j'ai tenté d'installer les paquets python3-pycuda et de python-pycuda-doc, nvidia-cuda-toolkit-doc pour résoudre les problèmes dans les tests faisant intervenir pycuda. J'ai retiré nvidia-

```
export PYBUILD_TEST_ARGS=\
-k "not test_that_multiple_compiles_do_not_occur_for_same_source \
and not test_const_as_call_arg \
and not test_const_in_return" \
--ignore=$(CURDIR)/pynx/processing_unit/test/test_cuda.py \
--ignore=$(CURDIR)/pynx/holotomo/test/test_holotomo.py \
--ignore=$(CURDIR)/pynx/ptycho/runner/test/test_runner.py \
--ignore=$(CURDIR)/pynx/scattering/test/test_scattering.py \
--ignore=$(CURDIR)/pynx/cdi/runner/test/test_runner.py \
--ignore=$(CURDIR)/pynx/cdi/test/test_cdi.py \
--ignore=$(CURDIR)/pynx/ptycho/runner/tps25a.py \

#$(CURDIR)/pynx/tests
%:
        dh $@ --buildsystem=pybuild


Modification de l'ajout du code en :

override_dh_auto_test:
        PYBUILD_TEST_ARGS="\
                --ignore=$(CURDIR)/pynx/processing_unit/test/test_cuda.py \
                --ignore=$(CURDIR)/pynx/holotomo/test/test_holotomo.py \
                --ignore=$(CURDIR)/pynx/ptycho/runner/test/test_runner.py \
                --ignore=$(CURDIR)/pynx/scattering/test/test_scattering.py \
                --ignore=$(CURDIR)/pynx/cdi/runner/test/test_runner.py \
                --ignore=$(CURDIR)/pynx/cdi/test/test_cdi.py \
                --ignore=$(CURDIR)/pynx/ptycho/runner/tps25a.py \
                -k 'not test_that_multiple_compiles_do_not_occur_for_same_source and not test_const_as_call_arg and not test_const_in_return'" \
        dh_auto_test
```

Implémentation des tests à skip dans le fichier *debian/rules*

```
        "Failed initialising GPU. Please check GPU name or CUDA/OpenCL installation" + s0)
pynx.cdi.runner.runner.CDIRunnerException: Failed initialising GPU. Please check GPU name or CUDA/OpenCL installation
  original error: Could not find a suitable GPU. Please check GPU name or CUDA/OpenCL installation
Caught exception for scan None:

   Failed initialising GPU. Please check GPU name or CUDA/OpenCL installation
  original error: Could not find a suitable GPU. Please check GPU name or CUDA/OpenCL installation
on dbdbe71fd9d2 [Linux-6.12.32-amd64-x86_64-with-glibc2.41]

Exception: Could not find a suitable GPU. Please check GPU name or CUDA/OpenCL installation

AssertionError: 1 is not false : Failed command-line: pynx-cdi-simulation --size 80

--size 80 --obj_radius 15 --nb_raar 50 --nb_hio 50 --nb_er 50 --obj_shape dodecahedron
PyNX processing unit: select OpenCL language from PYNX_PU environment variable
CDI runner: preparing processing unit
Searching available OpenCL GPU [ranking by global memory bandwidth]:
Ignoring Portable Computing Language (POCL) platform by default
```

Logs d'erreur sur la recherche de GPU NVIDIA

```
  File "/usr/lib/python3/dist-packages/pynx/ptycho/runner/tps25a.py", line 151, in check_params_beamline
    if self.params['scanfile'] is None:
       ~~~~~~~~~~~~~~^^^^^^^^^^^^^
KeyError: 'scanfile'
  -testing /usr/bin/pynx-tps25apty.py: exit(1) FAILED !!!
```

Exemple d'erreurs de code rencontrées

cuda-toolkit-doc et python3-pycuda car ils n'étaient pas installables lors du build. J'ai également résolu des problèmes de code. Notamment sur des variables qui n'était pas alloué et des petites erreurs de code, ou encore des variables inconnues d'un dictionnaire.

Clblast :

La procédure de lancement sans aucune modification est identique aux paquets précédents. L'ensemble des tests sont marqués comme FLAKY ou FAIL que ce soit pour ROCm ou POCL. Cependant après l'ajouts de la dépendance *libc6-dev* le test POCL à pu passer. Mais ce n'est pas le cas de ROCm qui a rencontré beaucoup de problème sur certaines routines et qui m'ont pris plusieurs jours à analyser. J'ai implémenté le fichier *control* pour l'autopkgtest ROCm. J'ai également ajouté duc ode dan le fichier *find_pocl.c* et *test-pocl.sh*. J'ai implémenté les tests PoCl et ROCm dans le fichier *control*. Après avoir lancé le sbuild, il s'avère que les tests faisant intervenir le CPU, soit POCL passent. Or ceux pour la partie GPU rencontre des problèmes de précision notamment liés à la nature des tests. La raison pour laquelle ces tests ne passeraient pas, serait que la carte ne prend pas en charge ou prend mal en charge les calculs en FP16 sur ROCm. Ces bibliothèques prennent en charge ces calculs mais cela ne suffirait peut-être pas à les faire passer au regard de la précision. Il faudrait donc augmenter la tolérance dans le but de voir si cela serait suffisant (*rocBLAS, MIOpen et ROCm-Lib*). Cela peut provenir du fait que la carte a une limitation au niveau du noyau. Apparemment ce sont des problèmes courants avec FP16 sur ROCm, et intrinsèques à la carte que l'on possède. C'est symptomatique si l'on n'a pas des cartes professionnelles. Problèmes courants avec FP16 sur ROCm sur les problèmes de précision. Les tests CLBLAST échouent parfois à cause de tolérances trop strictes pour FP16. ROCm peut utiliser des conversions implicites FP16 en FP32 (pour compatibilité), ce qui peut expliquer les différences numériques. Un support matériel incomplet peut être la cause. Certaines cartes (comme la RX 6400) supportent FP16, mais avec des extensions limitées (ex: pas de FP16/WMMA sur RDNA 2 grand public). La carte que l'on possède aurait un support FP16 partiel. Ce qui expliquerait notre problème et d'autres problèmes que l'on pourrait avoir à l'avenir.

```
Test-Command: debian/tests/tests-pocl.sh
Depends: clinfo, clblast-tests, colorized-logs, gcc, ocl-icd-opencl-dev | opencl-dev, pocl-opencl-icd , libc6-dev
Features: test-name=tests-pocl
Architecture: amd64 arm64 i386
Restrictions: allow-stderr

Test-Command: rocm-test-launcher debian/tests/tests-pocl.sh
Features: test-name=tests-rocm
Depends:
 clblast-tests,
 colorized-logs,
 gcc,
 ocl-icd-opencl-dev | opencl-dev,
 rocm-opencl-icd,
 pkg-rocm-tools,
 libc6-dev,
 rocm-device-libs-17,
 hipcc,
 libclang-common-17-dev,
 clinfo,
 mesa-utils,
 libclblast-dev,
Architecture: amd64 arm64 i386
Restrictions: allow-stderr, skippable
```

Fichier *control* après implémentation

Source :

https://www.amd.com/content/dam/amd/en/documents/radeon-tech-docs/instruction-set-architectures/rdna2-shader-instruction-set-architecture.pdf
https://cnugteren.github.io/clblast/clblast.html
https://ar5iv.labs.arxiv.org/html/1795.05249 cf (section 3.1.Library Design et 3.2)
https://localai.io/basics/build/ (dépendances nécessaires rocblas, miopen-hip)
https://www.amd.com/en/developer.html

Nous avons une version ancienne (ROCm 3.5.9) avec des bugs connus sur FP16 après quelques recherches. Les cartes grand public comme la RX 6400 ne sont pas prioritaires pour les optimisations FP16 dans ROCm. Les erreurs sont dans les routines suivantes.

HGEMMBATCHED, MAXPVBATCHED, IHAMAX

- *HGEMMBATCHED* : Sur RX 6400, utilise des conversions implicites FP16 en FP32
- *MAXPVBATCHED* : Limité par la bande passante mémoire, Tolérances FP16 souvent relâchées (cf. vos tests)
- *IHAMAX :* Sensible aux erreurs d'arrondi FP16 Vos tests montrent 10/12 échecs + nécessité d'ajuster les tolérances

Les tests FP16 échouent car les tolérances par défaut sont trop strictes pour le FP16 sur GPU grand public. Notre RX 6400 utiliserait des conversions FP16 en FP32 silencieuses, d'où les erreurs.

J'ai fait un bugs report à l'upstream. clblast: Bug precisions in autopkgtest testing compatibility with AMD ROCm by using command sbuild . Voici la conversation : https://github.com/CNugteren/CLBlast/issues/588#issuecomment-3185095556

```
Exemple pris pour le paquet "compyle":
Pour des fichiers python:

export PYBUILD_TEST_ARGS=\
-k "not test_that_multiple_compiles_do_not_occur_for_same_source \
and not test_const_as_call_arg \
and not test_const_in_return" \
--ignore=$(CURDIR)/compyle/tests/test_cuda.py \
--ignore=$(CURDIR)/compyle/tests/test_parallel.py \
--ignore=$(CURDIR)/compyle/tests/test_opencl.py \
$(CURDIR)/compyle/tests

%:
	dh $@ --buildsystem=pybuild
```

J'ai tenté de skip les tests mais cela reste des tests importants donc je ne peux pas les skips comme ça. Voici un exemple de code tiré du code pour le paquet *Compyle* dans le à mettre dans le fichier *debian/rules* pour skip les tests en .py

En conclusion, tous les autopkgtest ont été implémenté, POCL passe mais pas encore ROCm pour les problèmes évoqués. En attente que le mainteneur Debian fasse le nécessaire pour résoudre ce problème majeur sur le support AMD ROCm.

Clpeak :

La procédure de lancement sans aucune modification est identique aux paquets précédents. Les tests passent lorsque l'implémentation des autopkgtest pour ROCm n'est pas encore faite. J'implémente alors les tests ROCm dans le fichier *control*.

```
Test-Command: debian/tests/upstream-binaries-pocl clpeak
Depends: clpeak, pocl-opencl-icd, clinfo,ocl-icd-opencl-dev
Features: test-name=clpeak-pocl
Restrictions: skippable, allow-stderr , skip-not-installable
Architecture: amd64 arm64 armel armhf i386

Test-Command: rocm-test-launcher clpeak
Depends: clpeak, rocminfo, clinfo, rocm-opencl-icd, pkg-rocm-tools,libclang-common-17-dev
Features: test-name=clpeak-rocm
Restrictions: skippable, allow-stderr, skip-not-installable
Architecture: amd64 arm64 ppc64el
```

Fichier *Control* après implémentation

J'ai ajouté le fichier *upstream-binaries-pocl* afin de gérer l'aspect ROCm. Je me suis inspiré du fichier d'origine *upstream-binaries* .

