Opérations collectives dynamiques dans StarPU / NewMadeleine

Philippe Swartvagher

Opérations collectives dynamiques dans StarPU / NewMadeleine

Stage de fin d'études du cursus Ingénieur - Informatique

Philippe Swartvagher

Maître de stage :

Alexandre Denis

Responsable pédagogique :

Mathieu Faverge

29 août 2019

Remerciements

Je souhaiterais vivement remercier toutes les personnes qui ont contribué au bon déroulement de mon stage.

Un grand merci donc à Alexandre Denis, mon maître de stage, qui m'a fait découvrir le monde du calcul distribué, qui m'a guidé tout au long de ce stage et qui a su répondre à la moindre de mes (nombreuses !) interrogations. Merci à Nathalie Furmento et Samuel Thibault pour leurs réponses précises à toutes mes questions techniques. Merci à toute l'équipe TADaaM et en particulier aux occupants de l'open-space, où a toujours régné une ambiance chaleureuse.

Sommaire

Introduction

Ce stage de six mois marque la dernière étape de ma formation d'ingénieur en informatique. Après avoir suivi un semestre spécialisé dans la cyber-sécurité, le choix de mon stage s'est porté vers une thématique bien différente : le calcul haute performance.

Plusieurs raisons m'ont poussé à explorer un domaine différent de ma spécialité. Tout d'abord, le calcul haute performance m'intéresse tout autant que la cyber-sécurité. Faire un stage en calcul haute performance me permet d'explorer également ce domaine, et ainsi étoffer mon profil d'ingénieur en informatique. Ensuite, ce stage propose une poursuite en thèse sur le même sujet. Faire une thèse est une orientation professionnelle qui m'attire depuis quelques années, pour pouvoir analyser en profondeur une problématique, le tout en suivant une démarche expérimentale. Mon stage se déroulant à l'Inria, laboratoire de recherche en informatique, cela permet d'entrer dans le monde de la recherche scientifique et de découvrir les problématiques qui occupent actuellement les chercheurs.

Le sujet du stage consiste à améliorer la façon dont un nœud de calcul envoie les mêmes données à plusieurs autres nœuds de calcul. Au lieu que ce soit un unique nœud qui se charge d'envoyer la donnée à tous les nœuds, les nœuds qui ont déjà reçu la donnée peuvent également se charger de l'envoyer à des nœuds qui doivent encore la recevoir. Cela permet de soulager la charge d'envoi du nœud initialement émetteur de la donnée, augmenter la vitesse de propagation de la donnée et ainsi espérer un meilleur passage à l'échelle. Cet algorithme pour diffuser une donnée à plusieurs destinataires n'est pas nouveau, cependant sa mise en place au sein de bibliothèques logicielles développées à l'Inria, de façon à ce qu'il ne soit pas nécessaire d'attendre que tous les nœuds destinataires soient prêts à recevoir la donnée pour commencer à l'envoyer, est le principal apport de ce stage. Des évaluations des performances ont également été réalisées.

Ce rapport décrit le travail réalisé durant ce stage. Une première partie pose le cadre du stage : elle présente l'Inria, donne un aperçu du domaine du calcul haute performance, introduit les bibliothèques logicielles utilisées et précise les objectifs du stage. La deuxième partie rentre dans les détails de l'implémentation. La dernière partie explique les performances obtenues et la façon dont elles ont été mesurées.

1 Contexte du stage

J'ai effectué mon stage à Inria Bordeaux Sud-Ouest du 4 février au 26 juillet 2019.

Cette partie vise à présenter l'organisme et l'équipe qui m'ont accueilli, le domaine du calcul haute performance, les bibliothèques logicielles avec lesquelles j'ai travaillées et finalement l'objectif de ce stage.

1.1 Présentation de l'Inria

L'Inria (Institut National de Recherche en Informatique et Automatique) est un institut de recherche en mathématiques et informatique. Il se compose de 200 équipes de recherche, réparties dans 8 centres à travers toute la France.

Le centre de Bordeaux Sud-Ouest se compose d'équipes s'intéressant principalement au calcul haute performance, à la simulation numérique, à la robotique et à la santé.

L'équipe qui m'accueille, TADaaM (Topology-Aware System-Scale Data Management for High-Performance Computing Applications), a comme objectif la conception d'une couche logicielle qui permet l'abstraction des caractéristiques matérielles du système (ressources disponibles, topologies mémoire et réseau...) et des besoins logiciels des applications exécutées (threads, données...). Cette couche logicielle permet également l'optimisation de l'exécution des applications en les répartissant au mieux sur tout le système de calcul. Les sujets abordés sont à la fois théoriques (ordonnancements optimaux, prédiction du flot d'exécution des programmes...) et pratiques (développement de bibliothèques de communications réseau, analyse des caractéristiques matérielles des machines pour améliorer la vitesse d'exécution des logiciels...).

