CHEVALIER Maxime
Réseaux Informatiques et Communication Multimédia
POLYTECH GRENOBLE
Rapport de stage 4A.

Simulation parallèle optimiste : prise en main et benchmarking

Tome Principal
ET
Annexe

Année universitaire
2016-2017
Période du stage
8 mai 2017 – 6 août 2017

Remerciements

Je tiens à remercier tout le personnel du LIG pour son accueil chaleureux, et plus précisément l'équipe POLARIS.

Je remercie également toutes les personnes ayant participé à l'organisation des conférences.

Je remercie Fabienne Boyer d'avoir accepté d'être ma tutrice pédagogique.

Je remercie Annie Simon de s'être occupée rapidement des démarches administratives (et son porte manteau !).

Je remercie également Vincent Danjean pour son aide précieuse dans le debbuging des différentes applications, pour son aide en script Bash et ses conseils.

Je remercie Lucas Mello Schnorr pour ses conseils et son aide avec Grid5000.

Je remercie tout particulièrement mes encadrants Florence Perronnin et Arnaud Legrand pour m'avoir permis d'effectuer ce stage, m'avoir intégré à l'équipe, m'avoir fait découvrir les conférences organisées et surtout pour le temps qu'ils m'ont accordé.

Enfin, merci à tous pour votre bonne humeur et votre sympathie durant ces trois mois.

Table des matières

Table des illustrations

1. Introduction

1) GRANDS SYSTEMES DE CALCUL

De nos jours, les ordinateurs de calcul à haute performance (HPC : Hight Performance Calcul) envahissent le quotidien, sans être forcément visibles pour le consommateur. Les HPC sont pourtant au cœur d'une compétition internationale[1] et sont les pivots de nombreuses industries (nucléaire, météorologie, recherche, finance, data-science, ...). Aujourd'hui beaucoup d'entreprises possède un cluster (grille de calcul), mais le prix de cet équipement est très élevé, d'où la démocratisation de services comme Amazon Web Service, Windows Azure, Google Compute Engine. Pour ceux qui souhaite posséder leur propre infrastructure, la question du dimensionnement se pose, afin de limiter les coûts.

Le gain de performance de ces supercalculateurs s'explique par la loi de Moore qui prédit le doublement du nombre de transistors dans une puce tous les deux ans. Cependant, de l'aveu même de Gordon Moore, cette loi ne sera plus valide dans quelques années. Il est en effet, de moins en moins évident de surmonter les contraintes physiques liées à de si petites échelles. Les machines contiennent donc de plus en plus de cœurs moins performant pour réduire la consommation d'énergie et la chaleur dégagée, et sont également équipées de GPU (Graphics Processing Unit) permettant un calcul parallèle très efficace sur des opérations de même type. Les supercalculateurs sont également composés de plus en plus de nœuds reliés entre eux par des réseaux complexes. Tous ces facteurs augmentent la complexité des applications déployées sur de telles machines. En effet, elles sont souvent développées à partir de postulat simple (comme la topologie réseau en anneaux). Il n'est donc pas rare d'observer d'énormes gains de performances lorsqu'on optimise de telles applications (utilisation de la bonne topologie réseau, des CPU, des GPU, ...).

Dans cette optique, des simulateurs ont été développés permettant de simuler l'exécution des applications sur des topologies réseaux différentes, en prenant en compte le débit des liens entre les machines, les temps de routages dans les équipements, les temps de stockage de données, le tout résultant d'une simulation fine de l'exécution. Ils permettent également de simuler différentes applications en même temps, combiné à du trafic réseau pour plus de réalisme. En effet, peu d'application utilise l'ensemble des nœuds d'un HPC. La plupart en utilisent quelques-uns, et donc en utilisant des nœuds proches pour une même application on réduit son impact sur les performances réseau. La topologie et le placement des applications sur les HPC est un des grands défis de ce domaine. La simulation prend donc tout son sens et c'est un secteur de recherche aujourd'hui très actif avec plusieurs approches, dont la simulation parallèle et la simulation séquentielle.

[1] https://www.top500.org/

2) OBJECTIFS DU STAGE

Mon stage est composé de quatre objectifs principaux :

1. Familiarisation avec la simulation parallèle et prise en main des outils logiciels (e.g. Codes, SimGrid)
2. Élaboration d'un benchmark commun grâce à la simulation d'HPL
3. Comparaison expérimentale
4. Exploration de différentes pistes d'amélioration algorithmique

Le premier point a été réalisé, mais les difficultés rencontrées pour le second n'ont pas permis d'atteindre le reste des objectifs. Les objectifs 2 et 3 ont donc été traités partiellement et l'objectif 4 n'a pas été abordé.

3) ENTREPRISE

Mon stage a été effectué au sein du Laboratoire Informatique de Grenoble (LIG), plus précisément dans l'équipe Polaris (Performance analysis and Optimization of LARge Infrastructures and Systems), chargé d'étudier les performances des grands systèmes distribués mais aussi les systèmes générant de très grandes quantités de données.

Le LIG accueille 24 équipes de recherche réparties autour de 5 grands axes de recherche :

- Traitement de Données et de Connaissances à Grande Échelle
- Méthodes Formelles, Modèles et Langages
- Systèmes interactifs et cognitifs
- Systèmes répartis, Calcul Parallèle et Réseaux
- Génie des Logiciels et des Systèmes d'Information

Ce laboratoire est au sein du bâtiment IMAG sur le campus de Saint-Martin d'Hères, qui accueille au total 6 laboratoires :

- Agence pour les Mathématiques en Interaction avec les Entreprises et la Société (AMIES)
- Grenoble Alpes Recherche Infrastructure de Calcul Intensif et de Données (GRICAD)
- Laboratoire d'Informatique de Grenoble (LIG)
- Laboratoire Jean Kuntzmann (LJK)
- MI2S
- VERIMAG

Grâce à toutes les compétences présentes au sein du bâtiment, l'IMAG possède une visibilité mondiale, et permet à tous de profiter des nombreux séminaires et des nombreuses conférences en lien avec le monde de l'Informatique.

4) ENVIRONNEMENT DE TRAVAIL ET DEVELOPPEMENT DURABLE

Le bâtiment IMAG, inauguré en 2016, a été entièrement conçu pour réduire et optimiser la consommation d'énergie. En effet, outre l'isolation, le bâtiment bénéficie de volets et de lumière intelligents. Les volets s'ouvrent et se ferment grâce à des capteurs de lumière extérieurs. En été, ils empêchent la lumière d'entrer afin de limiter l'accumulation de chaleur et donc l'utilisation de la climatisation. En hiver, ils laissent au contraire entrer la lumière.

La température est, elle, régulée grâce à des pompes à chaleur réversibles, qui en été s'adaptent à la température extérieure (température du bâtiment en adéquation avec la température extérieure). La géothermie est également utilisée pour rafraîchir le bâtiment. En hiver, en plus des pompes à chaleur, l'air chaud généré par le datacenter est utilisé. Ce dernier a également pour objectif de mutualiser les différents datacenters de l'UGA et des hôpitaux afin de réduire les coûts mais aussi l'impact environnemental. Enfin, la disposition des espaces de travail a elle aussi été conçue par l'architecte pour être la moins énergivore (exemple tous les postes sont près des fenêtres pour la lumière).

Le LIG inscrit lui ses projets dans l'informatique ambiante et durable permettant, grâce à ses équipes pluridisciplinaires, d'intégrer les aspects technologiques et sociétaux à ses recherches.

5) STRUCTURE DU RAPPORT

Je vais dans un premier temps définir dans quel contexte le stage a été effectué, puis je vous décrirai l'état de l'Art actuel. Dans un second temps je vous exposerai la méthode utilisée durant le stage pour l'aborder et le mener à bien. En troisième partie nous verrons les expériences et résultats obtenus.

2. Contexte

Les applications parallèles utilisent la technologie MPI (Message Passing Interface) pour communiquer entre machines distantes, ou entre différents cœurs d'un processeur. Cette technologie développée dans les années 90 s'est rapidement imposée car très simple. Elle permet de communiquer point à point ou de manière collective (tous les processus, ou via des groupes). Le standard MPI est implémenté par différentes librairies, comme MPICH et OpenMPI. D'autres implémentations sont disponibles, comme celle de Intel optimisée pour ses processeurs. Grâce à cette technologie, il est facile de créer des algorithmes parallèles, où chaque processus effectue une partie d'un calcul plus conséquent. Cette technologie a donc favorisé l'apparition de machines de calcul de plus en plus puissantes, permettant de traiter des problèmes de plus en plus importants. Une compétition s'est alors enclenchée entre les pays, car elles permettent des simulations dans des domaines comme la physique ou la chimie, dans des domaines stratégiques comme le nucléaire ou de traiter et d'analyser de grandes quantités de données, ...

