Mémoire de stage

Evan Potin

Université de Bordeaux

Mémoire de Stage

POTIN Evan

Contents

List of Figures

1 Remerciment

Ce rapport de stage fait partie de mon programme de Master en Calcul Intensif et Sciences des Données à l'Université de Bordeaux.

Je tiens à exprimer ma gratitude à Monsieur Olivier Aumage pour m'avoir donné l'opportunité de réaliser ce stage, ainsi que pour son accompagnement et tout le soutien technique qu'il a su m'apporter tout au long de cette expérience.

Je suis également très reconnaissante envers Messieurs Antsa Rasamoela et Valentin Hazard pour le suivi attentif et la disponibilité dont ils ont fait preuve tout au long du stage.

Enfin, je souhaite remercier toute l'équipe STORM pour leur accueil chaleureux et leur convivialité.

Merci pour votre lecture.

2 Introduction

Ce stage a été réalisé pour répondre aux besoins du projet SKAO. Ce dernier a pour objectif d'implanter deux champs d'antennes, l'un en Afrique du Sud et l'autre en Australie. Ces champs d'antennes font partie de l'observatoire SKA (SKAO) et serviront à collecter en continu des données astronomiques.

Ces antennes vont générer une quantité massive de données, environ 10 To/s. Le SKAO a décidé d'effectuer un traitement préalable de ces données avant leur stockage. Une chaîne de traitement a déjà été mise en place, sous le nom de DDFacet. Cependant, une contrainte majeure de ce traitement est que pour chaque heure de données récupérées, le traitement doit également être effectué en une heure. Avec la parallélisation actuelle présente dans DDFacet, ce ratio n'est pas respecté.

L'équipe STORM propose donc d'essayer d'utiliser Stream-PU (anciennement AFF3CT-core). Ce dernier est conçu pour accomplir les tâches suivantes :

- **Définition des composants de flux de données : modules, tâches et sockets**
  Stream-PU permet de définir de manière précise les composants essentiels du flux de données, y compris les modules, les tâches et les sockets. Ces éléments sont fondamentaux pour structurer et gérer efficacement le traitement des données.
- **Implémentation des modules et tâches élémentaires**
  Stream-PU offre des fonctionnalités pour implémenter des modules et des tâches élémentaires. Ces implémentations sont cruciales pour exécuter les opérations de base sur les données, facilitant ainsi la création de systèmes complexes et performants.
- **Environnement d'exécution multi-thread avec des constructions de réplication et de pipeline parallèle**
  Stream-PU intègre un environnement d'exécution multi-threadé qui prend en charge la réplication et les constructions de pipeline parallèle. Cela permet une exécution simultanée et optimisée des tâches, améliorant ainsi les performances et l'efficacité du traitement des données.

Je vais donc vous présenter cela dans ce rapport, en commençant par un contexte général, suivi par la proposition de solution que nous avons avancée, puis en détaillant ce qui a été implémenté au cours de ce stage. Enfin, je conclurai par une réflexion sur le stage.

3 Contexte

3.1 Observatoire SKA

L'Observatoire SKAO est une organisation internationale regroupant des pays du monde entier. Sa mission est de construire et d'exploiter des radiotélescopes ultra-modernes pour transformer notre compréhension de l'Univers et apporter des bénéfices à la société grâce à la collaboration et à l'innovation mondiale.

Le SKAO est une infrastructure de nouvelle génération dédiée à l'astronomie radio et au traitement de données massives, qui promet de révolutionner notre compréhension de l'Univers et des lois de la physique. Grâce à une technologie de pointe, il est prévu qu'il ait un impact significatif, non seulement dans le domaine des sciences, mais aussi au-delà.

L'observatoire a une portée mondiale avec son siège global situé au Royaume-Uni, ses deux télescopes implantés dans des zones très calmes en Afrique du Sud et en Australie, ainsi que des installations de support pour les opérations des télescopes.