L'équipe est composée de 6 chercheurs permanents, 4 doctorants, 1 ingénieur, 1 alternant et 2 stagiaires ingénieurs.

Mon stage se déroule en collaboration avec l'équipe Storm, qui s'intéresse à la compilation, aux supports d'exécution et à l'analyse d'applications de calcul haute performance.

1.2 Le calcul haute performance

Le calcul haute performance (souvent abrégé *HPC* pour *High Performance Computing*) consiste à utiliser des ordinateurs pour effectuer d'importants calculs numériques. Les ordinateurs utilisés pour le HPC (aussi appelés *super-calculateurs*) ont une puissance qui est principalement limitée par les technologies du moment. L'objectif avec ces ordinateurs est d'atteindre la vitesse de calcul la plus importante possible.

Le HPC est principalement utilisé pour du calcul scientifique, et notamment des simulations numériques : prévisions météorologiques, climatologie, simulations nucléaires, mécanique, physique, aérodynamisme, chimie, *etc.*

Pour maximiser la puissance de calcul d'un ordinateur, il est possible d'améliorer à la fois le matériel électronique (processeurs, mémoires, interfaces réseau...) et les logiciels utilisés (systèmes

d'exploitation, bibliothèques, algorithmes...). Lorsque les limites des technologies actuelles sont atteintes, plusieurs de ces ordinateurs sont connectés entre eux pour former un super-calculateur. Chaque ordinateur est alors appelé un **nœud**. Les calculs requis par les programmes sont ensuite répartis sur ces nœuds.

De nos jours, chaque nœud possède plusieurs processeurs physiques (CPU), et même parfois des processeurs graphiques (GPU). De part leur architecture, ces derniers sont bien plus performants que les CPU pour exécuter parallèlement des instructions, mais leurs performances s'effondrent dès qu'il s'agit d'exécuter des instructions de contrôle (conditions, *etc*).

Concernant la mémoire vive, elle peut être **partagée** ou **distribuée**. Elle est partagée lorsque plusieurs processeurs d'un même nœud se partagent la mémoire de ce nœud ; elle est distribuée lorsqu'elle est répartie sur plusieurs nœuds.

Les communications entre nœuds doivent également se faire le plus rapidement possible. C'est pourquoi des interfaces réseaux spécifiques au HPC ont été développées : elles sont bien plus rapides que les interfaces ordinaires et ont une latence très faible. L'acteur majeur dans ce milieu est Infini Band [1]. À titre de comparaison, la connexion réseau Ethernet d'un ordinateur classique a une bande passante de 1 Gbit/s avec une latence de plusieurs dizaines de microsecondes, alors que Infini Band propose des bandes passantes de l'ordre de la centaine de Gbit/s et une latence inférieure à la microseconde. Ce type d'interface réseau a également la particularité de se programmer en espace utilisateur, pour éviter de parcourir la pile réseau du noyau et gagner le temps nécessaire à un appel système.

La puissance des super-calculateurs se mesure en **flops** (*Floating-point operations per second*, ou *opérations en virgule flottante par seconde*). Summit, le super-calculateur le plus puissant au monde selon le classement Top500 [2] de novembre 2018, visible sur la Figure 1, a une puissance de calcul de 143 500 Tflops. Un ordinateur personnel a une puissance d'une quarantaine de Gflops.

Figure 1 – Le super-calculateur Summit. Carlos Jones / ORNL - WikiMedia

Pendant mon stage, j'ai eu accès aux plateformes PlaFRIM (Inria, Université de Bordeaux) et Curta (Mcia) pour mes expérimentations. PlaFRIM a une puissance de calcul d'environ 96

Tflops. Les nœuds que j'ai utilisés possédaient deux processeurs INTEL XEON avec 12 cœurs cadencés à 2,5 GHz, 128 Go de RAM, des interfaces ETHERNET à 10 Gbit/s et des interfaces INFINI BAND à 40 Gbit/s ou OMNI-PATH à 100 Gbit/s selon les nœuds.