Le TOP500[2] des supercalculateurs a donc vu le jour. Aujourd'hui le plus puissant est le SUNWAY TAIHULIGHT (Chine) possédant plus de 10 millions de cœurs et ayant une puissance potentielle de 125 435,9 TFlop/s[3] (actuellement le maximum observé est 93 016,6 TFlop/s). La France est classée 19ème avec le Pangea (220 800 cœurs, 5 283,1 TFlop/s).

Ce TOP500 est basé sur les résultats du benchmark HPL[4] utilisant MPI. Ce benchmark calcule la décomposition LU d'une matrice de grande taille $A \in \mathbb{R}^{n \times n}$. Il permet ainsi via le nombre d'opération équivalent à $\frac{2}{3}n^3 + 2n^2 + O(n)$ et le temps de calcul d'obtenir la puissance en Flop de la machine sur laquelle il est exécuté. C'est un programme qui se

[2] https://www.top500.org/
[3] Tera floping-point operations per second
[4] HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, créé par Jack Dongarra en 1979, http://www.netlib.org/benchmark/hpl/

parallélise très bien car il y a relativement peu d'échanges entre les nœuds et beaucoup de calcul. C'est donc le premier programme à être lancé sur ces supercalculateurs, chaque équipe pouvant choisir la taille du problème et les différents algorithmes de résolution afin d'obtenir le meilleur résultat. En effet, chaque super calculateur possède sa propre topologie réseau, souvent complexe. Afin qu'HPL puisse être générique il a été conçu par rapport à une topologie réseau plus simple (en général les applications parallèles sont développées sur le modèle de la topologie réseau "en anneau") qui ne correspond pas à la topologie réelle, pouvant impacter les performances, d'où les nombreux paramètres d'optimisation mis à disposition par HPL. Ce dernier est pourtant critiqué car il ne reflète pas la performance réelle d'un système, mais en donne seulement une idée.

Un autre benchmark, nommé AMG[5] résout un système linéaire sur des grilles 3D non structurées. Il découpe le problème en cubes de taille égale qu'il assigne aux processus MPI. Ces derniers calculent donc un système 3D et communiquent les résultats à leurs voisins, permettant ainsi de continuer la résolution. Il nécessite beaucoup d'accès mémoire et de communications entre les processus MPI, ce qui le rend très synchrone (au contraire d'HPL) et donc plus proche des calculs scientifiques réalisés de nos jours.

3. Etat de l'Art

Comme indiqué précédemment, il existe plusieurs types de simulation pour les applications parallèles. La simulation séquentielle où le simulateur possède son propre temps interne et un ordonnanceur qui « séquentialise » les tâches des différents processus, et la simulation parallèle avec deux approches possibles[6].

L'approche "space parallel" (Figure 1.a) décompose horizontalement l'exécution d'un programme, où chaque processus maintient ses variables d'état tout au long de la simulation. Cette approche est flexible et applicable dans la plupart des cas.

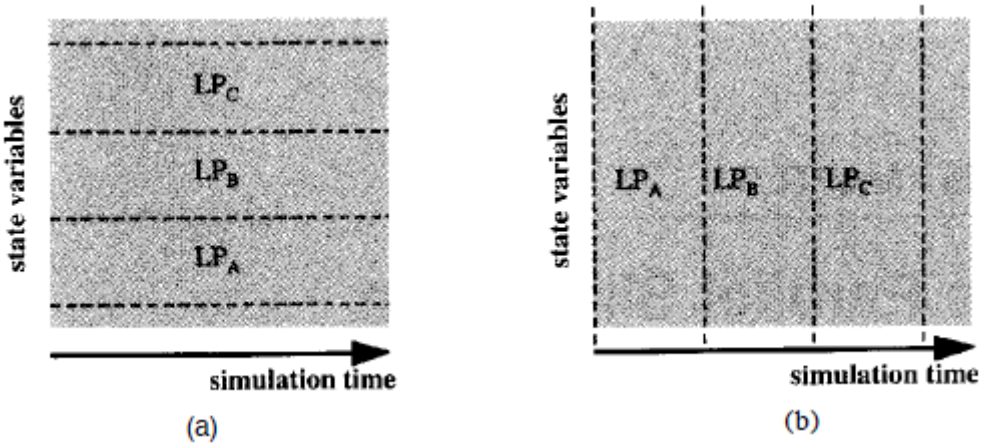

Figure 1 (a) Approche "Space parallel" (b) Approche "Time parallel" [7]

L'approche « time parallel » (Figure 1.b) décompose verticalement l'exécution d'un programme, où les processus sont en charge de toute la simulation sur une période de temps

[5] Algebraic Multigrid solver, http://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks/amg/
[6] Parallel and Distributed Simulation System, Fujimoto R. M., 2000, pp. 178-191
[7] Parallel and Distributed Simulation System, Fujimoto R. M., 2000, figure 6.1

donné. Cette approche maximise le parallélisme car les processus sont alors indépendants les uns des autres. Cependant, si dans l'approche (a), les processus ont tous leurs états de départs, dans l'approche (b) c'est le cas seulement pour LP_A.

Pour déterminer l'état initial des process LP_B et LP_C, il y a trois approches possibles

- **Le calcul par réparation (Fix-Up computations) :** Le processus calcule une trajectoire à partir d'un état initial aléatoire. Une fois l'état final de LP_A déterminé, si ce dernier correspond à l'état initial de LP_B c'est gagné, sinon il faut recalculer LP_B. C'est le même raisonnement pour tous les processus. Cette technique n'est pas optimale car il est rare que l'état final soit le même que l'état initial de la période suivante. Cependant, en gardant les traces d'exécutions précédentes, un processus tombant sur un état précédemment calculé, connait déjà son chemin d'exécution. La Figure 2 est un exemple de ce type d'approche. En simulation séquentielle il faudrait 4 temps de calcul pour effectuer la simulation complète, mais dans cet exemple il n'en faut que 3.

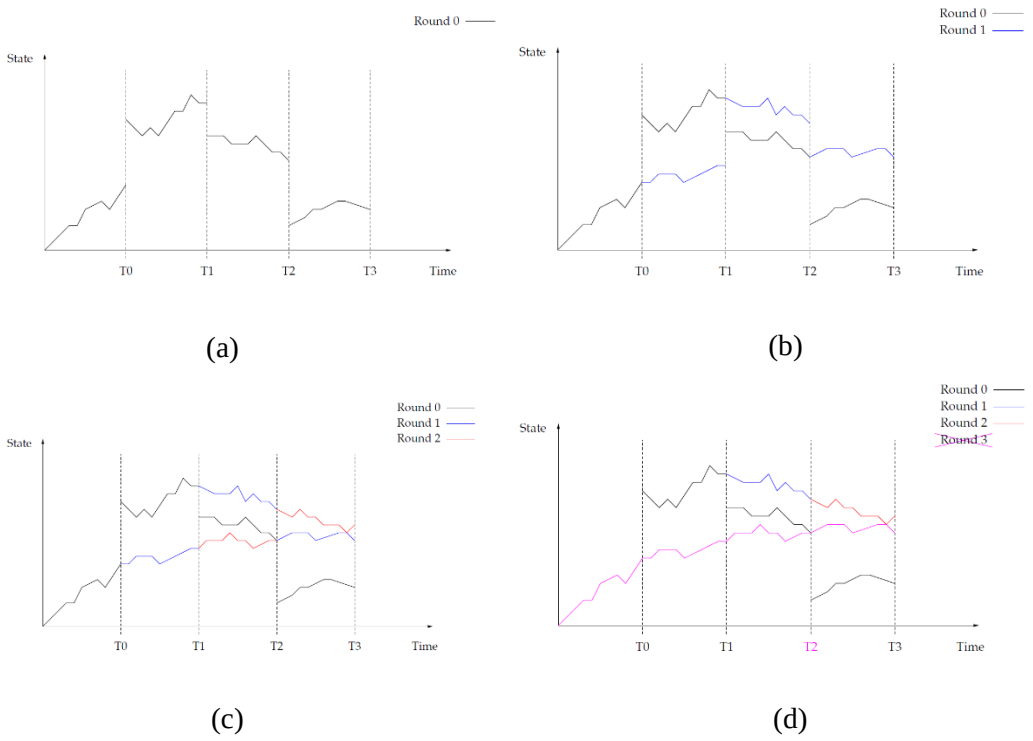

Figure 2 Exemple du fonctionnement de l'algorithme Nicol (Pre-Fix Computation)[8]

- **Le pré-calcul d'état à des temps spécifiques (Precomputation of state at specific time division points) :** En se basant sur des états du système pouvant facilement être déterminés (comme un overflow/underflow) il est possible de repartir la charge de calcul entres ces points. L'état initial devient donc un postulat de départ, et lorsqu'un processus termine son temps d'exécution, il transmet son état final au processus suivant, qui calcule la trajectoire jusqu'à obtenir son propre état initial. Cette approche est difficile à mettre en place, car il faut trouver de tels postulats, et il est possible de ne jamais les atteindre (par exemple un overflow est très rare).