Lorsqu'il sera pleinement opérationnel, le SKAO sera un observatoire mondial unique avec deux télescopes répartis sur trois continents, au service de ses États membres et partenaires.

3.2 Les Télescopes du SKA

Avec des centaines de paraboles et des milliers d'antennes, les télescopes du SKAO seront les radiotélescopes les plus avancés au monde.

En collaboration avec d'autres infrastructures de recherche de pointe, les télescopes du SKAO exploreront les frontières inexplorées de la science et approfondiront notre compréhension de processus clés, tels que la formation et l'évolution des galaxies, la physique fondamentale dans des environnements extrêmes, et les origines de la vie.

3.3 Les différents sites

Le SKAO est une organisation distribuée, un observatoire mondial utilisant deux télescopes répartis sur trois continents. Chaque télescope dispose d'un Centre d'Opérations Scientifiques (SOC) et d'un Centre d'Opérations Techniques (EOC) dans son pays hôte. Nos trois pays hôtes sont :

- Le Royaume-Uni, où se trouve le Siège Global, sur le site de Jodrell Bank dans le Cheshire, à côté de l'Observatoire de Jodrell Bank de l'Université de Manchester et du Centre d'Engagement, un site du patrimoine mondial de l'UNESCO.
- L'Australie, où se trouve le télescope SKA-Low. Le SOC du SKA-Low est à Perth, en Australie-Occidentale (WA), et l'EOC du SKA-Low est à Geraldton, WA.
- L'Afrique du Sud, où se trouve le télescope SKA-Mid. Le SOC du SKA-Mid est au Cap, et l'EOC du SKA-Mid est à Klerefontein, près de Carnarvon dans la province du Cap Nord.

3.4 Le projet

C'est dans ce contexte que s'inscrit le projet sur lequel nous allons travailler. Comme mentionné dans l'introduction, ces antennes produisent une grande quantité de données (10 To/s) qui devront être traitées par une chaîne de traitement avant d'être stockées. Cette chaîne de traitement a été développée et porte le nom de DDFacet. Vous pouvez la voir ci-dessous.

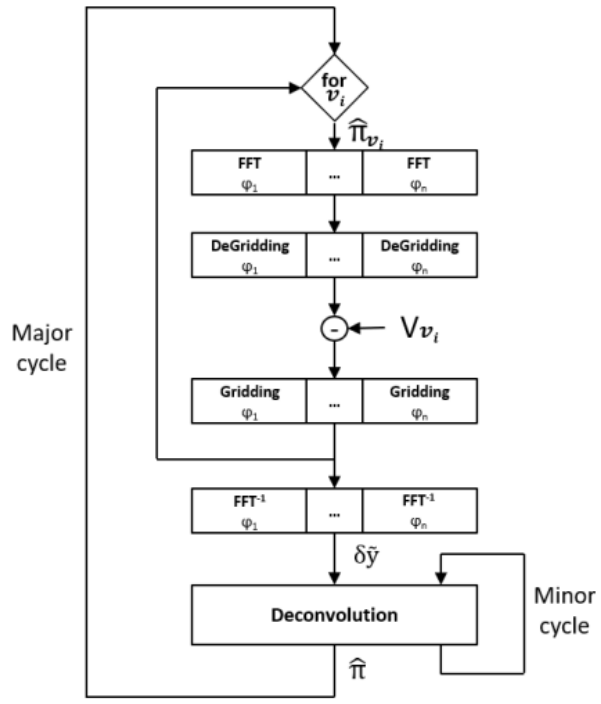

Figure 1: Schéma représentant la chaine d'exécution de DDFacet, issue de l'article *Parallelisation of the Wide-Band Wide-Field Spectral Deconvolution Framework DDFacet on Distributed Memory HPC System*

Je ne suis pas un expert en la matière, car ces opérations sont assez complexes sur le plan mathématique, mais voici ce que j'ai compris de chaque étape :