Toutes ces caractéristiques particulières des ordinateurs dédiés au HPC (répartition des calculs sur de nombreux nœuds, communications rapides, vitesse de calcul, ...) imposent d'adapter la façon de concevoir les programmes qui vont s'exécuter sur ces super-calculateurs. Pour gagner le moindre temps de calcul, il est nécessaire d'optimiser au mieux les programmes, d'exploiter au maximum les possibilités du matériel (instructions des processeurs, optimisations des caches et de la localité des processus, réactivité des programmes...). Heureusement pour le développeur de telles applications, de nombreux standards et bibliothèques ont émergé pour faciliter l'écriture des programmes pour le HPC.

MPI (*Message Passing Interface*, [3]) est un de ces standards que j'ai rencontré durant ce stage. Cette norme définit un ensemble de prototypes de fonctions pour transmettre des données entre plusieurs ordinateurs. De nombreuses bibliothèques logicielles implémentent cette norme : il est possible de citer OPENMPI [4] (à ne pas confondre avec OPENMP [5], qui est un ensemble de directives pour compilateurs pour indiquer comment paralléliser des portions de code), MPICH [6], MVAPICH [7], NEWMADELEINE [8] et MPC [9].

1.3 StarPU et NewMadeleine

STARPU et NEWMADELEINE sont deux bibliothèques logicielles qui facilitent le développement de logiciels HPC.

STARPU est un ordonnanceur de tâches pour des machines aux architectures hétérogènes [10]. Chaque architecture ayant souvent un modèle de programmation et une API (*Application Programming Interface*) qui lui sont propres, il est difficile d'utiliser simultanément toutes les unités de calcul d'un ordinateur. Même si les algorithmes sont écrits pour toutes les unités de calcul, il reste ensuite à répartir l'exécution du programme sur les différentes unités de calcul et gérer les transferts mémoires des données, pour ne citer que les deux aspects les plus difficiles. STARPU s'attaque justement à ces derniers. Le développeur d'une application basée sur STARPU décompose ses algorithmes en une suite de tâches. Pour chaque tâche, il fournit le code des algorithmes pour chaque architecture ciblée. Ces tâches et leurs codes sont ensuite fournis à STARPU, ainsi que le graphe de dépendances des tâches : elles peuvent utiliser comme données d'entrée les résultats d'autres tâches. STARPU va ensuite se charger d'ordonnancer ces tâches, de les répartir sur les différentes unités de calcul disponibles, de gérer leur mémoire et de les exécuter. Les communications réseaux sont déléguées à une autre bibliothèque, qui implémente l'interface MPI. STARPU est développé par l'équipe STORM.

NEWMADELEINE est développé par l'équipe TADAAM. Il s'agit d'une bibliothèque de communication supportant de multiples interfaces (mémoire partagée ou distribuée, interfaces réseaux spécifiques au HPC...), qui assure la progression asynchrone des communications en tâche de fond. Ses bonnes performances viennent du fait qu'elle applique des stratégies d'optimisation sur les flux provenant de plusieurs threads [11]. NEWMADELEINE implémente en plus l'interface MPI pour être compatible avec toutes les bibliothèques qui s'appuient sur cette interface pour leurs

communications.

StarPU a des besoins en terme de communication qui ne sont pas complètement compatibles avec le standard MPI. MPI est adapté aux applications qui alternent les phases de calcul et de communications réseau de façon synchronisée entre tous les noeuds. Les instances de StarPU lancées sur différents nœuds s'exécutent indépendamment les unes des autres, elles ne sont donc pas synchronisées entre elles. Une instance sur un nœud peut choisir un ordonnancement des tâches différent de celle sur un autre nœud, avoir des tâches qui ne s'exécutent pas à la même vitesse... StarPU a donc un modèle de communication assez irrégulier et a, par conséquence, besoin de réactivité vis-à-vis des communications. Ces irrégularités peuvent provoquer un grand nombre de requêtes actives simultanément, à tel point que les bibliothèques MPI ont souvent du mal à suivre en terme de performances. De même, StarPU tire beaucoup profit du multi-threading, modèle d'exécution souvent peu supporté par les bibliothèques MPI. NewMadeleine répond à ce cahier des charges sans problème, puisqu'il a été conçu pour faire du multi-threading, montre de très bonnes performances lorsque le nombre de requêtes croît et peut fonctionner de façon asynchrone. Ses mécanismes internes lui confèrent une grande réactivité vis-à-vis des requêtes entrantes ou sortantes. C'est pourquoi une version de StarPU a été développée avec une intégration directe des fonctionnalités de NewMadeleine [12], sans passer par l'interface MPI.

StarPU et NewMadeleine sont écrites en C, diffusées avec une licence libre et font l'objet de nombreuses publications scientifiques de la part de leurs équipes respectives.