[8] Graphiques tirés du diaporama « Parallel Sampling of Markov Chains » de Benjamin Briot et Jean-Marc Vincent (2016)

- **Le calcul par préfix (Parallel prefix computations) :** Si l'état de la simulation peut être formulé de manière linéaire, cette approche peut être utilisée car il existe de très bons algorithmes de résolution parallèle.

Nous allons en premier lieu étudier les différentes approches de la simulation « space parallel » décrites dans « Parallel and Distributed Simulation » de Richard Fujimoto [1]

- **La synchronisation conservative :** Permet d'avoir le même résultat qu'une exécution séquentielle sur un processeur monocœur. Cette propriété peut être montrée en s'assurant que chaque processus logique traite les messages MPI par leur date. Dans les premières versions de ces algorithmes, afin qu'il y ait du parallélisme dans la simulation, l'algorithme CMB permettait de s'assurer que les processus logiques traitent leurs évènements seulement si aucun évènement de date plus petite existe dans le système. Pour cela il fallait un réseau FIFO, et la mise en place d'un lookahead. Ce dernier assure un temps minimum entre l'envoi de deux messages par un processus, et permet donc d'avancer dans la simulation. Les dernières versions de ces algorithmes ont tenté de supprimer le lookahead et ont remplacé l'algorithme CMB par d'autres techniques comme l'algorithme YAWNS qui utilise des points globaux de synchronisation.
- **La synchronisation optimiste :** Cette technique repose sur un principe simple, toutes les opérations effectuées doivent être réversibles. Il faut donc définir l'inverse de chaque fonction, et stocker tous les messages ainsi que tous les états des variables. Cette technique est donc très consommatrice de mémoire. Grâce à cela, les processus avancent chacun à leur rythme et si un processus reçoit un évènement qu'il aurait dû traiter précédemment, il effectue un rollback à la date de l'évènement grâce aux informations stockées et s'il a envoyé des messages entre les deux, il envoie l'inverse des messages envoyés, qui s'ils n'ont pas été traités s'annulent avec le vrai message, et s'ils ont été traités propage le rollback (cela permet de faire du rollback en cascade simplement). Puis il reprend son exécution à la date du message reçu. Afin de limiter la consommation de mémoire (il serait mal avisé de sauvegarder l'ensemble de la simulation), un GVT (global virtual time) permet de s'assurer que le temps T est le temps minimum de la simulation, et donc de supprimer toutes les informations précédant T. C'est une barrière temporelle, avant laquelle on ne peut plus revenir.

Dans ce rapport nous allons nous concentrer sur la simulation optimiste et la simulation séquentielle.

Il existe deux approches pour capturer le fonctionnement d'une application MPI. L'approche off-line et l'approche on-line.

L'approche off-line consiste à exécuter l'application MPI, et à enregistrer tous les appels MPI (rendu possible par l'interface PMPI qui permet l'enregistrement de callback à toutes les fonction MPI) afin d'obtenir une trace de l'application, contenant chaque appel MPI, daté, avec la source et la destination. Une fois cette trace obtenue, il est possible de la rejouer sur un simulateur, en changeant les paramètres d'exécution, comme la topologie réseau. Cependant cette approche est limitée, car elle nécessite le lancement de l'application sur un HPC pour avoir la trace. De plus, le changement des paramètres, comme la topologie, n'a pas forcement de sens car les traces sont réalisées sur une topologie spécifique, et l'exécution de l'application n'est pas forcement déterministe. Enfin, certains simulateurs proposent

d'extrapoler les traces à partir de plus petites, mais l'extrapolation de traces d'application s'adaptant à la topologie ou aux entrées/sorties n'a pas de sens, puisqu'on reporterait les possibles défauts (par exemple de la topologie) sur une autre, mais également les points forts de l'exécution initiale. Il faut donc se montrer prudent avec cette approche.

L'approche on-line, permet d'exécuter l'application dans un environnement virtuel contrôlé, une sorte d'émulation. L'analyse de la trace ainsi obtenue lors de l'émulation est bien plus significative car on maitrise l'environnement d'exécution. Il est donc possible de comparer ces différentes topologies car elles ont réellement été prises en compte lors de l'émulation. De plus cette approche permet de simuler des HPC sur un ordinateur portable, et ne nécessite donc pas l'utilisation d'un HPC.

Je vais maintenant vous présenter différents outils disponibles dans ce domaine (non exhaustif).

SimGrid, développé par l'INRIA, est un simulateur d'applications parallèles. Il permet, à partir de modèles, de simuler une exécution à très grande échelle (en million de nœuds), mais aussi d'exécuter une application MPI directement sur le simulateur, afin de tracer son exécution, et ensuite de pouvoir rejouer ces traces dans le simulateur avec des paramètres différents, comme la topologie réseau. Le simulateur est séquentiel et est optimisé pour, même si une version parallèle (optimiste) existe. Ce simulateur permet donc de générer de grosses traces d'exécution sans avoir accès à un super calculateur, car le temps est interne au simulateur (approche on-line)

Psi3 (Perfect simulator) est un autre simulateur développé par l'Inria dédié aux files d'attentes réseaux (en temps discret). Le logiciel fournit des traces d'exécution qui sont ensuite utilisables pour analyse. Il est basé sur les chaînes de Markov et permet d'effectuer des simulation « space parallel » et « time parallel ».

ROSS (Rensselaer's Optimistic Simulation System) est un simulateur parallèle à évènements discrets. Il s'exécute de manière parallèle et permet de simuler des modèles à plusieurs millions d'objet (un objet étant un élément du système). Il permet une exécution optimiste, mais aussi une exécution séquentielle.

CODES (Co-Design of Exascale Storage) est un simulateur basé sur ROSS. CODES a pour but d'utiliser le parallélisme massif afin d'explorer les architectures réseaux et de stockage de demain, pour les usages scientifiques. Ce simulateur profite de tous les avantages de ROSS, et permet en plus de rejouer des traces d'exécution au format DUMPI (approche off-line).

TraceR est un outil basé sur CODES utilisé pour prédire les performances réseaux en simulant l'exécution d'application de calcul haute performance par envoi de messages. Il permet de rejouer des traces d'application au format OTF2 (Scroe-P) et BigSim (approche off-line).

SST (Structural Simulation Toolkit) est un ensemble d'outils permettant d'explorer les systèmes à accès concurrent, où le matériel, la programmation et le système de communication impactent les performances. Il permet à la fois de travailler sur des modèles et de faire du « rejeu de trace » au format OTF2 et DUMPI (approche on-line et off-line)

Tous ces simulateurs proposent le « rejeu » de trace d'application via les deux approches décrites plus haut. Ils utilisent ensuite ces informations sur les messages pour effectuer la simulation dans leurs modèles ce qui permet d'aller plus vite qu'une exécution réelle car il n'y a pas de calcul effectué.

Le traçage possède un autre avantage : il est possible grâce à lui de visualiser l'exécution d'une application parallèle grâce à des applications comme Vite ou Vampir. Il est également possible, en plus de la génération de trace, d'utiliser des outils comme TAU (Tuning and Analysus Utilities) qui permettent de profiler l'exécution d'une application, et donc d'observer pourquoi les performances chutent et ainsi facilement déterminer ce qui peut être amélioré.

Aujourd'hui les acteurs du secteur n'utilisent pas le format de trace standardisé récemment, qui est l'OTF2 (Open Trace Format version 2), et il est donc très difficile de comparer ces systèmes, chacun ou presque, ayant un format propriétaire.

4. Méthodologie

- Organisation

Tout d'abord, ce stage a été effectué dans une optique de reproductibilité. Tout ce qui a été accompli a été noté dans un journal, ce qui a également permis à mes encadrants de suivre l'évolution mon travail (le document en question étant disponible sur GitHub). La chronologie et la disponibilité du document en ligne nous ont également aidés lorsque mes encadrants étaient en déplacement ou lors des préparations de réunion. Nous avons également organisé plusieurs réunions afin de définir/ redéfinir les objectifs au fur et à mesure de mes avancées (avec au minimum un point par semaine, des échanges de mail quasi-quotidien).
Ensuite, afin de m'aider dans mon organisation, j'ai mis en place un tableau de bord sur GitHub (cf. Annexes 3) contenant 4 colonnes : à faire, en cours, fait et piste abandonnée. Ainsi j'ai pu travailler sur différentes pistes à la fois, indépendamment de l'état de chacune (certaines étant bloquées, comme HPL lors de l'interaction avec les utilisateurs de CODES via la mailing list).