En effet, nous pouvons vérifier l'accès au CPU AMD ou au GPU AMD ROCm lors du build.

```
Preuves que POCL fonctionne :
Platform & Device
Platform: Portable Computing Language → POCL est bien utilisé.
Device: cpu-haswell-AMD Ryzen Threadripper 3990X → Le CPU est reconnu comme un device OpenCL via POCL.
Driver version
Driver version: 6.0+debian (Linux x64) → Version du pilote POCL (paquet Debian).
Benchmarks réussis
Toutes les métriques sont calculées sans erreur (sauf half precision, non supportée).
Les performances (GFLOPS, GBPS, GIOPS) sont cohérentes pour un CPU.
Aucune erreur critique
Les 64 warnings generated sont normaux (POCL émet des warnings sur certaines optimisations).

Détails des résultats
Test    Résultat        Analyse
Global Memory Bandwidth ~50 GB/s        Normal pour un CPU (mémoire DDR4).
Single-Precision (FP32) 1771 GFLOPS     Performances typiques d'un Ryzen 3990X.
Double-Precision (FP64) 1626 GFLOPS     Ratio FP64/FP32 ~0.9x (attendu pour Zen2).
Transfer Bandwidth      13-14 GB/s (PCIe-like)  POCL simule un "device" avec une fausse mémoire dédiée.
Kernel Latency  133 µs  Un peu élevé (normal en CPU vs GPU).
```

rocm-test-launcher est un wrapper qui configure l'environnement ROCm avant de lancer cipeak. Il ne s'affiche pas dans la sortie, on ne verra pas *rocm-test-launcher* dans *clpeak*. Mais son rôle est de vérifier que ROCm est disponible et de désactiver d'autres backends OpenCL (comme POCL) pour forcer l'utilisation du GPU. L'ajout de *libclang-common-17-dev* a permis de résoudre beaucoup de problème. La dépendance *libclang-common-17-dev* est essentiel pour l'autopkgtest de clpeak car il fournit les outils de compilation LLVM/Clang requis par POCL (Portable Computing Language), l'implémentation OpenCL pour CPU. Sans lui, POCL ne peut pas compiler les kernels OpenCL lors des tests, entraînant des échecs silencieux ou des erreurs de build. Cette dépendance assure le bon fonctionnement de l'environnement de test et la validation des performances OpenCL. Lors du sbuild il se peut que le dossier des fichiers temporaire soit plein */tmp* , il faut donc penser à le vider régulièrement.

Helo,

I am in an internship at Synchrotron-Soleil , my directors are Frederic-Emmanuel PICCA and Emmanuel FARHI. I implemented Rocm and Pocl autopkgtest for architecture amd64 and arm64. And I would like to push it.
I implemented autopktests for rocm and pocl using Opencl on the package named Clpeak on local. The aim is to have an overview of code compatibility with various AMD graphics cards, the codes on all the AMD boards available for CI rocm for GPU and Pocl for CPU.

Would you allow me to push it?

We look forward to hearing from you !

Warmly,

Clément LONGEAC

Voici le Canvas de mail utilisé et en adaptant selon le paquet.

En conclusion l'ensemble des tests sont marqués comme PASS. Les autopkgtest sont donc implémenté. J'ai envoyé un mail à la *pan-maintainers* liste afin de demander si je pouvais push mes modifications. Je n'ai d'ailleurs à ce jour jamais obtenu de réponse à ce mail de la part des maintainers.

Haskell-futhark:

Le paquet *Haskell-Futhark* ne possède pas de repository git salsa a proprement parlé. Il faut aller le chercher près du paquet *DHG_packages*. DHG_packages git-salsa .Il s'agit d'un regroupement de paquet codé dans le langage de programmation Haskell. Ce regroupement de paquet contient donc le paquet *Haskell-Futhark* et ses dérivés. Ainsi, pour avoir accès au paquet *Haskell-Futhark*, il faut d'abord récupérer n *DHG_packages* en faisant un *debcheckout*. Une fois fait, aller dans le répertoire *cd /p/haskell-futhark*. De là je procéde de la même manière que le pour les autres paquets en le testant d'abord sans implémentation. Il n'y avait pas d'autopkgtest ROCm dans le répertoire */debian/tests/*. J'ai donc implémenté l'autopkgtest ROCm. Je m'appuie sur l'implémentation du fichier de commande *opencl* et ajoute énormément de dépendances afin que cela rende les tests marqués comme PASS. Les tests ne passaient pas à l'origine il y avait des problèmes de dépendances dans le build , dont voici le fichier *control* nouvellement implémenté.

```
Test-Command: debian/tests/multicore
Depends:
 build-essential,
 ocl-icd-opencl-dev,
 futhark,
 hipcc,
 libclang-common-17-dev,
 xauth,
 xvfb,
 libopenblas-pthread-dev,
 libopenblas0-pthread,
 libgmp-dev,
 libmpfr-dev,
 python3-all,
 libopenblas-dev,
 pkg-haskell-tools,
Features: test-name=haskell-futhark-multicore
Architecture: amd64 arm64 armel armhf i386
Restrictions: flaky, skip-not-installable,allow-stderr

Test-Command: debian/tests/opencl
Depends:
 build-essential,
 futhark,
 pocl-opencl-icd,
 ocl-icd-opencl-dev,
 hipcc,
 libmpfr-dev,
 libopenblas-pthread-dev,
 libopenblas0-pthread,
 libopenblas-dev,
 clinfo,
 libclang-common-17-dev,
 xauth,
 python3-all,
 libgmp-dev,
 xvfb,
 pkg-haskell-tools,
Features: test-name=haskell-futhark-opencl
Restrictions: flaky, skip-not-installable, allow-stderr
Architecture: amd64 arm64 armel armhf i386
Test-Command: rocm-test-launcher debian/tests/hip
Depends:
 build-essential,
 futhark,
 libmpfr-dev,
 hipcc,
 python3-all,
 libopenblas-dev,
 libgmp-dev,
 pkg-haskell-tools,
 libopenblas-pthread-dev,
 libopenblas0-pthread,
 rocminfo,
 pkg-rocm-tools,
 ocl-icd-opencl-dev,
 rocm-opencl-icd,
 libclang-common-17-dev,
 xauth,
 xvfb,
Restrictions: skippable, flaky, skip-not-installable, allow-stderr
Architecture: amd64 arm64 ppc64el
```

Fichier *control* implémenté

Les dépendances suivantes ne peuvent pas être installées ou ont des soucis. C'est la raison pour laquelle la construction du paquet ne fonctionne pas

```
(I)Dose_applications: --checkonly specified, consider all packages as background packages
(I)Dose_applications: Solving...
output-version: 1.2
native-architecture: amd64
report:
 -
  package: sbuild-build-depends-main-dummy
  version: 0.invalid.0
  architecture: amd64
  status: broken
  reasons:
   -
    missing:
     pkg:
      package: libghc-futhark-server-dev
      version: 1.2.3.0-1
      architecture: amd64
      unsat-dependency: libghc-futhark-data-dev-1.1.1.0-41a48:amd64
     depchains:
      -
       depchain:
        -
         package: sbuild-build-depends-main-dummy
         version: 0.invalid.0
         architecture: amd64
         depends: libghc-futhark-server-dev:amd64 (>= 1.2.3.0)
```

Logs d'erreur des dépendances broken et faisant planter le build

De là je ne sais pas comment procéder pour réparer les dépendances. De plus, il y a des erreurs dans les tests. Ces erreurs de dépendances sont nouvelles. Les tests sont stockés quelque part, mais pas dans le paquet lui-même. Quelques testes échouent encore.

En conclusion, les autopkgtest ont été implémentés mais ne passent pas encore pour ces raisons. Le paquet est toujours en cours de traitement mais me pose énormément de problème car je ne sais pas où son exactement situé les tests afin que je regarde le code et comprenne la raison pour laquelle, ils donnent en sortie une erreur.

Sasmodels :

La procédure de lancement sans aucune modification est identique aux paquets précédents. Les tests ne passaient pas car il y avait dépendances qui étaient dit *downgrade* , ce qui empêchait les tests de s'exécuter et de terminer les autopkgtests. Tout d'abord je suis allé dans le fichier suivant *debian/control* afin de résoudre cette erreur sur les paquets *python3-sasmodels* et *python-sasmodels-doc*. J'ai ajouté aux dépendances du paquets *python-sasmodels-doc* les dépendances suivantes : *gcc, libc6-dev | libc-dev, python3-numpy, python3-scipy, hipcc, sasmodels-private-libs,* dans le paquet *python3-sasmodels* j'ai ajouté uniquement *hipcc*. Cela permet de gérer la portabilité entre les GPU AMD et NVIDIA, car il compile sur tout. Ce qui le rend très polyvalent. Je vérifie bien que les tests font de l'Opencl en faisant *grep -r « Opencl »* , le tests *sasmodels/compyle.py* fait de l'OpenCl. Désormais les autopkgtest se lancent sans accros. Les autopkgtest *sasmodels-pocl* et *sasmodels-rocm* sont marqués comme PASS, voici le résultat des deux tests dans les deux cas :

```
compare.py::sasmodels.compare.push_seed PASSED                          [ 50%]
exception.py::sasmodels.exception.annotate_exception PASSED             [100%]
```

```
Test-Command: debian/tests/python3-sasmodels-opencl
Depends:
 python3-all,
 clinfo,
 pocl-opencl-icd,
 python3-docutils,
 python3-matplotlib,
 python3-pytest,
 python3-pyopencl,
 python3-pytest,
 @,
 sasmodels-private-libs,
 python-sasmodels-doc,
 python3-sasmodels,
Features: test-name=sasmodels-pocl
Restrictions: allow-stderr, skip-not-installable,

Test-Command: rocm-test-launcher debian/tests/python3-sasmodels-opencl
Depends:
 python3-all,
 clinfo,
 pkg-rocm-tools,
 rocm-opencl-icd,
 python3-docutils,
 python3-matplotlib,
 python3-pytest,
 python3-pyopencl,
 python3-pytest,
 @,
 sasmodels-private-libs,
 python-sasmodels-doc,
 python3-sasmodels,
Features: test-name=sasmodels-rocm
Restrictions: allow-stderr, skip-not-installable,skippable
```

Fichier *control* après implémentation

```
**Résultats:
========================= 2 passed, 1 warning in 0.90s =========================
autopkgtest [10:42:31]: test sasmodels-rocm: ----------------------]
autopkgtest [10:42:31]: test sasmodels-rocm:  - - - - - - - - - - results - - - -
sasmodels-rocm       PASS
autopkgtest [10:42:32]: @@@@@@@@@@@@@@@@@@@@ summary
python3              PASS
sasmodels-pocl       PASS
sasmodels-rocm       PASS
```

```
Done
The following NEW packages will be installed:
  clang-18 clinfo libclang-common-18-dev libclang-cpp18 libclang1-18 libedit2
  libgc1 libhwloc15 libllvm18 libllvmspirvlib18.1 libobjc-14-dev libobjc4
  libpocl2-common libpocl2t64 libstdc++-14-dev libz3-4 llvm-18-linker-tools
  llvm-spirv-18 ocl-icd-libopencl1 pocl-opencl-icd python3-mako
  python3-markupsafe python3-pyopencl python3-pytools
  python3-typing-extensions
The following packages will be DOWNGRADED:
  python-sasmodels-doc python3-sasmodels
0 upgraded, 25 newly installed, 2 downgraded, 0 to remove and 0 not upgraded.
E: Packages were downgraded and -y was used without --allow-downgrades.
E: Failed to process build dependencies
```

J'ai mis ci-dessus le log d'erreur que j'ai rencontré quand je parlais de paquet dit Downgraded. J'ai ensuite envoyé un mail au maintainers afin de push les modifications à *stuart@debian.org* et *dparson@debian.org* . Les mainteneurs Stuart Prescott et Drew Parson, m'ont posé beaucoup de question notamment sur la question de savoir si le paquet fonctionnerait même s'il n'y avait pas de GPU dédié sur la machine. En réalité cela fonctionnerait toujours j'ai marqué l'autopkgtest de ROCm comme étant en état skippable. Cela signifie que si la condition n'est pas satisfaite le test sera tout simplement skip et cela n'exercera aucune influence sur l'exécution du paquet. J'ai d'ailleurs brièvement explicité ceci aux mainteneurs qui n'avait pas abordé la question de ROCm auparavant. A la différence si la machine possède un GPU ROCm quel qu'il soit, l'autopkgtest ROCm sera en mesure de fonctionner sur ce GPU.