- **FFT :** Il s'agit d'une méthode permettant de calculer la transformée de Fourier, utilisée pour passer du domaine temporel au domaine fréquentiel.
- **Degridding :** Cette opération est l'inverse du gridding. Elle consiste à convertir des données échantillonnées régulièrement dans le plan uv (domaine fréquentiel) en données échantillonnées de manière irrégulière (visibilités mesurées).
- **Gridding :** Cette étape consiste à utiliser les visibilités (données irrégulièrement échantillonnées) pour estimer leurs valeurs sur une grille régulière dans le plan uv.
- **FFT-1 :** C'est l'opération inverse de la FFT, qui reconvertit un signal du domaine fréquentiel au domaine temporel ou spatial.
- **Déconvolution :** Cette technique est utilisée pour corriger les flous dans l'image observée, causés par les imperfections du télescope.

L'objectif du projet est donc d'optimiser cette chaîne de traitement. L'idée principale est de l'intégrer dans Stream-PU. Dans les sections suivantes, je détaillerai les différentes réalisations effectuées au cours de ce stage.

4 Proposition

Nous avons déjà commencé à introduire la proposition que nous allons implémenter. Nous allons donc essayer de paralléliser la chaîne que nous avons présentée ci-dessus. Pour ce faire, il faudra d'abord retirer la parallélisation déjà présente dans DDFacet. Bien que cette parallélisation soit opérationnelle, elle n'est pas très optimisée. En effet, les threads ne sont pas utilisés à leur plein potentiel et le nombre maximal de threads est rarement exploité lors de l'exécution. Nous espérons donc améliorer ces aspects.

Une autre proposition mentionnée était l'utilisation de l'algorithme OTAC, développé par Diane Orhan avec l'aide de l'équipe STORM. L'idée était d'utiliser OTAC pour simuler le parallélisme avec DDFacet afin d'évaluer l'efficacité de notre implémentation. Cette idée n'est apparue qu'à la fin du stage, lorsque nous avons réalisé que le temps disponible ne permettait pas de compléter l'implémentation initiale.

5 Éléments d'implémentation & déroulement du Stage

5.1 Prise en main des logiciels

Je vais donc présenter point par point le dérloulement du stage. La premiere etape a été de me familiarisé avec les logiciel. Donc de prendre en main DDfacet et StreamPU. J'ai donc d'abord commencer par utiliser StreamPU et non pas PyStreamPU car avant le premier rendez vous que l'on a fait avec Antsa Rasamoela et Valentin Hazard qui sont les interlocuteur que l'on aura tout au long du stage. C'est eux qui faisait le pont entre le SKA et L'inria. Donc comme je le disais, j'ai commencer par prendre StreamPU en main, je n'ai pas eu de probleme particulier avec cela. J'ai reproduit la séreie d'exemple qui etait proposer dans code source. Ensuite, la prise en main avec DDFacet. On a un peu plus peiner. DDFacet est livrée avec des images docker. Ces derniere sont la pour pas que l'on est besoin d'installer tout les packages néssésaire a DDFacet, qui sont nombreux, sur notre machine. On a eu du mal a mettre en place ces machines docker sur Plafrim car pour utiliser ces machines, il nous faut les droit administrateur. On a donc utilisé une autre image, une singularity cette fois ci. Avec cette derniere, tout fonctionnais. On pouvais lancer DDFacet sur Plafrim.

Un fois cela réalisé, avec Olivier Aumage, nous avons pris la decision de mettre en place un enviroment guix dans lequelle nous pourrions utiliser DDFacet. Les images ne sont pas trés propices a la paralélisation et ce serait beacoup plus pratique et portable pour tout le monde si on pouvais mettre en place cette environement.