- Familiarisation

Pour aborder ce stage de recherche dont l'objectif principal était d'évaluer et de comparer les simulations proposées par CODES et SimGrid grâce au benchmark HPL utilisé par tous les super calculateurs pour le TOP500, il m'a premièrement fallu découvrir l'état de l'Art, et le comprendre afin de bien cibler les enjeux du stage. Après avoir étudié les différents types de simulation existants grâce aux travaux de Fujimoto, Briot – Vincent, Jefferson, Liu – Carothers (cf. Références), ce qui m'a pris environ 2 semaines, il a fallu que je comprenne et que j'installe chaque outil nécessaire à mon objectif.

- Prise en main

Dans un premier temps je me suis concentré sur l'utilisation de CODES, car ce n'est pas un outil utilisé par l'équipe POLARIS, contrairement à SimGrid. Nous avons tenté de générer des traces DUMPI de l'exécution d'HPL et de les rejouer sur CODES, mais malheureusement, cela n'a pas fonctionné. Après un échange de mail avec la mailing list de

CODES, nous avons découvert que les traces DUMPI étaient incohérentes, avec des messages envoyés à un processus X qui arrivaient en fait à un processus Y. L'équipe de CODES a supposé l'existence d'un overflow dans l'outil de traçage DUMPI développé par SST. Cette irrégularité a permis à l'équipe de CODES de mettre en place un système de vérification des traces.

Il a donc été décidé d'utiliser le simulateur TraceR, basé sur CODES mais utilisant des traces OTF2 générées par la librairie Score-P. Après l'envoi d'un bug report à l'équipe de Score-P, nous avons pu générer des traces OTF2 de l'exécution d'HPL, mais là encore le replay n'a pas fonctionné. Le problème vient donc d'HPL qui optimise son fonctionnement en utilisant MPI de manière asynchrone.

Après cet échec, nous avons décidé de ne pas utiliser HPL, qui possède de très bonne prédisposition au « space parallel », mais AMG, dont le replay a fonctionné sur CODES et TraceR. De même, l'exécution d'AMG ne posait pas de problème sur SimGrid, et il était possible de rejouer de petites traces (8 processus).

Toujours dans un souci de reproductibilité, un script d'installation de tous ces outils a été créé afin de faciliter la poursuite de cette recherche.

Un résumé des outils utilisés et leur interfaçage est disponible en annexe. La partie verte étant ce qu'on pensait devoir utiliser au début du stage, le reste représentant l'analyse de l'existant.

5. Expérimentations

Afin de réaliser les expériences, l'installation et la configuration ont dans un premier temps été réalisées sur mon ordinateur portable personnel, mais au vu du nombre de logiciels et des difficultés d'interactions entre les composants, je suis ensuite passé sur une machine virtuelle afin d'élaborer un script d'installation.

Le plan d'expérimentation souhaité est montré dans la figure ci-dessous

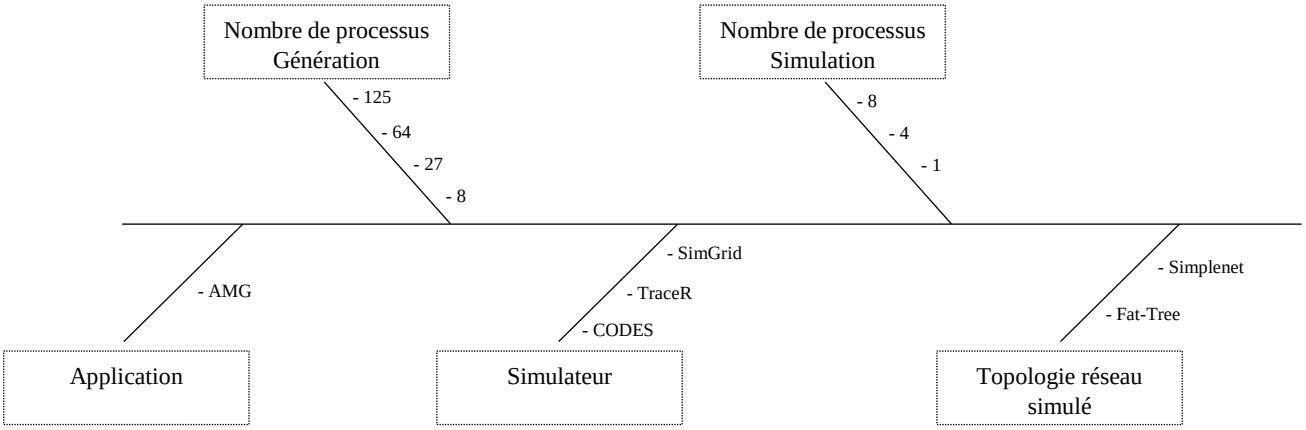

Figure 3 Plan d'expérience

L'expérience consiste donc à tester AMG sur des tailles différentes et assez proches afin de voir la tendance des résultats, et de rejouer les traces générées sur différents simulateurs en utilisant deux modes de simulation : séquentiel (1 processus) et optimiste (4 et 8 processus),

sur des topologies réseaux différentes pour déterminer l'impact de la topologie sur les simulations.

Une fois tous les outils installés, il a fallu se décider sur la manière de générer des traces d'application de grande taille. Dans un premier temps nous avons pensé utiliser Simgrid, car il a déjà permis dans une autre recherche, de tracer au format pajé (différent de celui utilisé de base dans SimGrid). Cependant, les nouvelles versions de SimGrid étant toujours plus optimisées en sont également devenues plus complexes, et il n'a pas été possible de tracer dans un autre format. Nous avons donc décidé d'utiliser Grid5000, un testbed français réparti dans toute la métropole. Il est composé de 1000 nœuds pour un total d'environ 8000 cœurs. Une fois l'image fonctionnelle d'un système Linux avec tous les outils créés, nous avons pu lancer nos expériences (sur un maximum de 125 cœurs). Cependant, il s'est avéré impossible d'utiliser TraceR sur Grid5000, nous ne l'avons donc pas testé. De même, il s'est avéré impossible de rejouer des traces de plus de 8 processus avec SimGrid. Seule l'émulation d'AMG a été effectuée avec SimGrid

Les expériences ont été réalisées durant le dernier mois de stage.

6. Analyse des résultats

L'état des simulateurs selon la plateforme en fin de phase d'expérimentation est récapitulé ci-dessous :

	Machine personnelle			**Grid5000**		
	SimGrid	Codes (DUMPI)	TraceR (OTF2)	SimGrid	Codes (DUMPI)	TraceR (OTF2)
AMG	✓	✓	✓	✓	✓	✗
HPL		✗	✗		✗	✗

Figure 4 Tableau récapitulatif du fonctionnement des outils par plateforme

Légende :

✓ Le traçage et la simulation fonctionnent

✓ Le traçage et l'émulation fonctionnent, mais pas la simulation

✗ Le traçage et/ou la simulation ne fonctionne pas

Le benchmark final porte donc sur les simulateurs SimGrid (en émulation) et CODES (en « rejeu » séquentiel et parallèle) avec en point de comparaison l'exécution réelle d'AMG. Afin que toutes les expériences, de la génération à la simulation, ou de l'exécution, ou de l'émulation soient réalisées sur les mêmes machines, nous avons utilisé le cluster Edel (Grid5000) basé à Grenoble. Dans les résultats ci-dessous, la charge de calcul de l'exécution de base d'AMG augmente en même temps que le nombre de processus utilisé (le problème à calculer est de plus en plus gros).

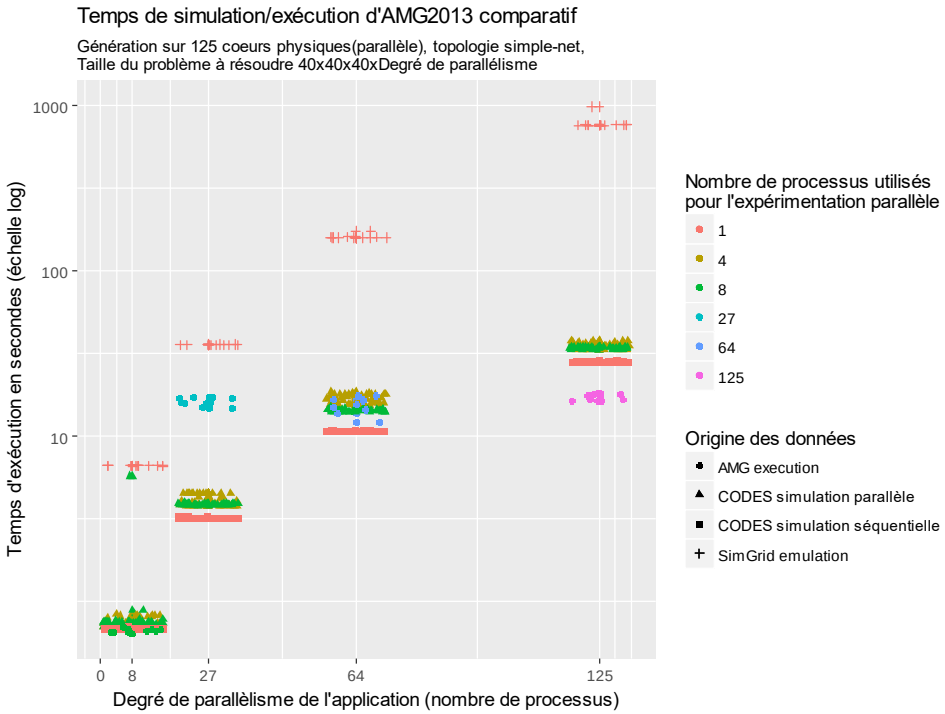

Figure 5 Comparatif du temps de simulation/exécution d'AMG 2013 sur un petit problème

Dans la figure ci-dessus, vous avez un aperçu général des résultats pour un problème de taille *40x40x40xDegré de parallélisme*. Les expériences sur CODES ont été répétées 50 fois, et les autres expériences 10 fois pour éliminer les données aberrantes. L'échelle de l'axe des ordonnées est en logarithme Les données ont été écartées horizontalement pour plus de lisibilité, mais correspondent bien, par groupe, à la valeur notée en abscisse.