Thanks for working on the packages Clément.
In principle fine to proceed as far as I'm concerned. Stuart is deeper in the SASModels code than I am so let him give the thumbs up.

Can you confirm that the patched package works fine also on "normal" systems without a GPU?

Drew

Dernier mail des mainteneurs

Voici l'archive de Juillet 2025 de mailing list de Debian-pan-maintainers : Alioth list - Debian-pan-maintainers July -2025 Je laisse cette capture afin que l'on puisse consulter les mails.

- [Debian-pan-maintainers] Push ROCm et Pocl autopkgtest pour le pkg Sasmodels *LONGEAC Clément*
  - [Debian-pan-maintainers] Lancement de ROCm et Pocl autopkgtest pour le paquet Sasmodels *Drew Parsons*
    - [Debian-pan-maintainers] Push ROCm et Pocl autopkgtest pour le pkg Sasmodels *LONGEAC Clément*
      - [Debian-pan-maintainers] Lancement de ROCm et Pocl autopkgtest pour le paquet Sasmodels *Stuart Prescott*
      - [Debian-pan-maintainers] Push ROCm et Pocl autopkgtest pour le pkg Sasmodels *LONGEAC Clément*
      - [Debian-pan-maintainers] Lancement de ROCm et Pocl autopkgtest pour le paquet Sasmodels *Drew Parsons*

En conclusion, dans le but de push mes modifications Frédéric-Emmanuel PICCA, m'a conseillé de créer une nouvelle branche à l'aide la commande *git checkout -b <nom de la branche> ; git add . ; git commit ; git push -u origin <nom de la branche>* , cette branche s'appelle *pu-rocm* . La création de cette nouvelle branche permet de laisser le choix mainteneurs de faire un merge sur la branche master ou non plutôt que de faire un merge request qui prendrai beaucoup de temps. Ainsi, les modifications ont été implémentée pour les autopkgtest ROCm et POCL, les tests passent et ont été push sur une nouvelle branche.

Hashcat :

La procédure de lancement sans aucune modification est identique aux paquets précédents. Il n'avait pas de fichier *control*, ni *opencl*. J'ai implémenté et fait de nombreux patchs pour les autopkgtest ROCm et POCL,mais il y a encore des erreurs. Il reste encore des erreurs que je ne parviens pas à résoudre, notamment sur l'exécution des tests eux-mêmes en .vc .pl .sh et .pm .

Libhmbeagle :

La procédure de lancement sans aucune modification est identique aux paquets précédents. En réalité il n'y avait aucun autopkgtest quel qu'il soit. Ni de fichier *debian/tests* dans lequel il n'y avait pas non plus de fichier *control* ni *opencl*. Le paquet a été lancer sans implémentation. Sans surprise le paquet se construisait sans accros, justement puisqu'il n'y avait pas de tests. Désormais il faut implémenter les autopkgtest, je créer le répertoire *mkdir tests* dans le directory *libhmbeagle/debian* et les fichiers *control* et *opencl.sh* en faisant *touch <nom du fichier>*. Je rends ensuite exécutable le script shell exécutable en faisant *chmod +x opencl.sh*. J'ai d'abord créé le fichier *control* en ajoutant toutes les dépendances, architectures, features etc

```
Test-Command: debian/tests/opencl.sh
Depends:
 @,
 clinfo,
 pocl-opencl-icd,
 python3-all,
 python3-genx,
 python3-numba,
 python3-pytest,
 python3-pytest-mock,
 python3-pytest-xvfb,
 xauth,
 xvfb,
 python3-pooch,
Features: test-name=opencl-pocl
Restrictions: allow-stderr, flaky, skip-not-installable,

Test-Command: rocm-test-launcher debian/tests/opencl.sh
Depends:
 @,
 clinfo,
 pkg-rocm-tools,
 rocm-opencl-icd,
 python3-all,
 python3-genx,
 python3-pooch,
 python3-numba,
 python3-pytest,
 libclang-common-17-dev,
 python3-pytest-mock,
 python3-pytest-xvfb,
 hipcc,
 rocm-device-libs-17,
 xauth,
 xvfb,
Features: test-name=opencl-rocm
Restrictions: allow-stderr, flaky, skip-not-installable, skippable
```

Fichier *control* implémenté

Je prends en compte les erreurs que nous avons pu commettre auparavant, j'ajoute au dépendances *hipcc* et *libclang-common-17-dev*. Et suis la procédure pour implémenter les tests ROCm et POCL. Dans un second temps j'ai implémenté le fichier de command, celui qui servira dans le fichier *control* dans la section *Test-Command*. J'ai fait de

```
#!/bin/bash
set -euf

#export HOME="$AUTOPKGTEST_TMP"

#cd "$(dirname "$0")/../.."  # remonte à la racine du projet libhmsbeagle

#echo "Running all test scripts in tests/"

#find tests/ -maxdepth 1 -type f -name '*.sh' -print | while read -r script; do
#    echo "Running $script"
#    sh "$script"
#done

# Aller dans le répertoire des tests
cd tests

# Exécuter tous les scripts .sh avec bash
for test_script in *.sh */*.sh; do
    if [ -f "$test_script" ]; then
        echo "Running $test_script"
        bash "$test_script" || true
    fi
done
```

Fichier *opencl.sh* implémenté

nombreux essai en créant plusieurs fichiers que j'ai par la sui supprimé. Mais j'avoue avoir beaucoup réfléchi sur la manière d'implémenter astucieusement ce fichier et de le faire correctement et proprement.

La raison pour laquelle j'ai souhaité écrire un script en shell est que les tests du paquet sont en .sh . Lorsque j'ai voulu lancer les tests cela n'a pas totalement fonctionner. Comme peut le montrer le code commenté. Cependant, j'ai réalisé que je pouvais quand même le faire avec bash. J'ai eu pas mal de problème avec cela car je ne savais pas comment exécuter des scripts en .sh , je savais facilement le faire avec des scripts en .py mais en .sh. J'en ai conclu que cela devait certainement plus ou moins la même chose. Sauf que j'ai galéré une demi-journée pour trouver la syntaxe exacte me permettant d'exécuter tous les tests .sh de tous les directory internet à celui-ci *libhmbeagle/tests* . J'ai donc opté pour le code que l'on voit ci-dessus et qui n'est pas commenté. Cependant il y avait quelque erreur dans les scripts de tests en .sh. Ce qui conduisait les autopkgtest à retourner une erreur. J'ai fait un patch dessus, il s'agit du test ,*test_ressource.sh*.

```
---
 tests/test_resource.sh | 24 ++++++++++++------------
 1 file changed, 12 insertions(+), 12 deletions(-)

diff --git a/tests/test_resource.sh b/tests/test_resource.sh
index c6f9e0c..aad3e30 100755
--- a/tests/test_resource.sh
+++ b/tests/test_resource.sh
@@ -1,14 +1,14 @@
 #!/bin/bash

-if [ -z "${1}" ];
-then
-    echo "Usage: test_resource.sh resource_number"
-    echo
-    ../examples/synthetictest/synthetictest.cpp --resourcelist
-else
-    TEST_RESOURCE=${1}
-    echo "Testing resource..."
-    ../examples/synthetictest/synthetictest.cpp --resourcelist | grep "Resource $TEST_RESOURCE" -A 1 | grep -o --color=never ': .*'
-    ./run_tests.sh $TEST_RESOURCE 2>&1 | grep "\*\*\*[^:]*:" -B 2
-    echo "All tests completed."
-fi
+if [ -z "${1}" ];
+then
+    echo "Usage: test_resource.sh resource_number"
+    echo
+     ../examples/synthetictest/synthetictest --resourcelist
+else
+    TEST_RESOURCE=${1}
+    echo "Testing resource..."
+    ../examples/synthetictest/synthetictest --resourcelist | grep "Resource $TEST_RESOURCE" -A 1 | grep -o --color=never ': .*'
+    ./run_tests.sh $TEST_RESOURCE 2>&1 | grep "\*\*\*[^:]*:" -B 2
+    echo "All tests completed."
+fi
```

Patch du test, *test_ressource.sh*

Ce patch a permis de rendre les autopkgtest possible que ce soit au niveau de POCL et de ROCm. Tous sont marqués comme PASS. J'ai d'ailleurs fait un mail à la *debian-pan-maintainers* et au mainteneur du paquet afin de push les modifications. J'ai alors été autorisé à le faire. J'ai push sur la branche master.

J'ai également été ajouté au groupe Debian Med et au projet partagé Debian Deep Learning.

En conclusion, les autopkgtest POCL et ROCm ont été implémentés et push sur la branche *master*.

Pyopencl :

La procédure de lancement sans aucune modification est identique aux paquets précédents. Le test POClL seul est passé lors du lancement sans implémentation mais lorsque l'on ajoute ROCm les tests ont échoué.

```
Error: Failed to compile source (from CL or HIP source to LLVM IR).
autopkgtest [16:35:09]: test pyopencl-rocm:  - - - - - - - - - - results - - - - - - - - - -
pyopencl-rocm        FAIL non-zero exit status 1
autopkgtest [16:35:09]:  - - - - - - - - - - running shell - - - - - - - - - -
bash: cannot set terminal process group (1840): Inappropriate ioctl for device
bash: no job control in this shell
```

Dont voici le log d'erreur. On constate que le code a du mal avec les calculs HIP. Ainsi, lors de l'implémentation je décide d'ajouter aux dépendances celles-ci dans le but de résoudre cette erreur ; *hipcc* et *rocm-device-libs-17*. Pour avoir plus de détail, on peut aller sur https://packages.debian.org/sid/hipcc . Voici le fichier control implémenté avec la partie ROCm que j'ai ajouté en suivant encore une fois la procédure. C'est-à-dire en ajoutant le script *rocm-test-launcher* . C'est un script ou un outil fourni par la suite ROCm (Radeon Open Compute). Son but principal est de configurer et préparer un environnement de test isolé pour exécuter des tests qui nécessitent un runtime ROCm fonctionnel. *Hipcc* sert pour compiler le code HIP (langage pour GPU d'AMD) pendant la construction des tests. Il traduit le code compatible CUDA/HIP en code exécutable sur les GPU AMD. *rocm-device-libs-17* est essentiel pour que le code compilé par *hipcc* puisse s'exécuter correctement sur le matériel AMD concret. Cela a permis de résoudre le problème. Il n'y a pas de fichier *opencl* ici car le code est déjà inclus dans le fichier *control* c'est un cas un peu particulier mais cela se fait en pratique.