5.2 Prise en main des logiciels

Je vais maintenant présenter le déroulement du stage étape par étapes. La première étape a consisté à me familiariser avec les logiciels, en commençant par DDFacet et StreamPU. J'ai d'abord commencé par utiliser StreamPU plutôt que PyStreamPU. Avant notre premier rendez-vous avec Antsa Rasamoela et Valentin Hazard, nos principaux interlocuteurs qui ont fait le lien entre le SKA et Inria, nous n'avions pas encore l'information que le logiciel était en Python.

Comme je l'ai mentionné, j'ai d'abord pris en main StreamPU, et je n'ai pas rencontré de problèmes particuliers. J'ai reproduit la série d'exemples proposée dans le code source. Ensuite, j'ai commencé à me familiariser avec DDFacet, ce qui a été un peu plus complexe. DDFacet est fourni avec des images Docker qui permettent d'éviter d'installer directement tous les packages nécessaires (qui sont nombreux) sur notre machine. Cependant, nous avons rencontré des difficultés pour configurer ces images Docker sur Plafrim, car cela nécessitait des droits administrateur. Nous avons donc opté pour une autre solution, en utilisant une image Singularity à la place. Avec celle-ci, tout fonctionnait correctement, et nous avons pu lancer DDFacet sur Plafrim.

Une fois cette étape terminée, Olivier Aumage et moi avons décidé de mettre en place un environnement Guix pour utiliser DDFacet. Les images ne sont pas très adaptées à la parallélisation, et il serait beaucoup plus pratique et portable pour tout le monde de disposer de cet environnement.

5.3 Mise en place de l'environnement Guix

Notre première tâche a été de créer un fichier `.scm` qui, une fois exécuté sur une machine avec Guix installé, télécharge et installe toutes les bibliothèques nécessaires dans un environnement virtuel où nous pourrons utiliser DDFacet. Cette tâche m'a pris plus de temps que prévu. Tout d'abord, Guix n'est pas très facile à prendre en main. C'est un outil très pratique, mais sa

documentation est difficile à utiliser. Elle est principalement destinée à ceux qui connaissent déjà Guix et savent ce qu'ils veulent ou doivent faire. Pour quelqu'un qui ne le connaît pas du tout, elle est, à mon avis, presque inutilisable, car on s'y perd facilement. J'ai donc choisi d'examiner le code source de Guix et de parcourir les différents fichiers `.scm` qui s'y trouvent pour essayer de comprendre comment cela fonctionnait.

Le second problème majeur que j'ai rencontré concernait l'installation des différentes bibliothèques. Il y en avait beaucoup, et elles n'étaient pas toutes simples à installer. Pour cela, j'ai d'abord essayé d'installer chacune de ces bibliothèques indépendamment sur ma machine. L'idée était de créer un script bash pour chaque bibliothèque, afin de pouvoir reproduire le contenu de ces scripts dans Guix. Nous avons donc installé les bibliothèques suivantes : `sofa-c`, `casacore-3.5.0-ddfacet`, `casarest`, `python-casacore`, `makems`, `pyxis`, `lofarbeam`, et `meqtrees-timba`. Ce sont celles pour lesquelles nous avons dû apporter le plus de modifications lors de l'installation avec Guix.