On remarque que le temps d'émulation de SimGrid explose à grande échelle. Ce n'est donc pas une solution viable pour un code non préparé (SimGrid a été préparé pour accueillir HPL, et est capable d'en simuler une exécution avec une matrice de taille 4 000 000 et 4 096 processus MPI, ce qui représente un HPC, en 47 heures)[9]

En ce qui concerne le temps d'exécution d'AMG, il varie peu entre 27, 64 et 125 processus (entre 16 et 17 secondes). Il prend le même temps voir moins que les simulateurs sur 64 et 125 processus. Pour ce benchmark et cette taille de problème, la simulation n'apporte rien, le temps d'exécution de base étant inférieur au temps de simulation.

Enfin en ce qui concerne CODES, on remarque que la simulation séquentielle est plus rapide que la simulation optimiste en « space parallel » sur ces tailles d'instance. Mais pour y voir plus clair, nous allons dans un premier temps zoomer sur le degré de parallélisme le plus petit (8 processus MPI).

[9] Capacity Planning of Supercomputers: Simulating MPI Applications at Scale , Tom Cornebize, 2017

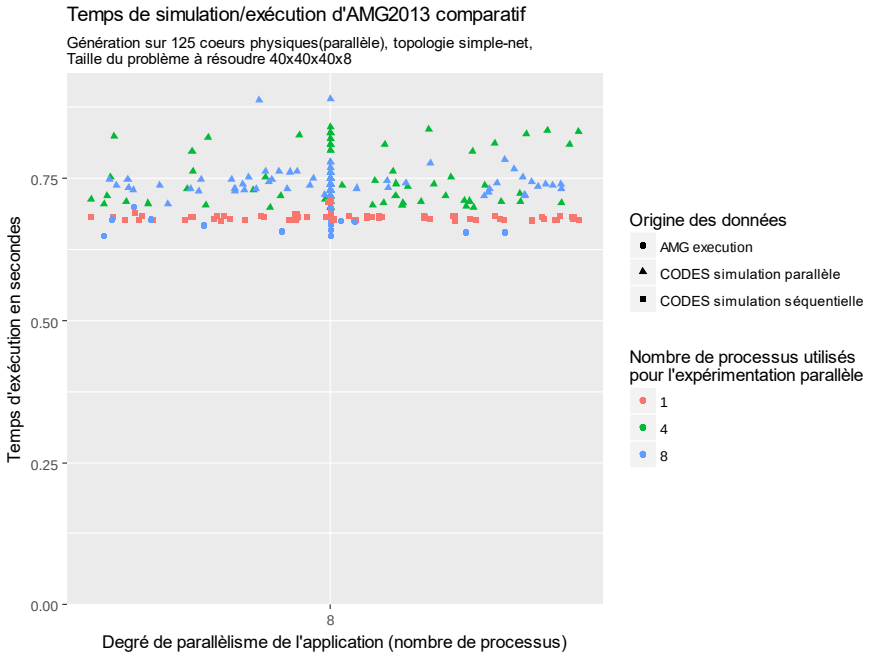

Figure 6 Comparatif des temps de simulation/exécution d'AMG2013 sur un degré de parallélisme 8

Avec cette taille de problème, on remarque que la simulation prend déjà plus de temps que l'exécution réelle, et que la simulation parallèle prend plus de temps que la simulation séquentielle, ce qui s'explique par le coût de création des processus.

Maintenant, analysons seulement les différents types de simulation proposés par CODES, soit le séquentiel et l'optimiste (parallèle).

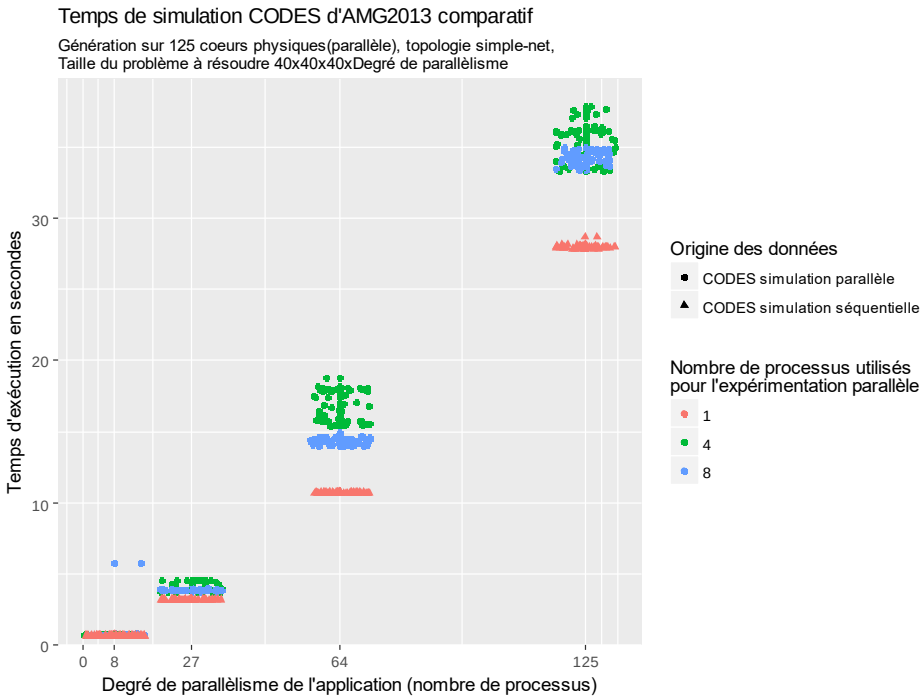

Figure 7 Comparatif des temps de simulations d'AMG2013 sur CODES en fonction de la méthode utilisée

Sur cette figure, le résultat est clair. Pour ces tailles de données, la simulation optimiste n'est pas efficace. L'écart entre les deux types de simulation pour les mêmes données initiales

augmentent au fur et à mesure que la taille du problème augmente. Mais il est toujours possible que la tendance s'inverse à plus grande échelle.

Nous avons donc augmenté la taille du problème pour voir si la tendance se confirme, ou s'inverse. On passe donc d'une taille de problème maximum (pour 125 processus) de 8 000 000 à 125 000 000.

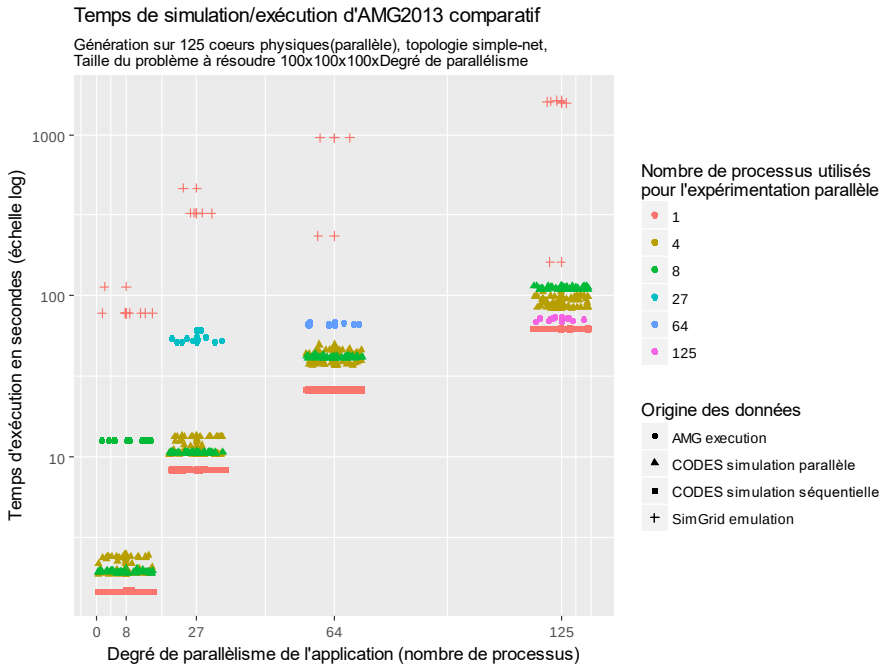

Figure 8 Comparatif du temps de simulation/exécution d'AMG 2013 sur un grand problème

Au premier coup d'œil, on observe sur la Figure 8 la même tendance de courbe que sur la Figure 5. Cependant, il y a deux changements par rapport à la taille de problème précédente. L'émulation d'AMG par Simgrid (représenté par + sur la figure 8) n'est pas stable, il aurait fallu, en ayant plus de temps, augmenter le nombre d'expériences pour ce type de données afin de supprimer les valeurs aberrantes ou confirmer l'instabilité. Le second changement concerne le temps d'exécution d'AMG, qui est plus long que le temps des différents types de simulation. La simulation devient donc intéressante à partir de ces tailles de problèmes. Pour 125 processus MPI, seule la simulation séquentielle fait mieux que l'exécution réelle d'AMG, d'environ 13% (soit une dizaine de secondes).