Fichier *control* implémenté

```
Test-Command: set -efu
 ; cp -a test "$AUTOPKGTEST_TMP"
 ; for py in $(py3versions -r 2>/dev/null)
 ; do cd "$AUTOPKGTEST_TMP/test"
 ; echo "Testing with $py:"
 ; $py -m pytest --verbosity=2 2>&1
 ; done
Depends:
 python3-all,
 python3-pyopencl,
 python3-pytest,
 @,
 python3-scipy,
 clinfo,
 pocl-opencl-icd
Features: test-name=pyopencl-pocl
Architecture: amd64, arm64, armel, armhf, i386
Restrictions: allow-stderr, needs-internet, skip-not-installable

Test-Command:rocm-test-launcher
 ; set -efu
 ; cp -a test "$AUTOPKGTEST_TMP"
 ; for py in $(py3versions -r 2>/dev/null)
 ; do cd "$AUTOPKGTEST_TMP/test"
 ; echo "Testing with $py:"
 ; $py -m pytest --verbosity=2 2>&1
 ; done
Depends:
 python3-all,
 python3-pyopencl,
 python3-pytest,
 @,
 python3-scipy,
 clinfo,
 libclang-common-17-dev,
 pkg-rocm-tools,
 rocm-opencl-icd,
 rocm-device-libs-17,
 hipcc
Features: test-name=pyopencl-rocm
Architecture: amd64, arm64, armel, armhf, i386
Restrictions: allow-stderr, needs-internet, skip-not-installable
```

```
====== 298 passed, 7 skipped, 2 xfailed, 381 warnings in 95.11s (0:01:35) ======
autopkgtest [17:08:00]: test pyopencl-rocm: ----------------------]
autopkgtest [17:08:01]: test pyopencl-rocm:  - - - - - - - - - - results - - - - - -
pyopencl-rocm         PASS
autopkgtest [17:08:01]: @@@@@@@@@@@@@@@@@@@@@@ summary
pyopencl-pocl         PASS
pyopencl-rocm         PASS
```

Voici les tests marqués comme PASS à l'issu du build du paquet pyopencl.

En conclusion les autopkgtest ROCm et POCL ont été implémentés, les tests sont marqués comme PASS. J'ai commis toutes les modifications mais pas encore PUSH. Nous n'avons pas le droit de push les modifications sur un paquet qui ne nous appartient pas, je ne l'ai donc pas push selon les conseils de Frédéric-Emmanuel PICCA. J'ai envoyé un mail aux mainteneurs et à la debian-pan-maintainers liste afin de push les modifications. Je n'ai eu à ce jour aucune réponse.

Compyle :

La procédure de lancement sans aucune modification est identique aux paquets précédents. Lors du lancement sans les autopkgtest, ceux-ci échouent. Il y a des erreurs dans les tests. Désormais j'implémente les autopkgtest POCL et ROCm. Je commence par créer le fichier *control* et le fichier *opencl1.sh* dans le répertoire *debian/tests/*.

```
Test-Command: debian/tests/opencl1.sh
Depends:
 clinfo,
 pocl-opencl-icd,
 ocl-icd-opencl-dev,
 hipcc,
 libclang-common-17-dev,
 python3-all,
 python3-pyopencl,
 python3-pytest,
 python3-scipy,
Architecture: amd64, arm64, armel, armhf, i386
Features: test-name=opencl-pocl
Restrictions: allow-stderr, skip-not-installable

Test-Command: rocm-test-launcher debian/tests/opencl1.sh
Depends:
 clinfo,
 rocm-opencl-icd,
 pkg-rocm-tools,
 ocl-icd-opencl-dev,
 hipcc,
 libclang-common-17-dev,
 python3-all,
 python3-pyopencl,
 python3-pytest,
 python3-scipy
Architecture: amd64, arm64, armel, armhf, i386
Features: test-name=opencl-rocm
Restrictions: allow-stderr, skip-not-installable, skippable
```

Fichier *control* implémenté

Les tests qui échouaient jusqu'à présent ont été ignorer car cela rendait le passage beaucoup plus aisé. Il s'agit des tests contenus dans le directory *Compyle/tests/*, ce sont les tests *test_array.py* et *test_low_level.py* . Je l'ai implémenté dans le fichier *debian/rules* afin que cela soit global plutôt que local. C'est-à-dire que les tests skip, ne sont pas skip au niveau du fichier *opencl* mais directement au niveau du fichier *rules*.

```
--------------------------------- debian/rules ---------
index 5f21401..f1953de 100755
@@ -5,2 +5,3 @@ export PYOPENCL_CTX=0
 export PYBUILD_NAME=compyle
+
 export PYBUILD_TEST_ARGS=\
@@ -11,2 +12,4 @@ and not test_const_in_return" \
  --ignore=$(CURDIR)/compyle/tests/test_parallel.py \
+--ignore=$(CURDIR)/compyle/tests/test_array.py \
+--ignore=$(CURDIR)/compyle/tests/test_low_level.py \
 $(CURDIR)/compyle/tests
```

De plus, pour réaliser cela j'ai écrit le fichier de fichier de test-Command *opencl1.sh* .Le but de ce script est d'exécuter tous les fichiers de tests python contenus dans le répertoire *compyle/tests/* et son arborescence.

```
#!/bin/sh -e
export HOME="$AUTOPKGTEST_TMP"
export POCL_WORK_GROUP_METHOD=c
set -efu

cd "$(dirname "$0")/.."  # remonte à la racine du dépôt source

for py in $(py3versions -r 2>/dev/null); do
    echo "Testing with $py:"
    $py -m pytest -v compyle/tests/
done
```

Une fois que tout ceci a été implémenté je relance le build. Il se trouve que désormais les tests passent sans difficultés. Il faut avoir conscience que le processus de recherche des dépendances, de compréhension etc a pris quelques jours.

En conclusion, les autopkgtest POCL et ROCm ont été implémentés, un mail a été également fait pour la debian-pan et pour le mainteneur. A ce jour je n'ai eu aucune réponse à mon mail.

Gpyfft :

La procédure de lancement sans aucune modification est identique aux paquets précédents. En réalité il n'y avait que la partie python, pas d'autopkgtest pour POCL ni ROCm. Il possédait un *debian/tests* dans lequel il n'y avait pas de fichier *opencl*. Le paquet a été lancé sans implémentation. Sans surprise le paquet se construisait sans accros, justement puisqu'il n'y avait pas de tests POCL ni ROCm. J'ai donc implémenté les autopkgtest pour POCL et ROCm.

```
Test-Command: set -efu
 ; for py in $(py3versions -r 2>/dev/null)
 ; do cd "$AUTOPKGTEST_TMP"
 ; echo "Testing with $py:"
 ; $py -m pytest --pyargs gpyfft.test
 ; done
Depends: python3-all, python3-gpyfft,python3-pyopencl, python3-pytest, python3-pytest-mock, @, clinfo, pocl-opencl-icd
Features: test-name=gpyfft-pocl
Architecture: amd64, arm64, armel, armhf, i386
Restrictions: allow-stderr, skip-not-installable


Test-Command: rocm-test-launcher
 ; set -efu
 ; for py in $(py3versions -r 2>/dev/null)
 ; do cd "$AUTOPKGTEST_TMP"
 ; echo "Testing with $py:"
 ; $py -m pytest --pyargs gpyfft.test
 ; done
Depends:
 python3-all,
 python3-gpyfft,
 python3-pyopencl,
 python3-pytest,
 python3-pytest-mock,
 @,
 clinfo,
 rocm-opencl-icd,
 libclang-common-17-dev,
 pkg-rocm-tools,
 rocm-device-libs-17,
 hipcc
Features: test-name=gpyfft-rocm
Architecture: amd64, arm64, armel, armhf, i386
Restrictions: allow-stderr, skip-not-installable, skippable
```

Fichier *control* implémenté

J'ai inclus les Test-Command dans le fichier control car cela fonctionnait beaucoup que quand ils étaient séparés dans le fichier *opencl*. Le contenu du fichier *opencl* est identique au contenu des cellules Test-Command dans le fichier ci-dessus. L'implémentation était en réalité assez rapide car il n'y avait qu'à exécuter les tests python à l'aide de *pytest*. Une fois tout ceci implémenté j'ai relancé le build. Cela a permis de faire passer les test POCL et ROCm. De cette manière ces tests sont marqués comme PASSED. A la demande de Frédéric-Emmanuel PICCA j'ai pu push *gpyfft* sans faire de mail aux mainteneurs. En conclusion, les autopkgtest POCL et ROCm ont été implémentés et push sur la branche *master*.

```
Passage du paquet gpyfft :

autopkgtest [09:34:24]: test autodep8-python3:  - - - - - - - - - - results - - - - - - - - - -
autodep8-python3     PASS (superficial)
autopkgtest [09:34:24]: @@@@@@@@@@@@@@@@@@@@ summary
gpyfft-pocl          PASS
gpyfft-rocm          PASS
autodep8-python3     PASS (superficial)
```

Pyvkfft :

La procédure de lancement sans aucune modification est identique aux paquets précédents. En réalité il n'y avait que la partie python, pas d'autopkgtest pour POCL ni ROCm. Il possédait un *debian/tests* dans lequel il y avait un fichier *opencl* écrit par Frédéric-Emmanuel PICCA un an plus tôt. Le paquet a été lancé sans implémentation. Sans surprise le paquet se construisait sans accros, justement puisqu'il n'y avait pas de tests POCL ni ROCm. J'ai donc implémenté les autopkgtest pour POCL et ROCm.

```
Test-Command: debian/tests/opencl
Depends:
 clinfo,
 @,
 python3-pyopencl,
 python3-all,
 python3-pyvkfft,
 pocl-opencl-icd,
Features: test-name=pyvkfft-pocl
Architecture: amd64, arm64, armel, armhf, i386
Restrictions: allow-stderr, skip-not-installable, flaky

Test-Command: rocm-test-launcher debian/tests/opencl
Depends:
 clinfo,
 @,
 rocm-opencl-icd,
 pkg-rocm-tools,
 python3-pyopencl,
 python3-all,
 python3-pyvkfft,
 libclang-common-17-dev,
 rocm-device-libs-17,
 hipcc,
Features: test-name=pyvkfft-rocm
Restrictions: allow-stderr, skip-not-installable, flaky, skippable
```

Fichier *control* implémenté

Après cela je vérifie la présence d'opencl par l'utilisation de l'outil *grep -r « opencl »* ou encore de l'utilisation de l'arbre de dépendance du paquet *Debtree pyvkfft* . En effet, pyvkfft fait bien de l'opencl. Les tests sont marqués comme PASS. Je n'ai pas eu à toucher le fichier Opencl, car il fonctionnait.

```
Passage du paquet pyvkfft :

autopkgtest [17:50:07]: test pyvkfft-rocm: ----------------------]
autopkgtest [17:50:07]: test pyvkfft-rocm:  - - - - - - - - - - results - -
pyvkfft-rocm         PASS
autopkgtest [17:50:07]: @@@@@@@@@@@@@@@@@@@@ summary
pyvkfft-pocl         PASS
pyvkfft-rocm         PASS
```

Cela a permis de faire passer les test POCL et ROCm. De cette manière ces tests sont marqués comme PASSED. A la demande de Frédéric-Emmanuel PICCA j'ai pu push *pyvkfft* sans faire de mail aux mainteneurs. En conclusion, les autopkgtest POCL et ROCm ont été implémentés et push sur la branche *master*.