Je ne vais pas détailler ici tous les problèmes que j'ai rencontrés avec chaque bibliothèque, car ce n'est pas forcément intéressant. Cependant, je vais vous expliquer à titre d'exemple la phase d'installation de Guix pour `makems` et la détailler.

```
(define-public makems
  (package
    (name "makems")
    (version "1.5.5")
    (source
      (origin
      (method git-fetch)
       (uri (git-reference
             (url "https://github.com/ratt-ru/makems.git")
             (commit (string-append "v" version))))
       (file-name (git-file-name name version))
       (sha256
         (base32 "0lkm0cqw47nkl9n9qsqc51x32hsyxsr6sxg28ppf0mlbhrcv73aa"))))
    (propagated-inputs
     (list
       boost
       casacore-3.5.0-ddfacet
       casarest
       gcc-toolchain-11
       gfortran
       python-casacore
       cfitsio
       lapack
       openblas
       wcslib
       ))
    (build-system cmake-build-system)
    (arguments
     `(#:tests? #f                      ;no test
       #:phases
       (modify-phases %standard-phases
         (add-after 'unpack 'fix-install-dir
            (lambda* (#:key inputs #:allow-other-keys)
             (let ((gcc (search-input-file inputs "/bin/gcc"))
                   (gxx (search-input-file inputs "/bin/g++")))
```

```
            (substitute* "LOFAR/CMake/variants/variants"
                         (("/usr/bin/gcc") gcc)
                         (("/usr/bin/g\\+\\+") gxx))
            (substitute* "LOFAR/LCS/Common/src/shmem/dlmalloc.c"
                         (("#include \"/usr/include/malloc.h\"") "#include <malloc.h>")))
            #t))
        (replace 'configure
          (lambda* (#:key outputs #:allow-other-keys)
            (chdir "LOFAR")
            (invoke "pwd")
            (let ((lofar-dir (getcwd)))
              (mkdir-p "build/gnu_opt")
              (chdir "build/gnu_opt")
              (invoke
               "cmake"
               "-G" "Unix Makefiles"
               "-DCMAKE_IGNORE_PATH=/usr;/usr/bin;/usr/lib;/usr/lib64;/usr/include"
               "-DCMAKE_CXX_STANDARD=11"
               (string-append "-DCMAKE_MODULE_PATH:PATH=" lofar-dir "/CMake")
               "-DUSE_LOG4CPLUS=OFF"
               "-DBUILD_TESTING=OFF"
               "-DCMAKE_BUILD_TYPE=Release"
               (string-append "-DCMAKE_INSTALL_PREFIX=" (assoc-ref outputs "out"))
               "../.."))))))
    (synopsis "")
    (description "")
    (home-page "https://github.com/ratt-ru/makems")
    (license license:gpl2+)))
```

Je vais donc décrire cet exemple pour illustrer ce que j'ai fait et les problèmes que l'on peut rencontrer. Pour chacune des bibliothèques, la première étape consiste à télécharger le code source (pour celles qui sont sur GitHub, sinon il fallait télécharger le code source et créer un lien vers celui-ci dans le fichier `.scm`). Ensuite, on prépare l'installation en indiquant les bibliothèques nécessaires. Comme on peut le voir ici, pour `makems`, nous avons besoin de `casacore-3.5.0-ddfacet`, `casarest`, et `python-casacore`, qui sont également des bibliothèques que nous avons dû installer dans le fichier `.scm`. Cela a également causé quelques problèmes.

Puis vient la phase d'installation. Dans Guix, cette phase se déroule en étapes, définies par le `build-system`. Une fois que l'on a choisi la suite d'étapes à utiliser, il faut les adapter au cas sur lequel on travaille. On peut donc créer, supprimer, ou modifier des phases. Dans notre exemple, nous avons ajouté une phase appelée `fix-install-dir`. Dans cette phase, nous modifions une ligne du fichier **LOFAR/CMake/variants/variants** car celui-ci contient un lien brut vers le chemin par défaut du compilateur C++. Dans notre cas, nous devons utiliser le compilateur C++ installé dans l'environnement virtuel Guix. Ce problème a été particulièrement difficile à détecter, car nous ne comprenions pas pourquoi l'exécution ne prenait pas en compte le lien vers le compilateur que nous spécifiions dans la commande d'exécution (nous l'avions inclus dans les chemins définis dans la phase `replace 'configure`). Enfin, dans cette phase `replace 'configure`, nous configurons les différents chemins nécessaires.

La mise en place de cet environnement Guix a pris beaucoup plus de temps que prévu. Finalement, nous avons terminé cette partie trois mois après le début du stage.