1.4 Communications collectives

Les communications réseau sont l'un des goulots d'étranglement pour les performances des applications HPC. Mon stage s'intéresse à un type de communications réseau : les communications collectives ; et en particulier lorsqu'un message émanant d'un unique nœud doit parvenir à plusieurs destinataires (mécanisme aussi appelé *multicast* dans le jargon des réseaux, ou même *broadcast* par abus de langage venant du standard MPI). Ce type de communication montre rapidement des problèmes de performances lorsque le nombre de destinataires augmente.

Lors de l'exécution d'un programme, StarPU peut s'apercevoir que plusieurs tâches réparties sur des nœuds différents ont besoin de la même donnée d'entrée, présente sur un unique nœud. Dans ce cas, StarPU envoie la donnée à tous les nœuds qui en ont besoin pour exécuter leur tâche : cela constitue une communication collective. Nombreux sont les algorithmes de calculs mathématiques qui doivent transmettre des résultats intermédiaires à plusieurs sous-calculs.

Plusieurs algorithmes de routage existent pour transmettre une donnée d'un nœud vers plusieurs nœuds (une liste en est donnée dans [13], partie 2.1.1). Le plus naïf, illustré par la Figure 2, consiste à ce que le nœud source envoie séquentiellement les données vers les nœuds destinataires. Cet algorithme a une complexité linéaire en nombre de nœuds destinataires. Pendant mon stage, nous nous sommes intéressés à l'algorithme utilisant des arbres binomiaux (voir Figure 3) : les nœuds qui ont déjà reçu les données, se transforment en nœuds source pour les nœuds qui n'ont pas encore reçu les données, et ce, récursivement (sur la Figure 3 : 0 commence par envoyer à 4 ; ensuite pendant que 0 envoie à 2, 4 envoie à 6 ; puis pendant que 0 envoie à 1, 2 envoie à 3 et 4 envoie à 5). Cet algorithme a une complexité meilleure, puisque logarithmique en nombre de

nœuds destinataires.

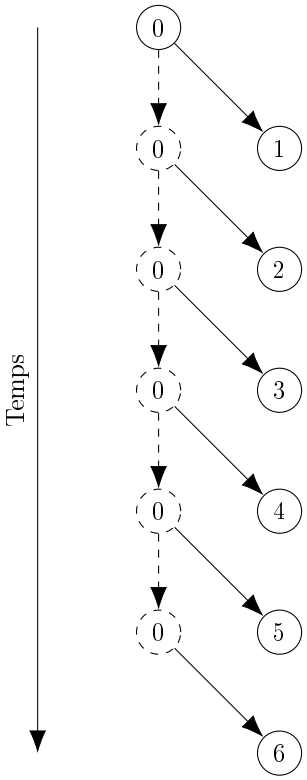


FIGURE 2 – Algorithme naïf de collective

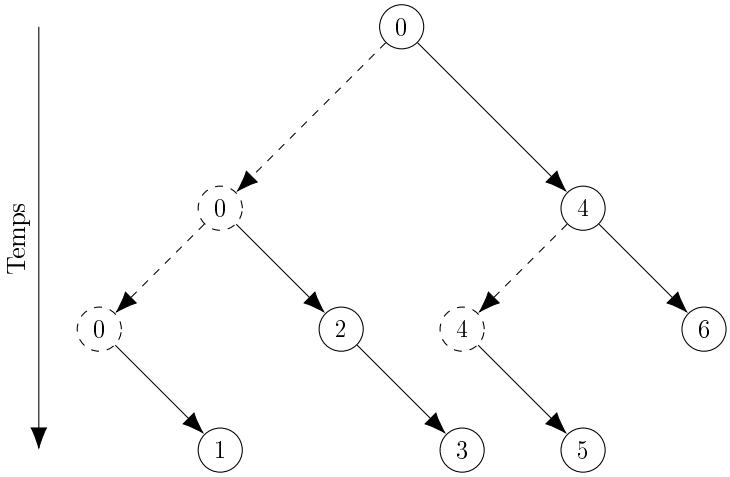


FIGURE 3 – Algorithme de collective avec arbre binomial

Le choix de l'algorithme de routage le plus adapté pour une collective dépend de nombreux paramètres, dont le nombre de nœuds destinataires, la taille des données à transmettre, les caractéristiques du réseau, *etc.* Durant ce stage, le choix s'est porté sur celui avec l'arbre binomial, car c'est celui qui devrait offrir les meilleurs compromis, *i.e.* améliorer les performances des collectives quels que soient le nombre de nœuds destinataires et la taille des données.