Silx :

La procédure de lancement sans aucune modification est identique aux paquets précédents. Lors du lancement sans implémentation les autopkgtest *opencl* , *no-opencl* sont marqués comme PASS. Cependant, l'autopkgtest *command1* est marqué comme FAIL avec comme code d'erreur 1.

```
Résultat :
        no-opencl            PASS
        opencl               PASS
        command1             FAIL non-zero exit status 1

        E: Autopkgtest run failed.

        +------------------------------------------------------------------------------+
        | Cleanup                                  Thu, 15 May 2025 14:28:58 +0000 |
        +------------------------------------------------------------------------------+

        Purging /build/reproducible-path
        Not cleaning session: cloned chroot in use

        +------------------------------------------------------------------------------+
        | Summary                                  Thu, 15 May 2025 14:28:59 +0000 |
        +------------------------------------------------------------------------------+

        Autopkgtest: fail
        ------------------------------------------------------------------------------
        Build Architecture: amd64
        Build Type: binary
        Build-Space: 422772
        Build-Time: 232
        Distribution: unstable
        Host Architecture: amd64
        Install-Time: 70
        Job: /home/experiences/grades/longeac/silx_2.2.1+dfsg-3.dsc
        Lintian: fail
        Machine Architecture: amd64
        Package: silx
        Package-Time: 312
        Piuparts: fail
        Source-Version: 2.2.1+dfsg-3
        Space: 422772
        Status: successful
        Version: 2.2.1+dfsg-3
        ------------------------------------------------------------------------------
        Finished at 2025-05-15T14:17:32Z
        Build needed 00:05:12, 422772k disk space
        dgit ok: build successful, results in silx_2.2.1+dfsg-3_multi.changes
```

Lors de l'analyse de l'erreur j'ai pu constater qu'il y a un problème avec le test command, celui-ci ne fonctionne pas lorsque l'on lance la commande :

```
longeac@grades-01:~/silx/debian/tests$
xvfb-run -s "-screen 0 1024x768x24  -ac
+extension GLX +render -noreset" sh
debian/tests/gui
xvfb-run: error: Xvfb failed to start
```

L'erreur provient du fichier gui, il déclare glxinfo mais mesa-utils n'était pas installé :

```
longeac@grades-01:~$ glxinfo
Error: unable to open display
```

Afin de résoudre cette erreur j'ai installé *mesa-utils* à l'aide de la commande suivante : *sudo apt install mesa-utils*

Les fichiers *gui*, *no-opencl* et *opencl* n'ont pas été modifiés, seul le fichier *control* a été modifié. Au départ j'avais créé plusieurs fichier opencl , un pour POCL et un pour ROCm. Seulement, afin de travailler proprement, il faut en faire un unique fichier qui gère l'*opencl* et ensuite distinguer les tests par les *features*.

```
Tests: no-opencl
Depends:
 @,
 python3-all,
 python3-pytest,
 python3-pytest-mock,
 python3-pytest-xvfb,
 xauth,
 xvfb,
Restrictions: allow-stderr

Tests: debian/tests/opencl
Depends:
@,
clinfo,
pocl-opencl-icd,
python3-all,
python3-pytest,
python3-pytest-mock,
python3-pytest-xvfb,
xauth,
xvfb,
Architecture: !i386
Restrictions: allow-stderr

Test-Command: rocm-test-launcher debian/tests/opencl
Depends:
 @,
 clinfo,
 rocm-opencl-icd,
 pkg-rocm-tools,
 python3-all,
 python3-pytest,
 python3-pytest-mock,
 python3-pytest-xvfb,
 xauth,
 xvfb,
 libclang-common-17-dev,
Architecture: !i386
Features: test-name=opencl-ROCm
Restrictions: allow-stderr , skippable

Test-Command: xvfb-run -s "-screen 0 1024x768x24 -ac +extension GLX +render -noreset" sh debian/tests/gui
Depends:
 mesa-utils,
 silx,
 xauth,
 xvfb,
Restrictions: allow-stderr
```

Fichier *control* implémenté

J'ai implémenté les autopkgtest ROCm et POCL. Je relance *sbuild* après l'implémentation dont voici les résultats. Je n'avais pas encore implémenté le containers *podman+rocm*. C'est pour cette raison que le test ROCm est marqué comme SKIP avec le code 77.

```
trapped status 143 and return success
autopkgtest [11:55:58]: test command1: ----------------------]
autopkgtest [11:55:59]: test command1:  - - - - - - - - - - results - - -
command1             PASS
autopkgtest [11:56:01]: @@@@@@@@@@@@@@@@@@@@ summary
no-opencl            PASS
opencl               PASS
opencl-ROCm          SKIP exit status 77 and marked as skippable
command1             PASS
```

Avec l'ajout de podman+rocm :

```
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-rocm'
trapped status 143 and return success
autopkgtest [11:08:08]: test command1: ----------------------]
autopkgtest [11:08:08]: test command1:  - - - - - - - - - - results - - -
command1             PASS
autopkgtest [11:08:08]: @@@@@@@@@@@@@@@@@@@@ summary
no-opencl            PASS
opencl               PASS
opencl-rocm          PASS
command1             PASS
```

J'ai également modifié le fichier *config.pl* pour obtenir le fichier quel'on connait plus haut et ajouté *libclang-common-17-dev* au fichier *control*

```
autopkgtest [17:09:46]: test command1:  - - - - - - - - - - results - - - -
command1             PASS
autopkgtest [17:09:47]: @@@@@@@@@@@@@@@@@@@@ summary
no-opencl            PASS
opencl               PASS
opencl-ROCm          PASS
command1             PASS
```

Cela a permis de faire passer les test POCL et ROCm. De cette manière ces tests sont marqués comme PASSED. A la demande de Frédéric-Emmanuel PICCA j'ai pu push *silx* sans faire de mail aux mainteneurs. En conclusion, les autopkgtest POCL et ROCm ont été implémentés et push sur la branche *master*.

Pyfai :

La procédure de lancement sans aucune modification est identique aux paquets précédents. Les autopkgtests sont lancés ici à l'aide de la commande *autopkgtest -B . -- schroot unstable-amd64-sbuild*. Rien n'a encore été modifié. Cette commande est lancée sans le container *podman-rocm*. Il n'était pas encore implémenté. Dont voici le résultat. Le test *no-opencl* est skip tandis que les autres passent. En réalité le passage des tests va s'avérer bien plus ardu et imprévisible que ce qu'il n'y parait.

```
command1             PASS
autopkgtest [16:30:13]: @@@@@@@@@@@@@@@@@@@@ summary
no-opencl            SKIP Test declares architecture as not supported: amd64
no-opencl            SKIP Test declares architecture as not supported: amd64
opencl               PASS
pyfai-rocm           PASS
command1             PASS
```

```
#!/bin/sh -e

# Test OpenCL using ROCm ICD (GPU)
# This script assumes rocm-opencl-icd is installed and working
# ROCm drivers must be correctly configured on the host system

export PYFAI_OPENCL=True
export PYOPENCL_COMPILER_OUTPUT=1

# skip test
# TestAzimHalfFrelon.test_medfilt1d

cp bootstrap.py run_tests.py pyproject.toml version.py README.rst "$AUTOPKGTEST_TMP"

for py in $(py3versions -s 2>/dev/null)
do cd "$AUTOPKGTEST_TMP"
   echo "Testing with $py:"
   xvfb-run -a --server-args="-screen 0 1024x768x24" $py run_tests.py -v -m --low-mem --installed
done
```

Le fichier de commande pour les tests faisant appel à l'opencl, soit les autopkgtest POCL et ROCm.

Je passe désormais à l'implémentation de ces autopkgtest dans le fichier *control*.

```
# tests that must pass

Test-Command: no-opencl
Architecture: !amd64 !arm64 !armel !armhf !i386
Depends:
 bitshuffle,
 python3-all,
 python3-pyfai,
 python3-tk,
 xauth,
 xvfb,
 python3-pyqt5.qtopengl,
 python3-pyqt5,
 libgl1-mesa-glx,
Features: test-name=no-opencl
Restrictions: allow-stderr, skip-not-installable


Test-Command: rocm-test-launcher debian/tests/opencl
Architecture: amd64 arm64 armel armhf i386
Depends:
 bitshuffle,
 clinfo,
 rocminfo,
 libnuma1,
 ocl-icd-libopencl1,
 rocm-opencl-icd,
 pkg-rocm-tools,
 python3-all,
 python3-pyfai,
 python3-tk,
 xauth,
 xvfb,
 libclang-common-17-dev,
 hipcc,
 rocm-device-libs-17,
Features: test-name=opencl-rocm
Restrictions: allow-stderr, skip-not-installable, skippable

Test-Command: debian/tests/opencl
Architecture: amd64 arm64 armel armhf i386
Depends:
 bitshuffle,
 pocl-opencl-icd,
 clinfo,
 python3-all,
 python3-pyfai,
 python3-tk,
 xauth,
 xvfb,
 libclang-common-17-dev,
Features: test-name=opencl-pocl
Restrictions: allow-stderr, skip-not-installable

Test-Command: xvfb-run -s "-screen 0 1024x768x24 -ac +extension GLX +render -noreset" sh debian/tests/gui
Depends:
 debhelper,
 mesa-utils,
 @,
 xauth,
 xvfb,
Restrictions: allow-stderr
```