5.4 Prise en main de PyStreamPU

Une fois cette partie terminée, il était temps de commencer à mettre en place notre fameuse chaîne d'exécution parallélisée. Mais avant cela, j'ai dû me familiariser avec PyStreamPU (qui est la version Python de StreamPU). Cette version est encore en cours de développement, réalisée par Romain Tajan. J'ai rencontré quelques difficultés pour prendre en main ce logiciel, car, comme je l'ai mentionné, il est encore en développement et n'a pas encore de documentation rédigée. J'ai donc dû découvrir son fonctionnement par moi-même, avec l'aide de Romain Tajan. Comme pour la prise en main de StreamPU, j'ai refait les codes d'exemples présents dans le code source. Cependant, j'ai eu plus de mal à m'y habituer. Pour la suite de ce rapport, il est nécessaire d'expliquer plus précisément les bases de StreamPU.

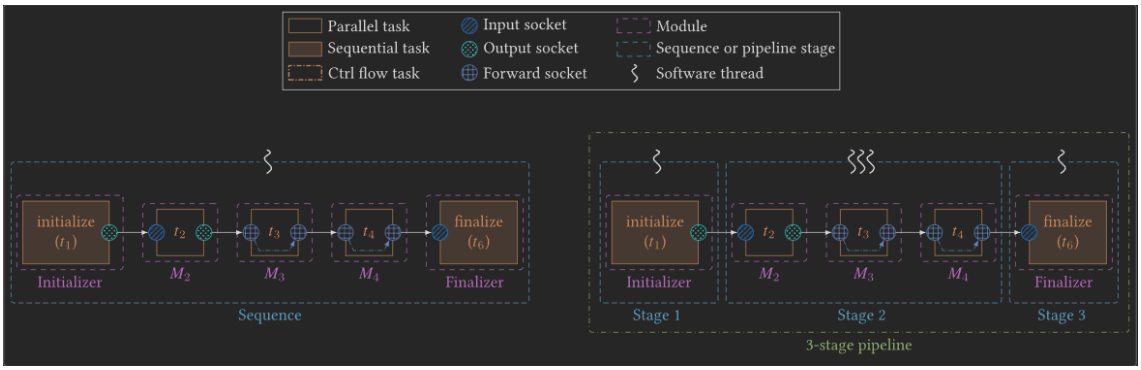

Figure 2: Exemple de chaîne d'exécution StreamPU copié de la documentation officielle

Sur cette image, nous voyons un exemple simple de chaîne d'exécution que l'on peut créer dans StreamPU. En général, chaque chaîne commence par une phase d'initialisation (**initialize**) et se termine par une phase de finalisation (**finalize**). Ces phases permettent au logiciel de comprendre clairement où commence et où se termine le flux d'exécution.

Entre ces deux phases, nous intégrons les différentes tâches que nous souhaitons exécuter. StreamPU propose un grand nombre de modèles de tâches prédéfinis, dont je vous en présenterai quelques-uns par la suite. Par exemple, on remarque que les tâches contenues dans le "Stage 2" peuvent être exécutées en parallèle. C'est l'une des fonctionnalités centrales de StreamPU : le logiciel est capable de paralléliser différentes tâches, voire des étapes entières de notre chaîne, si nous le lui demandons.

Nous avons également la possibilité de personnaliser cette parallélisation selon nos besoins, en spécifiant par exemple le nombre de processus à utiliser pour une tâche donnée.

Passons maintenant aux différents modules déjà intégrés dans StreamPU. Certains modules sont conçus pour effectuer des calculs spécifiques, tels que l'incrémentation (**incrementer**) ou l'itération (**iterator**). D'autres modules, quant à eux, sont dédiés à la gestion du comportement de la chaîne de traitement, permettant de gérer les interactions entre les différentes étapes.