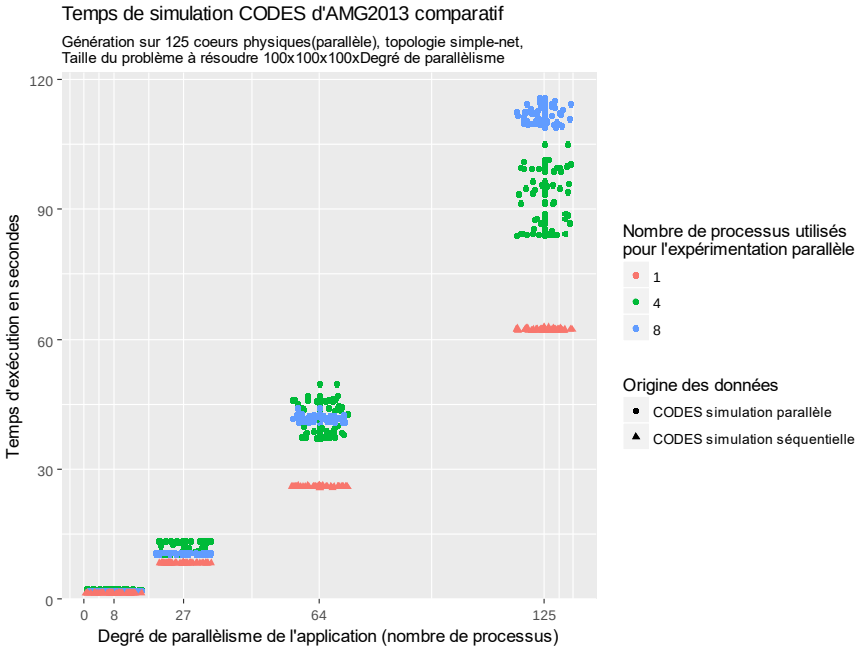

Figure 9 Comparatif des temps de simulations d'AMG2013 sur CODES en fonction de la méthode utilisée

En analysant la figure ci-dessus, concernant la simulation réalisée par CODES, on remarque deux phénomènes. Le premier concerne l'écart toujours plus important entre la simulation séquentielle et la simulation optimiste. La tendance ne laisse pas présager une réduction d'écart à grande échelle. Le second point porte sur l'instabilité des simulations parallèles. Avec une simulation optimiste de 4 processus, l'instabilité explose mais comprend cependant une borne inférieure. La simulation avec 8 processus semble plus stable, mais laisse présager une instabilité du même ordre à une échelle supérieure. Cette instabilité est dû au principe de la simulation optimiste, et au principe de « roll-back », qui affecte l'ensemble des performances s'il a lieu.

7. Conclusion

Les résultats préliminaires laissent à penser que la simulation optimiste n'est pas une voie d'avenir. Cependant, un seul benchmark a été réalisé avec AMG et celui-ci n'est pas le plus optimal pour ce genre de simulation, car il n'y a pas de grosses phases de calcul, mais plein de petites avec beaucoup de communications (comme montré en ANNEXES avec la visualisation de traces). De plus, seule une topologie réseau a été testée durant l'expérimentation. En ce qui concerne l'absence de résultats de la simulation d'HPL, cela montre que les simulateurs ne sont actuellement pas génériques, et ne s'adaptent pas à tous les logiciels utilisant MPI. Cela s'explique en partie par le fait que HPL optimise l'utilisation de MPI (à son profit) pour ne pas être bloqué pendant ses phases de calcul (implémentation de ses propres opérations collectives). Il est cependant regrettable que ces simulateurs d'HPC ne soient pas capable de rejouer les traces de l'application la plus connue dans le domaine (HPL). De plus, ces différents simulateurs mettent davantage en avant la simulation basée sur des modèles d'application, qui permettent de simuler facilement des instances de plusieurs millions de nœuds. Dans chaque documentation, le « rejeu » de trace ou l'émulation d'application MPI n'étaient donc pas très détaillés, ni facilement atteignables.

1) Court terme

A court terme il faut, pour finir le travail, réussir à intégrer SimGrid dans le processus d'expérimentation, ainsi que HPL. Et pour aller plus loin, effectuer les expériences à plus grande échelle. Pour rappel voici ce qui a été réalisé :

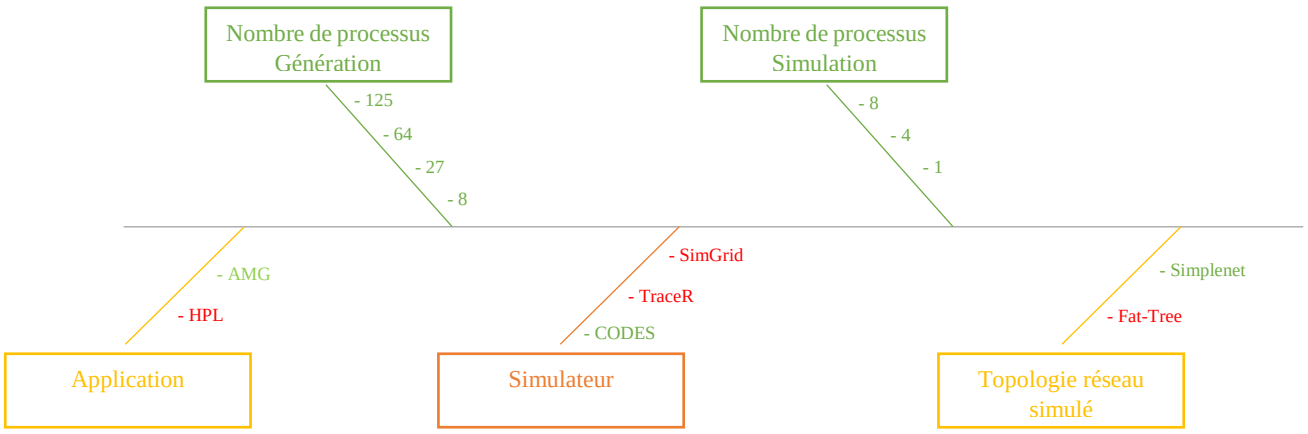

Figure 10 Bilan d'avancement du stage par rapport au plan d'expérience initial

Avec en rouge ce qui manque à l'expérience, et en vert ce qui a été réalisé.

Il serait également intéressant de garder la même taille de problème, indépendamment du nombre de processus utilisé. Avec AMG il faut utiliser la formule suivante

$$\sqrt[3]{\frac{tailleProblème}{nombreProcessus}}$$

Ainsi, en prenant la plus grosse taille de problème utilisée durant ces expériences, soit $100\times100\times100\times5\times5\times5 = 125000000$ cela donnerait les instances d'AMG suivantes :

Taille du cube de processus	Taille problème par processus	Taille totale
2x2x2	250x250x250	125000000
3x3x3	167x167x166	124998498
4x4x4	125x125x125	125000000
5x5x5	100x100x100	125000000

Il faut que l'ensemble reste le plus cubique possible afin de maximiser l'efficacité d'AMG. Avec ce protocole d'expérimentation, on observerait peut-être une meilleure performance de la simulation optimiste.

2) Long terme

Dans un futur à plus ou moins long terme, il serait nécessaire que ces simulateurs utilisent le même format de trace. En effet, les utiliser et les comparer s'avère d'une grande complexité car chaque format nécessite une librairie spéciale, un outil de visualisation différent, etc. … Un format commun permettrait à la communauté de se comparer, et donc de savoir si les pistes engagées sur de nouvelles méthodes de simulation sont efficaces. Une bibliothèque de trace pourrait alors voir le jour afin d'aider cette comparaison en fournissant des traces saines, de petite et grande taille, des principaux benchmarks et applications parallèles utilisées actuellement[10]. Un format de trace unique permettrait aussi de développer un outil de visualisation performant, gratuit et disponible sur toutes les plateformes. Actuellement, seul Vampir permet de visualiser des traces OTF2, mais il est payant.