Fichier *control* implémenté

Désormais le container *podman+rocm* est implémenté. Après cela j'obtiens de nombreuses erreurs, dues notamment au fait que les autopkgtest prennent trop de temps. Il faut savoir que le paquet Pyfai prend en moyenne 4h à construire. Les autopkgtest possèdent une limite de temps d'exécution que l'on peut lever manuellement en ajoutant dans le fichier *config.pl* l'option d'autopkgtest, *'--timeout=43200'* , cela correspond à plusieurs heures en plus de la limite d'origine. Il faut donc l'augmenter si l'on ne veut pas que les tests soient marqués comme Timed Out. Cela a d'ailleurs été le cas. J'ai mis plusieurs jours, à trouver dans les docs comment faire. Notamment dans la *bookworm* de sbuild, et des commandes possibles pour gérer ce problème dans le fichier config.pl . Après avoir considérablement augmenté le temps pour les autopkgtest je me suis heurté à un deuxième souci. Celui de la pile de travail de la carte elle-même. Je mets le test ROCm en premier afin qu'il s'exécute et voir s'il passe car cela reste la priorité. Mais j'ai plusieurs problèmes avec lui, il met très longs temps à se construire, de sorte que certains tests sont marqués comme expirés quoi que je fasse. Dans les parties ROCm, j'ai l'erreur "Taille maximale valide du groupe de travail 256 sur l'appareil <pyopencl. Dispositif 'gfx1034' sur 'AMD Accelerated Parallel Processing' à 0xe90bf90> 0.0 1.8714113798181157e-05 ". Je ne sais pas comment résoudre cela et d'où cela vient... J'ai fait beaucoup de recherches et je ne sais pas vraiment comment le résoudre. Cela semble être matériel, pour le résoudre, nous devons avoir un GPU AMD marqué PRO, pas une carte graphique de jeu. J'ai fait un mail à la debian science team (Message pour pyfai) pour en savoir plus et le mainteneur Jerôme Kieffer de l'ESRF, m'a répondu à ce sujet, expliquant que c'est un aspect étrange et qu'il n'avait pas vraiment pensé la question de ROCm. Ajout des dépendances au fichier *debian/tests/control*: Pour rocm: *rocminfo, libnuma1, ocl-icd-libopencl1* Pour no-opencl: *python3-pyqt5, python3-pyqt5.qtopengl, libgl1-mesa-glx,*

```
INFO:memProf:Time: 60.074s RAM: 0.000 Mb pyFAI.test.test_containers.TestContainer.test_rebin1d
=============FAIL: testPyfaiBenchmark (pyFAI.test.test_scripts.TestScriptsHelp.testPyfaiBenchmark) -----
 Traceback (most recent call last): File "/usr/lib/python3/dist-packages/pyFAI/test/test_scripts.py",
line 105, in testPyfaiBenchmark self.executeAppHelp("pyFAI-benchmark", "pyFAI.app.benchmark")
~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "/usr/lib/python3/dist-packages/pyFAI/test/test_scripts.py",
line 86, in executeAppHelp self.executeCommandLine(command_line, env) ~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^
File "/usr/lib/python3/dist-packages/pyFAI/test/test_scripts.py",
 line 79, in executeCommandLine self.assertEqual(p.returncode, 0) ~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^ AssertionError: 1 != 0
Maximum valid workgroup size 256 on device <pyopencl.Device 'gfx1034' on 'AMD Accelerated Parallel Processing' at 0xe90bf90> 0.0 1.871411379818157e-05
autopkgtest [18:23:38]: test opencl-rocm: ----------------------]
autopkgtest [18:23:38]: test opencl-rocm: - - - - - - - - - - results - - - - - - - - - - opencl-rocm FAIL non-zero exit status 1
```

Voici l'erreur concernant la pile de travail

J'ai ensuite rendu les tests plus verbeux et j'ai pu détecter les erreurs. Cependant, ce problème de pile de travail me prend un temps considérable car je ne sais pas comment faire. Cordell Bloor, a écrit ceci : « Je ne pense pas que cela soit lié à votre GPU. Je pense que le problème vient du fait que votre code tente de lancer trop de threads au sein d'un groupe de travail lorsqu'il exécute un noyau. Le nombre maximal de threads par groupe de travail (ou, dans la terminologie CUDA, threads par bloc) sur lesquels un noyau peut être exécuté simultanément est déterminé à la compilation. Il est de 256 par défaut, mais peut être porté à 1024 en utilisant l'option de compilation --gpu-max-threads-per-block=1024 (ou en annotant directement le noyau concerné). L'augmentation du nombre de threads par bloc réduit le nombre de registres disponibles pour chaque thread, ce qui peut réduire les performances. Une alternative plausible serait que le code tente de lancer trop de groupes de travail/blocs à chaque lancement du noyau, plutôt que simplement trop de threads par groupe de travail. Si tel est le cas, je ne connais pas d'autre solution que de corriger le code. Je crois qu'il existe également une limite de 256 blocs par lancement du noyau. » En clair, il n'y a pas de bonne solution. En conclusion, les autopkgtest ont été implémentés mais il reste encore de nombreuses erreurs que je ne sais malheureusement pas résoudre au vu de ma faible expérience. Je me suis heurté à des difficultés tant techniques que conceptuelle sur le sujet.

Python-dtcwt :

La procédure de lancement sans aucune modification est identique aux paquets précédents. Le lancement sans implémentation s'est soldé par des erreurs dans le build lui-même, pas dans les autopkgtest. Le paquet était jusqu'à présent sous Numpy 2.0, en conséquence certaine ligne de code python ne fonctionnaient plus. J'ai d'ailleurs fait un patch à ce sujet, j'ai remplacé dans l'ensemble du paquet *asfarray* par *asarray*, pour passer sous le nouveau nom de cette fonction. J'ai procédé de la même manière pour la fonction *assubsctype* que j'ai remplacé par *issubsdtype*.

```
ERROR :AttributeError: `np.asfarray` was removed in the NumPy 2.0 release. Use `np.asarray` with a proper dtype instead.
E: pybuild pybuild:389: test: plugin distutils failed with: exit code=1:
cd /build/reproducible-path/python-dtcwt-0.14.0/.pybuild/cpython3_3.13_dtcwt/build; python3.13 -m pytest tests
dh_auto_test: error: pybuild --test --test-pytest -i python{version} -p 3.13 returned exit code 13
make: *** [debian/rules:12: binary] Error 25
dpkg-buildpackage: error: debian/rules binary subprocess returned exit status 2
```

J'ai ensuite merge les branches *patch-queue/debian/master* et la branche *master*. J'ai ensuite implémenté le fichier *debian/tests/control* . Les autopkgtest se lancent mais il y a des erreurs de code dans les tests.

```
========================== short test summary info ==========================
FAILED tests/test_againstmatlab.py::test_transform3d_numpy - AssertionError: ...
FAILED tests/test_ifm1.py::test_float32_input - AssertionError: assert False
FAILED tests/test_registration.py::test_estimatereg - ValueError: solve: Inpu...
FAILED tests/test_xfm1.py::test_integer_input - assert np.False_
FAILED tests/test_xfm1.py::test_integer_perfect_recon - AssertionError: asser...
FAILED tests/test_xfm2.py::test_integer_input - assert np.False_
FAILED tests/test_xfm2.py::test_integer_perfect_recon - AssertionError: asser...
FAILED tests/test_xfm3.py::test_integer_input - assert np.False_
FAILED tests/test_xfm3.py::test_integer_perfect_recon - AssertionError: asser...
FAILED tests/test_xfm3.py::test_float32_input - assert np.False_
FAILED tests/test_xfm3.py::test_float32_recon - assert np.False_
========== 11 failed, 114 passed, 214 skipped, 16 warnings in 4.88s ==========
```

Je fais ensuite des patchs pour résoudre ces erreurs mais je prends beaucoup de temps en raison des problèmes de fusion. Je ne parviens pas à résoudre les erreurs contenues dans les tests, il s'agit d'erreurs de calcul. C'est-à-dire que le résultat en sortie n'est pas celui souhaité.

```
Test-Command: set -e
 ; cp -r tests $AUTOPKGTEST_TMP
 ; for py in $(py3versions -r 2>/dev/null)
 ; do cd "$AUTOPKGTEST_TMP"
 ; echo "Testing with $py:"
 ; $py -m pytest -v tests
 ; done
Depends:
 @,
 python3-all,
 python3-dtcwt,
 python3-pytest,
 python3-scipy,
 pocl-opencl-icd,
 clinfo,
 hipcc,
 python3-pytest-mock,
 python3-pytest-xvfb,
 xauth,
 xvfb,
Features: test-name=python-dtcwt-pocl
Restrictions: allow-stderr
```

Fichier *control* implémenté

```
Test-Command: rocm-test-launcher
 ; set -e
 ; cp -r tests $AUTOPKGTEST_TMP
 ; for py in $(py3versions -r 2>/dev/null)
 ; do cd "$AUTOPKGTEST_TMP"
 ; echo "Testing with $py:"
 ; $py -m pytest -v tests
 ; done
Depends:
 @,
 python3-all,
 python3-dtcwt,
 python3-pytest,
 python3-scipy,
 rocm-opencl-icd,
 pkg-rocm-tools,
 clinfo,
 hipcc,
 python3-pytest-mock,
 python3-pytest-xvfb,
 xauth,
 xvfb,
 libclang-common-17-dev,
 rocm-device-libs-17,
Features: test-name=python-dtcwt-rocm
Restrictions: allow-stderr
```

Fichier *control* implémenté

J'ai implémenté les autopkgtest pour ROCm et POCL mais il reste encore des erreurs dans les tests que je ne parviens pas à résoudre. En conclusion, les autopkgtest ont été implémentés mais il reste encore de nombreuses erreurs dans le build lui-même que je ne sais malheureusement pas résoudre au vu de ma faible expérience. Le paquet ne peut donc pas se construire correctement.

Xpra :

La procédure de lancement sans aucune modification est identique aux paquets précédents. Les autopkgtest ne se lancent pas en raison d'erreur dans le build. Dont voici les logs :

```
I: pybuild base:311: python3.13 setup.py clean
no distutils: No module named 'distutils', trying setuptools
Traceback (most recent call last):
  File "/build/reproducible-path/xpra-5.0+dfsg/setup.py", line 22, in <module>
    from distutils.core import setup
ModuleNotFoundError: No module named 'distutils'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/build/reproducible-path/xpra-5.0+dfsg/setup.py", line 27, in <module>
    from setuptools import setup
ModuleNotFoundError: No module named 'setuptools'
E: pybuild pybuild:389: clean: plugin distutils failed with: exit code=1: python3.13 setup.py clean
dh_auto_clean: error: pybuild --clean -i python{version} -p 3.13 returned exit code 13
make: *** [debian/rules:21: clean] Error 25
dpkg-buildpackage: error: debian/rules clean subprocess returned exit status 2
```

J'ai également implémenté le fichier *debian/tests/control* et ajouté le code nécessaire à l'exécution des tests dans le fichier *debian/tests/server* .

```
Test-Command: debian/tests/server
Depends:
 @,
 coreutils ,
 procps ,
 x11-apps ,
 x11-xserver-utils ,
 xauth ,
 clinfo ,
 pocl-opencl-icd,
 x11-common,
 @builddeps,
 setuptools,
 python-distutils-extra,
 stsci.distutils,
Features: test-name=xpra-pocl
Architecture: amd64, arm64, armel, armhf, i386
Restrictions: allow-stderr, skip-not-installable

Test-Command: rocm-test-launcher debian/tests/server
Depends:
 @,
 coreutils ,
 procps ,
 x11-apps ,
 x11-xserver-utils ,
 xauth ,
 clinfo ,
 pkg-rocm-tools,
 rocm-opencl-icd,
 x11-common,
 @builddeps,
 setuptools,
 python-distutils-extra,
 stsci.distutils,
 hipcc,
 xvfb,
 libclang-common-17-dev,
Features: test-name=xpra-rocm
Architecture: amd64, arm64, armel, armhf, i386
Restrictions: allow-stderr, skip-not-installable, skippable
```

Fichier *control* implémenté

```
---------------------------- debian/tests/server ------------
index 12d7c29..76a02e0 100755
@@ -59 +59,9 @@ xpra stop :314 || true
 xpra version :314 2>>/dev/null || printf ":: [[ All good ]]\n"
+
+cp -a test "$AUTOPKGTEST_TMP"
+
+for py in $(py3versions -r 2>/dev/null)
+do cd "$AUTOPKGTEST_TMP"
+   echo "Testing with $py:"
+   $py -m pytest --verbosity=2 2>&1
+done
```

Pour tenter de résoudre le problème j'ai installé : *python3-setuptools-rust* et *python3-distutils-extra*

J'ai implémenté les autopkgtest pour ROCm et POCL mais il reste encore des erreurs dans les tests que je ne parviens pas à résoudre. En conclusion, les autopkgtest ont été implémentés mais il reste encore de nombreuses erreurs dans le build lui-même que je ne sais malheureusement pas résoudre au vu de ma faible expérience. Le paquet ne peut donc pas se construire correctement.