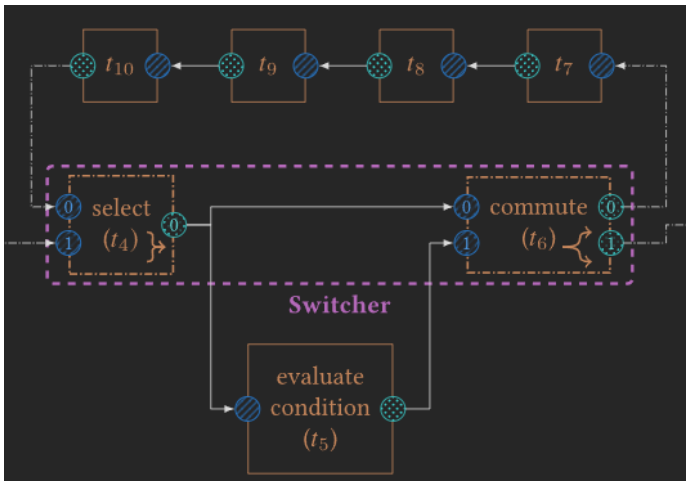

Figure 3: Exemple d'une boucle dans une chaîne d'exécution tiré de la documentation officielle

Ici, on voit comment, avec l'aide de StreamPU, il est possible de réaliser une boucle. Le module `select` envoie les données à évaluer à la tâche `T5`. Cette dernière transmet les résultats au module `commute`, qui décide soit de relancer la boucle, soit de la quitter en fonction du résultat.

Nous avons ici les principales fonctionnalités de StreamPU.

5.5 Première intuition d'implémentation

DDFacet propose cinq modes d'exécution : Predict, Subtract, Clean, Dirty, PSF, et Restore-AndShift. Chaque mode utilise une chaîne d'exécution différente. L'idée initiale est de créer une chaîne d'exécution pour chaque mode, avec une tâche préalable qui sélectionne la chaîne à lancer en fonction du mode souhaité.

C'est dans cette direction que nous avons décidé de progresser. Le principal problème rencontré était de déterminer par où commencer. Il était nécessaire de faire une trace de chaque chaîne de traitement dans un premier temps, ce qui n'était pas évident en raison des nombreuses fonctions imbriquées appelées. J'ai donc commencé par me concentrer sur le mode Predict. Une fois la trace du mode Predict réussie, l'étape suivante était de découper cette chaîne d'exécution de manière efficace pour optimiser la parallélisation. J'ai rapidement compris que trouver une découpe optimale sans voir l'exécution réelle serait compliqué. J'ai donc opté pour une implémentation plus naïve dans un premier temps.

5.6 Implémentation naïve

Cette implémentation consiste à ne pas chercher à optimiser le découpage pour le moment, mais simplement à diviser la trace du mode Predict en deux parties et à intégrer cela dans PyStreamPU. Comme mentionné précédemment, j'ai pris en main PyStreamPU, mais j'avais surtout utilisé les modules existants sans en créer de nouveaux moi-même. C'est donc ce que je devais faire ici.

Les chaînes de traitement dans DDFacet appellent de nombreuses fonctions, toutes appartenant à des classes créées dans DDFacet. Pour créer le premier module du mode Predict, que j'avais prévu de diviser en deux modules pour l'instant, je devais préparer une variable contenant toutes les informations nécessaires pour le second module, afin de les transmettre via un socket. Comme chaque module correspond à une tâche (voir les exemples précédents), les tâches ne peuvent pas partager directement des données. Il fallait donc passer toutes les informations

nécessaires via un socket. Cependant, les données nécessaires n'étant pas présentes dans la fonction principale de mon module posaient problème ; je ne pouvais pas créer un grand tableau de données à envoyer ultérieurement.

J'ai donc décidé de tenter de regrouper le contenu des fonctions appelées dans une grande fonction dans mon module. Bien que ce ne soit pas une solution définitive, je voulais obtenir un point de départ, quelque chose que je pourrais exécuter et améliorer ensuite. Je me suis rapidement rendu compte que cette approche n'était pas viable, en raison des nombreuses dépendances entre les différentes fonctions.