8. Bilan personnel

Ce stage de plus de trois mois a été une expérience enrichissante. Ayant déjà effectué un stage en entreprise lors de mon DUT Informatique où j'ai développé une application en relation étroite avec les utilisateurs finaux, j'étais curieux de connaitre le monde de la recherche afin d'avoir une autre perspective. J'ai beaucoup appris de par les nombreuses lectures nécessaires concernant le sujet en lui-même, mais aussi grâce aux conférences et soutenances de thèses auxquelles j'ai pu assister. J'ai également pu me former, dans une certaine mesure, aux outils de débogage GDB et Valgrind, à l'utilisation de Git, R (et plus précisément ggplot2) ainsi qu'au langage script Shell et Bash. J'ai pour la première fois, utilisé un testbed (Grid5000) afin de réaliser mes expériences. J'ai également une meilleure compréhension du fonctionnement des applications parallèles et des simulateurs, ainsi que leur importance dans ce domaine. Ce stage a également éprouvé mon sens de l'organisation, le nombre de piste de recherche étant important il m'a fallu m'organiser pour pouvoir les faire en parallèle sans me perdre entre les différentes pistes. Enfin, j'ai également eu l'occasion de pratiquer mon anglais au quotidien, mon voisin de bureau étant russe, et les mail/rapport de bug étant rédigés en anglais.

[10] Comme fait sur https://portal.nersc.gov/project/CAL/designforward.htm, qui propose pour différents benchmarks 2 formats de traces (DUMPI et IPM)

9. Références

[1] R. M. Fujimoto, «PARALLEL AND DISTRIBUTED SIMULATION SYSTEMS,» 2000.

[2] «Rensselaer's Optimistic Simulation System (ROSS),» [En ligne]. Available: http://carothersc.github.io/ROSS/about.html.

[3] «Co-Design of Exascale Storage (CODES),» [En ligne]. Available: https://xgitlab.cels.anl.gov/codes/codes.

[4] «Co-Design of Exascale Storage (CODES),» [En ligne]. Available: http://www.mcs.anl.gov/projects/codes/.

[5] «Trace Replay tool (TraceR),» [En ligne]. Available: https://github.com/LLNL/tracer.

[6] «Structural Simulation Toolkit (SST) DUMPI Trace Library,» [En ligne]. Available: https://github.com/sstsimulator/sst-dumpi.

[7] «TAU Performance System®,» [En ligne]. Available: https://www.cs.uoregon.edu/research/tau/home.php.

[8] «AMG2013 Homepage,» [En ligne]. Available: http://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks/amg/.

[9] T. Cornebize, «Capacity Planning of Supercomputers: Simulating MPI Applications at Scale,» [En ligne]. Available: http://hal.univ-grenoble-alpes.fr/hal-01544827/document.

[10] «Vampir (Visualization Tool),» [En ligne]. Available: https://www.vampir.eu/.

[11] «Versatile Simulation of Distributed Systems (SimGrid),» [En ligne]. Available: http://simgrid.gforge.inria.fr/index.php.

[12] «Scalable Performance Measurement Infrastructure for Parallel Codes (Score-P),» [En ligne]. Available: http://www.vi-hps.org/projects/score-p/.

[13] «Grid5000,» [En ligne]. Available: https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home.

[14] «HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers,» [En ligne]. Available: http://www.netlib.org/benchmark/hpl/.

[15] «Dumpi Trace Collection,» [En ligne]. Available: https://portal.nersc.gov/project/CAL/designforward.htm.

[16] «Structural Simulation Toolkit (SST),» [En ligne]. Available: http://sst-simulator.org/.

[17] F. W. L. B. M. D. P. H. P. L. K. S. J. T. V. W. J. W. H. Y. (. P. L. a. S. B. (. F. S. U. David Jefferson (UCLA) and Brian Beckman, «Distributed Simulation and the Time Warp Operating System,» 1987.

[18] C. C. J. C. P. C. R. R. Ning Liu, «Model and Simulation of Exascale Communication,» 2012.

[19] N. L. S. L. P. C. C. C. R. R. Jason Cope, «CODES: Enabling Co-Design of Multi-Layer Exascale Storage Architectures,» 2011.

[20] R. Fujimoto, «PARALLEL AND DISTRIBUTED SIMULATION,» 2015.

[21] A. L. a. Y. R. Henri Casanova, Parallel Algorithms, CRC Press, 2008.

[22] «Top 500 The List,» [En ligne]. Available: https://www.top500.org/.

[23] J.-M. V. Benjamin Briot, *Parallel Sampling of Markov Chains.*

10. ANNEXES

1) SCHEMA RECAPITULATIF DES OUTILS UTILISES, ET LEURS INTERACTIONS

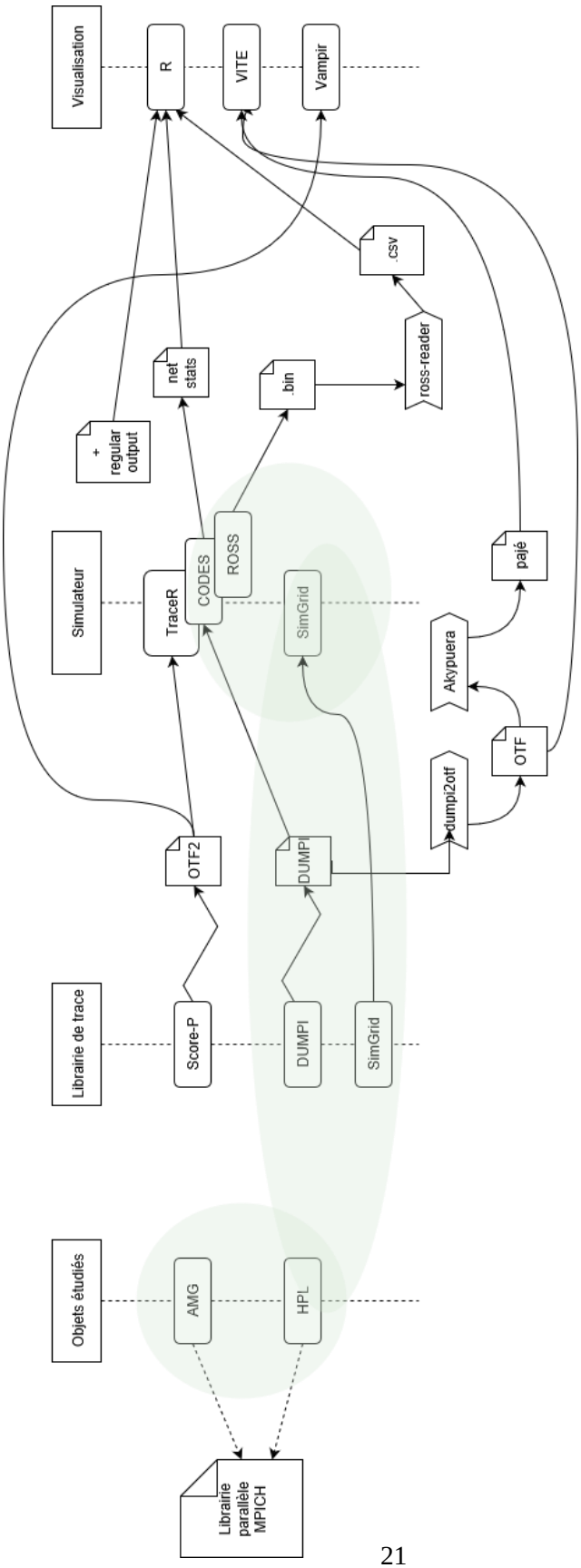

Figure 11 Schémas récapitulatif des outils utilisés et leurs interactions.