Pysph :

La procédure de lancement sans aucune modification est identique aux paquets précédents. Le paquet fait de l'opencl mais n'a pas été recompilé sur python3. J'ai implémenté les autopkgtest pour ROCm et POCL mais il reste encore des erreurs dans les tests que je ne parviens pas à résoudre.

```
Test-Command: debian/tests/pysph-test-elliptical
Depends: python3-pysph, python3-all, cython3, clinfo, pocl-opencl-icd
Features: test-name=pysph-pocl
Restrictions: allow-stderr, skip-not-installable

Test-Command: rocm-test-launcher debian/tests/pysph-test-elliptical
Depends: python3-pysph, python3-all, cython3, clinfo, rocm-opencl-icd, pkg-rocm-tools, hipcc, libclang-common-17-dev
Features: test-name=pysph-rocm
Restrictions: allow-stderr, skip-not-installable,skippable
```

Fichier *control* implémenté

En conclusion, les autopkgtest ont été implémentés mais il reste encore de nombreuses erreurs dans le build lui-même que je ne sais malheureusement pas résoudre au vu du temps que cela prendra et de la connaissance du paquet lui-même. Le paquet ne peut donc pas se construire correctement.

Python-xrt :

```
Test-Command: set -efu
 ; cp -a tests "$AUTOPKGTEST_TMP"
 ; for py in $(py3versions -r 2>/dev/null)
 ; do cd "$AUTOPKGTEST_TMP"
 ; XRT_BACKEND=POCL $py -m pytest -k "not test_polygonal_aperture and not test_undulator_convergence" --verbosity=2 2>&1
 ; done
Depends:
 python3-all,
 ca-certificates,
 python3-xrt,
 python3-pytest,
 python3-opencv,
 @,
 python3-scipy,
 clinfo,
 pocl-opencl-icd,
 ocl-icd-opencl-dev
Features: test-name=python-xrt-pocl
Restrictions: allow-stderr, needs-internet

Test-Command: rocm-test-launcher
 ; set -efu
 ; cp -a tests "$AUTOPKGTEST_TMP"
 ; for py in $(py3versions -r 2>/dev/null)
 ; do cd "$AUTOPKGTEST_TMP"
 ; HIP_PLATFORM=amd XRT_BACKEND=ROCM $py -m pytest -k "not test_polygonal_aperture and not test_undulator_convergence" --verbosity=2 2>&1
 ; done
Depends:
 python3-all,
 ca-certificates,
 python3-pyopencl,
 python3-pytest,
 python3-opencv,
 @,
 python3-xrt,
 python3-scipy,
 clinfo,
 rocm-opencl-icd,
 pkg-rocm-tools,
 rocminfo,
 hipcc,
 rocm-device-libs-17,
 libclang-common-17-dev,
 ocl-icd-opencl-dev
Features: test-name=python-xrt-rocm
Restrictions: allow-stderr, needs-internet, breaks-testbed, skippable
```

Fichier *control* implémenté

La procédure de lancement sans aucune modification est identique aux paquets précédents. Les autopkgtest ont échoué en raison d'un bug connu au sujet de la dépendance *mayavi2* donc cela renvoie une erreur. J'ai créé, le répertoire */tests/* et créé le fichier *control*. J'ai fait un mail au mainteneur mais celui ne m'a pas répondu. J'ai implémenté les autopkgtest POCL et ROCm dans le fichier *debian/tests/control* .

```
*****python-xrt code d'erreur pour le bug report
Setting up python3-xrt (1.6.0+ds1-1.1) ...
  File "/usr/lib/python3/dist-packages/xrt/backends/raycing/__init__.py", line 341
    def global_to_virgin_local(bl, beam=None, lo, center=None, part=None):
                                                  ^^
SyntaxError: parameter without a default follows parameter with a default
dpkg: error processing package python3-xrt (--configure):
 installed python3-xrt package post-installation script subprocess returned error exit status 1
Processing triggers for ca-certificates (20250419) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
Errors were encountered while processing:
 python3-xrt
E: Sub-process /usr/bin/dpkg returned an error code (1)
E: Failed to process build dependencies
autopkgtest [10:52:30]:  - - - - - - - - - - running shell - - - - - - - - - -
bash: cannot set terminal process group (13157): Inappropriate ioctl for device
bash: no job control in this shell
```

Exemple d'erreurs

J'ai rencontré énormément d'erreurs dans les fichiers de tests dont voici les corrections, cela m'a pris plusieurs semaines à analyser et à corriger. Il s'agit des tests :

J'invite à regarder le patch *0007-Files-corrections.patch* pour avoir l'ensemble des modifications faites.

```
From: LONGEAC Clement <longeac@grades-01.synchrotron-soleil.fr>
Date: Wed, 13 Aug 2025 15:40:28 +0200
Subject: Files corrections

---
 tests/raycing/test_polygonal_aperture.py |   7 +-
 tests/raycing/test_sources.py            | 114 ++++++++++++++++++++++++++------
 xrt/backends/raycing/__init__.py         |   2 +-
 3 files changed, 99 insertions(+), 24 deletions(-)
```

Avec l'ensemble de ces modifications les autopkgtest se lancent parfaitement, seulement ils sont marqués comme étant timed out. Et je ne sais pas comment résoudre le problème.

En conclusion, les autopkgtest POCL et ROCm ont été implémentés, les tests se lancent sans accros mais sont marqués comme Timed Out et je ne sais malheureusement pas comment résoudre ce problème dans le temps qu'il me reste pour mon stage et au vu de la rédaction de ce même rapport. Il ne reste plus qu'à corriger cette petite erreur et le paquet pourra normalement être push sur le git salsa. Toutes les modifications ont été commit ainsi que les patchs. J'ai également fait un bugs report durant l'implémentation.

2.3 Les paquets ne faisant pas d'OpenCL

Examl :

La procédure de lancement sans aucune modification est identique aux paquets précédents. Les autopkgtest sans implémentation sont marqués comme PASS. J'ai implémenté les autopkgtest ROCm et POCL dont voici les résultats du *sbuild*.

```
Tests: run-unit-test
Depends: @, pocl-opencl-icd, clinfo, examl
Restrictions: allow-stderr

Test-Command: rocm-test-launcher  debian/tests/run-unit-test
Features: test-name=run-unit-test-rocm
Depends: @, clinfo, examl, rocm-opencl-icd,pkg-rocm-tools, libclang-common-17-dev ,mpi-default-dev ,mpi-default-bin
Restrictions: allow-stderr, skip-not-installable, skippable
```

Fichier *control* implémenté

```
autopkgtest [10:14:46]: test run-unit-test-rocm:  - - - - - - - - - - results
run-unit-test-rocm   PASS
autopkgtest [10:14:47]: @@@@@@@@@@@@@@@@@@@@ summary
run-unit-test        PASS
run-unit-test-rocm   PASS
```

J'ai envoyé un mail à l'auteur du paquet Alexandros Stamatakis, pour savoir si le paquet faisait bien de l'opencl puisqu'il faisait de l'openMP. Seulement, il explique qu'Examl ne fait pas d'opencl. Donc l'implémentation devient inutile. En conclusion, les autopkgtest POCL et ROCm ont été implémentés, mais le paquet ne fait pas d'openCl, l'implémentation devient alors inutile au regard de notre besoin.

Pocl :

Pocl n'est pas implémenté car il s'agit du paquet natif. On implémente les tests avec *pocl-opencl-icd*. Il se trouve qu'il est inutile de le faire puisque POCL ne pourra jamais faire de ROCm justement parce que c'est POCL.

Genx :

La procédure de lancement sans aucune modification est identique aux paquets précédents. Les autopkgtest sans implémentation sont marqués comme PASS. J'ai implémenté les autopkgtest ROCm et POCL dont voici les résultats du *sbuild*. J'ai également créé le fichier *opencl*.

```
PAQUET genx :

autopkgtest [14:31:34]: test opencl-rocm: ----------------------]
autopkgtest [14:31:34]: test opencl-rocm:  - - - - - - - - - - results - - - -
opencl-rocm          PASS
autopkgtest [14:31:35]: @@@@@@@@@@@@@@@@@@@@ summary
command1             PASS
python               PASS
opencl-pocl          PASS
opencl-rocm          PASS
```

```
#!/bin/sh -e

set -efux

# Check that OpenCL isnt totally broken (note that it isn't totally working either)
# Uses device 0 platform 0, i.e. to use a real GPU manually install its opencl-icd before running this
# Mark the test has flaky, the important part is the CPU computation.

# display the OpenCL platformes

TMPDIR=$(mktemp -d)

clinfo

# POCL Configuration
echo "=== POCL Configuration ==="
export OCL_ICD_VENDORS=/etc/OpenCL/vendors/pocl.icd
export POCL_WORK_GROUP_METHOD=loop

for py in $(py3versions --supported 2>/dev/null); do
    echo "Testing with $py: "
    $py -m pytest -v  genx/tests -v
done

echo "SUCCESS: POCL is fully operational"
exit 0
```

Fichier *control* implémenté

Je vérifie que le paquet fait bien de l'opencl en utilisant les commandes suivantes :

```
grep -r "import pyopencl" .
grep -r "from pyopencl" .
grep -r "pyopencl"
grep -r "cl.*import"
grep -r "cl/" ./gpmx
grep -r "cl[[:space:]]*--" ./gpmx
grep -r "cl[[:space:]]"
grep -r "cl.*buffer"
grep -r "CL/v" .
```

J'ai envoyé un mail au mainteneur de paquet afin de savoir si *numba* fait de l'opencl. Il se trouve que cette dépendance ne fait pas d'Opencl. En conséquence le paquet ne fait pas d'OpenCl.

En conclusion, les autopkgtest POCL et ROCm ont été implémentés, mais le paquet ne fait pas d'openCl, l'implémentation devient alors inutile au regard de notre besoin.