J'ai donc essayé de réaliser un meilleur découpage. Bien qu'il ne soit pas encore optimal pour une parallélisation parfaite, il visait à réduire le transfert de données via les sockets. En effet, cette implémentation naïve avait révélé que le problème de performance était dû à un découpage inapproprié, entraînant un transfert excessif de données.

5.7 Problème du `ClassImagerDeconv`

Dans DDFacet, la classe `ClassImagerDeconv`, que nous appellerons simplement *Imager* pour des raisons de simplicité, est l'objet principal utilisé dans tous les modes. Avec la nouvelle découpe que j'avais réalisée, il ne restait plus que cet objet à passer entre les différentes tâches. Cependant, je ne savais pas comment transmettre un objet de type `ClassImagerDeconv` via un socket. D'après ce que j'avais compris des explications de Romain Tajan sur les types, seuls les types implémentés dans PyStreamPU pouvaient être utilisés. Par conséquent, j'ai tenté de faire passer le type de l'objet dans le socket, mais cela ne fonctionnait pas.

J'ai alors cherché d'autres solutions. J'ai d'abord envisagé de passer l'objet dans un tableau. Toutefois, avec PyStreamPU, cela nécessitait l'utilisation de tableaux NumPy, et donc je devais connaître la taille du tableau. Or, la taille de cet objet changeait d'une exécution à l'autre, ce qui compliquait les choses.

Une autre solution a été discutée avec Romain Tajan. Étant donné que PyStreamPU est une surcouche Python de la version C++ du logiciel StreamPU, il est possible de créer et de passer l'objet dans la couche C++ et de le traiter ensuite en Python. Cette approche pourrait permettre de surmonter les limitations liées au transfert direct des objets complexes via les sockets.

5.8 Simulation de pipeline avec OTAC

À ce stade, avec Monsieur Aumage, nous avons constaté que je n'aurais pas le temps de terminer même l'implémentation du premier mode de DDFacet. Ce que j'ai décrit ci-dessus peut sembler relativement rapide, mais tout ce qui a été accompli après la mise en place de l'environnement Guix représente deux mois de travail. Il me restait encore le rapport à rédiger, sans compter que je ne m'étais pas encore réellement penché sur la question de la suppression des outils de parallélisation déjà en place dans le projet, ce qui était nécessaire pour pouvoir utiliser PyStreamPU.

Nous avons donc décidé, avec Antsa et Olivier, de mettre ce travail de côté pour un futur stagiaire et d'explorer la réalisation d'une simulation avec le logiciel OTAC, comme mentionné précédemment. Malheureusement, je ne pourrai pas fournir plus de détails à ce sujet, car je réaliserai cette simulation après la soumission de ce rapport. Je n'ai pas encore examiné en profondeur le fonctionnement de OTAC ni les résultats qu'il pourrait fournir pour le moment. Par conséquent, je ne peux pas discuter des résultats potentiels que j'aurais pu obtenir avec DDFacet, car comme vous l'avez compris, je n'ai pas réussi à produire quelque chose de fonctionnel.

6 Conclusion & Perspectives

Ce stage a été une expérience enrichissante et a été un premier pas significatif dans le domaine de la recherche pour moi. Je tiens une fois de plus à remercier Monsieur Aumage pour m'avoir offert cette opportunité.

D'un point de vue performance, je suis quelque peu déçu. J'ai consacré beaucoup de temps à comprendre le fonctionnement de chaque logiciel, ce qui a peut-être pris plus de temps que prévu. Je n'ai pas réussi à être suffisamment efficace pour produire des résultats concrets. Cette situation est évidemment décevante.

Cependant, cette expérience reste très positive sur le plan personnel. J'ai eu l'occasion de voir comment fonctionne un laboratoire de recherche, d'observer l'entraide et le partage des connaissances entre collègues. J'ai particulièrement apprécié cette dimension collaborative du travail, ainsi que la possibilité d'assister à de nombreuses présentations sur les recherches des titulaires et des doctorants, couvrant des sujets variés et intéressants.