En vert, les outils à la base du stage, le reste a été découvert pendant le stages

2) VISUALISATION D'UNE TRACE AMG (8 PROCESSUS) AVEC VAMPIR

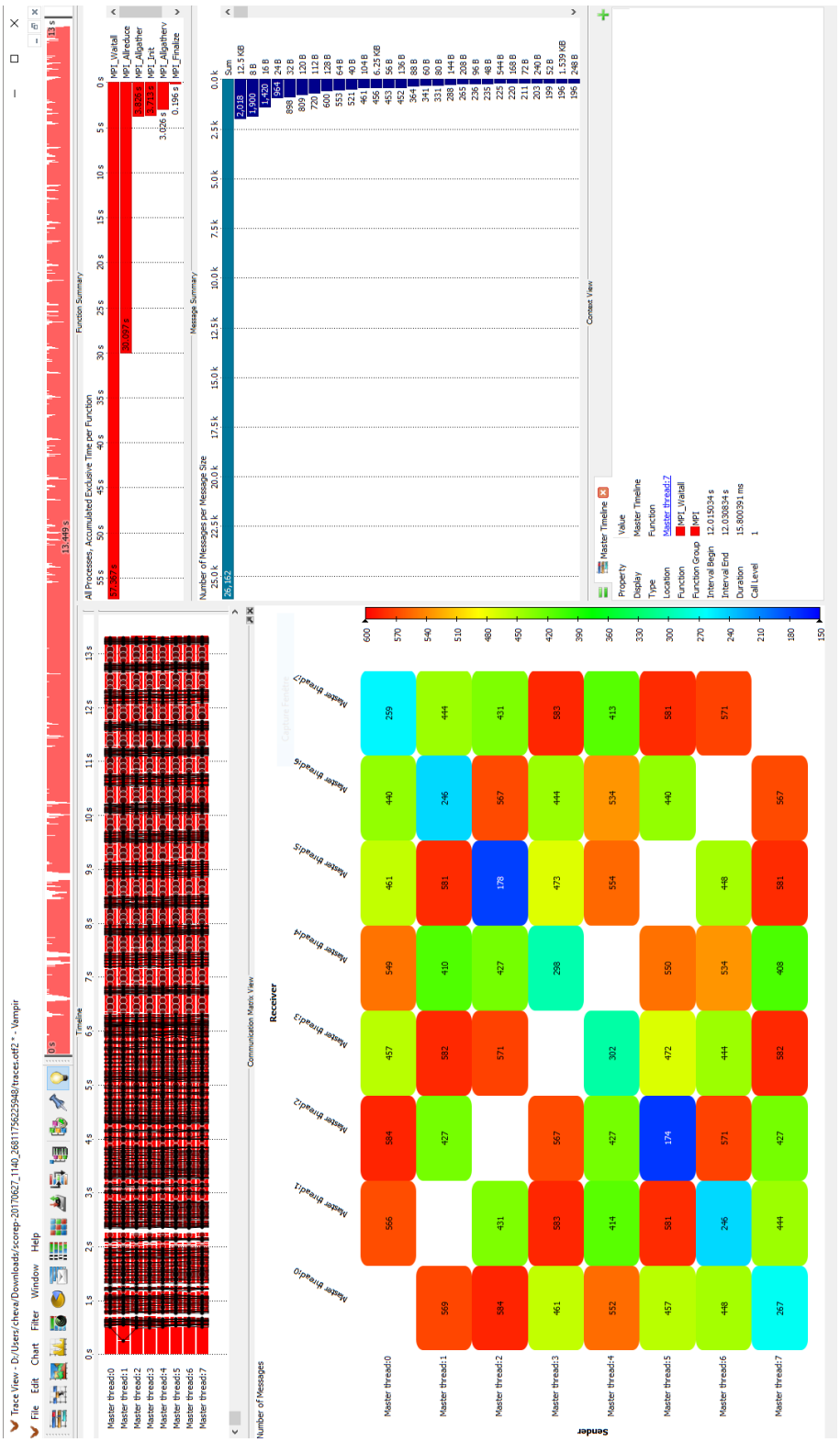

Figure 12 Capture d'écran Vampir

La bande supérieure représente la durée de la simulation, avec en rouge l'utilisation de MPI par les processus. Le rectangle supérieur gauche représente la trace d'exécution de chaque processus, avec l'utilisation de MPI en rouge et les interactions en noir. Le rectangle inférieur gauche montre le nombre de messages envoyé d'un processus à l'autre. Cela peut permettre de savoir où placer les nœuds dans un cluster afin de facilité la communication des nœuds qui communiquent beaucoup. Sur la partie droite on observe deux autres graphiques. Le premier montre le temps total passé dans chaque fonction MPI. Le second montre la taille des messages échangés.

3) TABLEAU DE BORD GITHUB

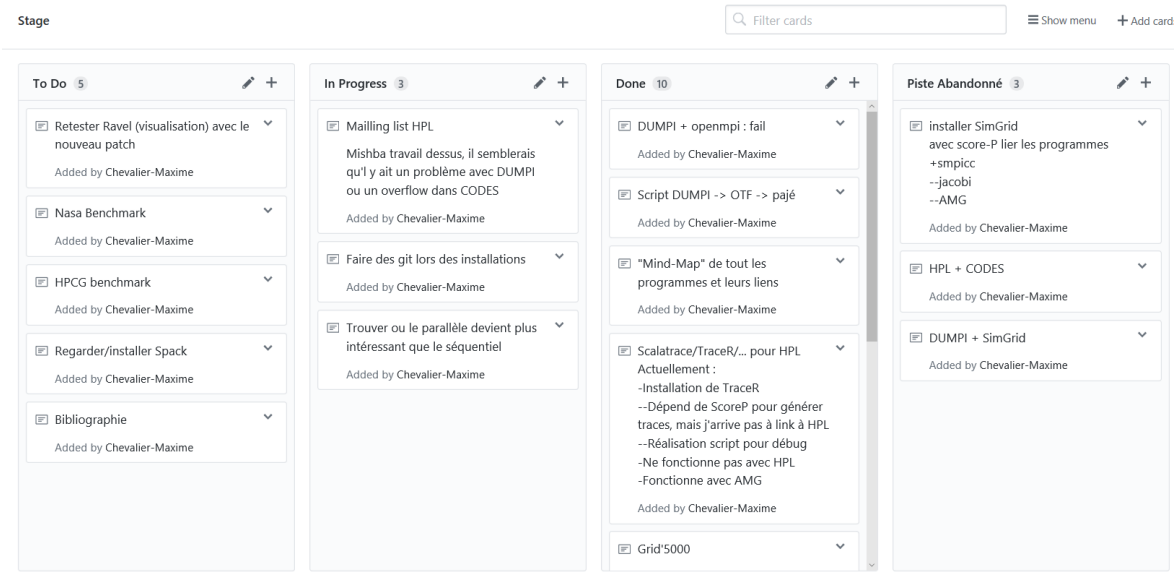

Figure 13 Tableau de bord GitHub utilisé pour l'organisation

4) GITHUB DU STAGE

Le journal et les scripts utilisés lors du stage sont disponibles à l'adresse suivante https://github.com/Chevalier-Maxime/Stage-LIG-2017

Etudiant : Chevalier Maxime **Année d'étude dans la spécialité :** 4

Entreprise : Laboratoire Informatique de Grenoble

Adresse complète :

UMR 5217 - Laboratoire LIG - Bâtiment IMAG - 700 avenue Centrale - Domaine Universitaire de Saint-Martin-d'Hères

Adresse postale : CS 40700 - 38058 Grenoble cedex 9 - France

Téléphone : 04 57 42 14 00

Responsable administratif : Simon Annie, Assistante d'équipes de recherche

Téléphone : 04 57 42 15 77

Courriel : Annie.Simon@inria.fr

Tuteur de stage : Perronnin Florence

Téléphone : 04 57 42 15 42

Courriel : florence.perronnin@imag.fr

Enseignant-référent : Boyer Fabienne

Téléphone : 04 76 63 57 93

Courriel : Fabienne.Boyer@imag.fr

Titre : (maximum 2 à 3 lignes).

Résumé :

De nos jours, les supercalculateurs sont de plus en plus performants, de plus en plus complexe et nécessite un investissement important. Afin d'aider les experts à les élaborer, des chercheurs ont développé des simulateurs permettant d'exécuter des applications parallèles utilisant la technologie MPI. Ces simulateurs sont capables d'émuler des réseaux complexes (Fat-Tree, Dragonfly, …) à très grande échelle, dignes du top500 des plus grands supercalculateurs actuels. Ils permettent outre la simulation de la topologie réseau, de simuler les CPU, les GPU, les différents niveaux de mémoire, … afin d'avoir une simulation fine. Ces simulateurs permettront donc, dans le futur d'élaborer facilement un supercalculateur et de le dimensionner aux applications pour lequel on le crée. Cependant, de nombreux simulateurs coexistent actuellement et utilisent différentes techniques de simulation d'application MPI. Nous comparerons deux d'entre eux, les simulateurs CODES et SimGrid, mais aussi deux manières différentes de simuler, la simulation séquentielle et la simulation parallèle optimiste.

Summary :

Nowadays, supercomputers are increasingly efficient, complex and more expensive. In order to help experts to size them, researchers are developing simulators able to run a parallel application using MPI technology. Theses simulators run these applications in a virtual tuned environment. They can easily simulate the current best supercomputer of the top500 list. It is possible to choose the network topology (Fat-Tree, Dragonfly, P2P, …), to describe different components as CPU, GPU, link capacity, different data level size and respond time, emulation of traffic network, … This tidy granularity allows the easy build of targeted supercomputer. After simulation, it is possible to observe bottleneck, … and overall performances thanks to other tools like TAU (a performance analyser) or visualization tools like Vampir. However, several simulators and several ways to simulate MPI applications exist. So, we will compare two of them, SimGrid and CODES, and also two ways of simulation: sequential and optimistic