Mission en plus

Frédéric-Emmanuel PICCA, m'avais donné comme mission de réaliser un code utilisant la commande *sbuild* et retournant le temps de construction des autopkgtest pour POCL et ROCm à la fin de la construction du paquet. Ce qui signifie que je devais lier ce code à la commande du *sbuild* afin qu'il ajoute cela à la fin. Il s'agit de la capture des temps et de les retourner dans le fichier de build , comme dans l'exemple. J'ai énormément cherché à ce sujet et il y a très peu de documentation là-dessus. Ce qui rend l'implémentation en pratique très compliquée et technique, pour moi qui n'ai pas une connaissance aussi poussée de l'environnement Débian que Frédéric-Emmanuel. J'ai donc mis de côté cette tâche pour me concentrer sur l'implémentation des paquets et des erreurs de code, qui me donne déjà beaucoup de fil à retordre.

exemple :

command1 200s
opecl-pocl 200s
opencl-rocm 600s

Conclusion

Ces quatre mois de stage au Synchrotron SOLEIL m'ont vraiment permis de plonger dans un environnement scientifique de haut niveau et de contribuer à un projet ayant un véritable impact pour la communauté. Ma mission consistait à développer et intégrer des tests automatisés (autopkgtests) dans l'environnement Debian, afin d'évaluer la compatibilité de différents paquets avec les GPU AMD via la plateforme ROCm. Cet objectif, inscrit dans une volonté de diversification des architectures matérielles et de réduction de la dépendance croissante à NVIDIA, m'a donné l'opportunité d'allier mes compétences en programmation, en administration Linux et en data science à un enjeu scientifique et technologique particulièrement d'actualité.

La réalisation de ces tests a exigé de ma part un travail très méthodique : j'ai d'abord étudié en profondeur les paquets concernés, puis j'ai cherché à comprendre leurs dépendances, mis en place des environnements isolés, configuré le GPU AMD, et rédigé des scripts adaptés. J'ai ainsi pu équiper plusieurs logiciels scientifiques (tels que *pyopencl*, *nabu*, *ufo-filters* ou encore *pyvkfft*) d'autopkgtests ROCm. Ces contributions, que j'ai eu la satisfaction de voir intégrées ensuite aux dépôts Debian, profitent non seulement au Synchrotron SOLEIL mais aussi à l'ensemble de la communauté scientifique internationale qui utilise ces paquets.

Cependant, je dois reconnaître que ce stage n'a pas été exempt de difficultés parfois décourageantes. Certains logiciels présentaient des contraintes techniques majeures, comme l'absence de code OpenCL ou la non-migration de Python 2 vers Python 3, ce qui a considérablement limité leur traitement. D'autres paquets ont révélé des problèmes intrinsèques, empêchant leur intégration complète malgré mes efforts. Par ailleurs, la configuration des environnements ROCm a parfois généré des incompatibilités complexes, sources de ralentissements et de tests échoués qui m'ont parfois frustré. Ces obstacles, bien qu'ils aient limité l'achèvement total de ma mission, se sont finalement révélés extrêmement formateurs pour moi. Ils m'ont appris à faire face à l'imprévu, à documenter mes démarches avec une rigueur que je n'avais pas forcément avant, et à collaborer activement avec mes encadrants pour trouver des solutions adaptées.

Au-delà de l'aspect purement technique, ce stage m'a énormément apporté sur le plan humain et professionnel. Travailler quotidiennement aux côtés de chercheurs et d'ingénieurs passionnés m'a donné une compréhension bien plus concrète du rôle essentiel de l'informatique scientifique dans la recherche expérimentale. J'ai également appris à mieux communiquer mes résultats, à confronter mes choix techniques à des enjeux plus larges (scientifiques, stratégiques, géopolitiques), et à vraiment situer mon travail dans une démarche collective. Ce stage a donc constitué une étape que je considère comme décisive dans la construction de mon identité d'ingénieur, à la croisée de la science fondamentale et de la data science.

En conclusion, cette expérience conforte pleinement mon projet professionnel : je veux poursuivre mes études dans la spécialité Data Science et Intelligence Artificielle à l'ESIEE Paris, les enrichir par un double diplôme avec le Politecnico di Milano, puis compléter mon parcours par un master en physique fondamentale. Mon ambition est désormais plus claire que jamais : mettre mes compétences en machine learning et en calcul haute performance au service de la physique des particules, idéalement au CERN, où l'analyse de données est au cœur de la recherche. Ce stage au Synchrotron SOLEIL n'était pas une fin en soi pour moi, mais bien une première pierre fondamentale. Il m'a offert un aperçu concret et passionnant de la façon dont mes passions pour l'informatique et la physique peuvent se conjuguer pour contribuer, à mon échelle, aux avancées scientifiques de demain.

Bibliographie

[1] https://fr.wikipedia.org/wiki/Intelligence_artificielle
[2] https://hardware.developpez.com/actu/372849/Le-fabricant-de-puces-NVIDIA-s-assure-92-pourcent-du-marche-des-cartes-graphiques-au-premier-trimestre-2025-laissant-AMD-et-Intel-a-la-traine/#:~:text=NVIDIA%20laisse%20peu%20de%20place%20%C3%A0%20la%20concurrence&text=AMD%20ne%20d%C3%A9tient%20que%208,pas%20eu%20d'impact%20statistique.
[3] https://www.synchrotron-soleil.fr/fr/lignes-de-lumiere
[4] https://home-cern.translate.goog/science/accelerators?_x_tr_sl=en&_x_tr_tl=fr&_x_tr_hl=fr&_x_tr_pto=rq#:~:text=An%20accelerator%20propels%20charged%20particles,world%20of%20the%20infinitely%20small.
[5] https://fr.wikipedia.org/wiki/Rayonnement_synchrotron
[6] https://fr.wikipedia.org/wiki/European_Synchrotron_Radiation_Facility
[7] https://visa.readthedocs.io/en/latest/overview/overview-contents.html
[8] soleil-packaging-overview
[9] MyWiki_OpenCL
[10] fr.wikipedia.org/wiki/CLOUD_Act
[11] https://www.riskinsight-wavestone.com/2021/11/le-cloud-act-une-maniere-de-rendre-les-donnees-non-souveraines/
[12] https://www.kiteworks.com/fr/glossaire/us-cloud-act/
[13] https://www.usine-digitale.fr/article/16-04-2024-microsoft-epingle-par-le-gouvernement-americain-pour-sa-culture-de-securite-lacunaire.N2211541
[14] https://www.oodrive.com/fr/blog/reglementation/conformite/cloud-act-les-etats-unis-donnent-un-cadre-legal-a-la-saisie-de-donnees-hebergees-a-letranger/
[15] https://siecledigital.fr/2021/07/05/etats-unis-donnees-microsoft/
[16] https://www.netexplorer.fr/blog/cloud-act-interet-de-stocker-donnees-en-france
[17] https://noyb.eu/fr/us-cloud-soon-illegal-trump-punches-first-hole-eu-us-data-deal
[18] https://www.ib-formation.fr/home/mag/infrastructure/systemes-d-exploitation/systemes-dexploitation-tendances-et-innovation#:~:text=Les%20syst%C3%A8mes%20d'exploitation%20pour%20ordinateur,-Dressons%20une%20liste&text=Windows%20%3A%2072%20%25,Chrome%20OS%20%3A%202%2C5%25
[19] https://www.debian.org/intro/about.fr.html
[20] https://fr.wikipedia.org/wiki/Debian
[21] https://www.debian.org/releases/index.fr.html
[21] https://www.debian.org/devel/testing.fr.html
[22] https://www.debian.org/doc/manuals/debian-handbook/sect.release-lifecycle.fr.html
[23] https://www.debian.org/News/2025/20250517
[24] https://wiki.debian.org/LTS
[25] https://debian-facile.org/doc:systeme:apt:branches-debian
[26] https://www.debian.org/doc/manuals/debian-handbook/sect.package-maintenance.fr.html
[27] https://wiki.debian.org/DebianScience
[28] https://wiki.debian.org/DebianScience/ContributingToDebianScience
[29] https://lists.debian.org/debian-science/
[30] https://tracker.debian.org
[31] https://packages.qa.debian.org/common/index.html
[32] https://qa.pages.debian.net/distro-tracker/
[33] https://www.debian.org/Bugs/
[34] https://wiki.debian.org/Salsa
[35] https://manpages.debian.org/testing/devscripts/salsa.1.en.html
[36] https://www.reddit.com/r/debian/comments/17y3goq/why_does_debian_calls_its_gitlab_salsa/
[37] https://wiki.debian.org/SalsaCI
[38] https://optimizedbyotto.com/post/debian-salsa-ci-gsoc-2025/
[39] https://debconf25.debconf.org/talks/137-salsa-ci-now-running-for-27-000-packages-what-have-we-learnt/
[40] https://wiki.debian.org/sbuild
[41] https://wiki.debian.org/PackagingTools
[42] https://wiki.debian.org/ContinuousIntegration/autopkgtest
[43] https://wiki.debian.org/DebianPAN/OpenCL
[44] https://wiki.debian.org/Projects/DebSrc3.0
[45] https://wiki.debian.org/debian/patches
[46] https://www.debian.org/doc/manuals/maint-guide/modify.en.html
[47] https://perl-team.pages.debian.net/howto/quilt.html

[48] https://unix.stackexchange.com/questions/331183/debian-packaging-whats-the-difference-between-patches-and-changelog
[49] https://raphaelhertzog.com/2012/08/08/how-to-use-quilt-to-manage-patches-in-debian-packages/
[50] https://www.debian.org/Bugs/Reporting
[51] https://wiki.debian.org/BugReport
[52] https://wiki.debian.org/reportbug
[53] https://wiki.debian.org/HowtoUseBTS
[54] https://www.debian.org/doc/manuals/debian-handbook/sect.common-procedures.en.html
[55] https://en.wikipedia.org/wiki/ROCm
[56] https://rocm.docs.amd.com/projects/HIPIFY/en/latest/index.html
[57] https://rocm.docs.amd.com/en/develop/reference/rocm-tools.html
[58] https://rocm.docs.amd.com/projects/rocminfo/en/latest/
[59] https://rocm.docs.amd.com/en/docs-6.3.1/reference/rocm-tools.html
[60] https://rocm.docs.amd.com/projects/install-on-linux/en/docs-6.3.0/reference/system-requirements.html
[61] https://rocm.docs.amd.com/en/develop/reference/gpu-arch-specs.html
[62] https://rocm.docs.amd.com/en/docs-6.1.5/compatibility/compatibility-matrix.html
[63] https://en.wikipedia.org/wiki/SYCL
[64] https://github.com/ROCm/ROCm/issues/4659
[65] https://en.wikipedia.org/wiki/CUDA
[66] https://developer.nvidia.com/cuda-zone
[67] https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
[68] https://docs.nvidia.com/cuda/parallel-thread-execution/index.html
[69] https://developer.nvidia.com/gpu-accelerated-libraries
[70] https://developer.nvidia.com/nsight-systems
[71] https://developer.nvidia.com/cuda-gpus
[72] https://developer.nvidia.com/tensor-cores
[73] https://pytorch.org/get-started/locally/
[75] https://www.nextplatform.com/2020/06/23/the-long-term-costs-of-vendor-lock-in/
[76] https://www.techspot.com/article/2520-gpu-pricing/
[77] https://hpc-wiki.info/hpc/CUDA
[78] https://fr.wikipedia.org/wiki/OpenCL