Avec le standard MPI, il est nécessaire pour établir ce genre de transmission de faire participer simultanément à la communication tous les nœuds impliqués, et que tous ces nœuds se connaissent les uns les autres. Ces contraintes sont nécessaires pour que chaque nœud puisse exécuter le même algorithme de routage, pour ainsi pouvoir se situer dans l'arbre de diffusion, et savoir à quels nœuds encore transmettre les données. Pour StarPU, puisqu'il n'y a pas de synchronisation entre les nœuds, attendre que tous les nœuds de la collective soient prêts à exécuter la collective s'avère très mauvais pour ses performances.

1.5 Objectifs du stage

Avant mon stage, StarPU détectait les communications collectives et appliquait ensuite l'algorithme naïf. Pour profiter de la complexité intéressante de l'algorithme binomial, mais sans perdre en performances, il a été proposé de faire des **collectives dynamiques** : les nœuds se rendent

compte qu'ils sont impliqués dans une collective seulement au moment où ils reçoivent des données. Cela évite de devoir préparer en amont la collective avec tous les nœuds et de ralentir ainsi l'exécution du programme. De plus, la collective est gérée par StarPU : c'est lui qui décide s'il faut utiliser une collective et c'est lui qui gère la réception s'il s'avère qu'il s'agit d'une collective. Ainsi, tout est fait de façon transparente pour l'utilisateur.

Le stage consistait à implémenter ce mécanisme de diffusion directement dans StarPU en utilisant l'API proposée par NewMadeleine. La suite de ce rapport de stage présente plus en détail le fonctionnement de l'implémentation, les tests réalisés et les résultats obtenus en terme de performances.

2 Travail réalisé

Durant ces six mois de stage, mon travail s'est décomposé en plusieurs parties :
- une première période a été nécessaire pour appréhender le sujet et se familiariser avec l'environnement ;
- j'ai ensuite construit uniquement avec NewMadeleine l'algorithme général et les structures de données nécessaires à l'implémentation des communications collectives ;
- cet algorithme a été intégré à StarPU, en respectant les contraintes de cette bibliothèque et en s'adaptant à son fonctionnement ;
- de nombreux tests ont été conçus pour pouvoir s'assurer du bon fonctionnement de différents cas d'utilisation possibles ;
- des tests de performances ont été menés pour mesurer l'impact de l'utilisation d'arbres de diffusion pour transmettre les données.

Cette partie se concentre sur la description du fonctionnement général de l'algorithme dans son implémentation dans NewMadeleine et son intégration à StarPU. Une sous-partie présente un problème rencontré, celui de la synchronisation des horloges, que je considère intéressant à détailler.

2.1 Mécanisme général

Dans un programme utilisant NewMadeleine, l'envoi des données se fait en précisant un certain nombre de paramètres :
- les données à envoyer ;
- le nœud destinataire ;
- le tag du message : étiquette qui permettra au nœud destinataire de catégoriser le message selon les besoins applicatifs et de ne pas mélanger les messages lorsque plusieurs messages arrivent simultanément.

La réception des données se fait presque avec les mêmes paramètres :
- où seront stockées les données reçues ;
- de quel nœud sont attendues les données (avec StarPU il n'est pas possible de dire que les données proviennent de n'importe quel nœud) ;

— le tag du message attendu.

L'envoi et la réception peuvent se faire de façon bloquante (la fonction d'envoi ne retourne qu'une fois l'envoi terminé et la fonction de réception une fois que les données sont reçues) ou non-bloquante : les fonctions retournent tout de suite, il est alors possible de renseigner une fonction à exécuter lorsque leur objectif est accompli.

Ces fonctions sont surchargées par StarPU pour pouvoir les proposer au développeur utilisant StarPU ou pour s'en servir en interne, si l'exécution d'une tâche nécessite des données non-disponibles sur son nœud d'exécution.

La mise en place des communications collectives avec des arbres binomiaux est facilitée avec NewMadeleine, puisque c'est une bibliothèque qui se repose sur un mécanisme de programmation événementielle : l'expression de l'algorithme en est simplifiée et cela le rend plus efficace que s'il avait fallu l'implémenter dans d'autres bibliothèques de communication non événementielles. En effet, NewMadeleine est capable de faire progresser les communications en tâches de fond, tout en réagissant aux événements provenant du réseau. Il est ainsi plus facile d'exécuter les communications collectives en arrière-plan, sans impliquer l'application utilisant NewMadeleine.

Avant de pouvoir utiliser un algorithme adapté aux collectives, il faut pouvoir détecter les collectives. Ensuite, concernant les envois, l'idée générale consiste à envoyer avec les données la liste des nœuds auxquels le nœud destinataire devra se charger de faire parvenir les données. Pour les réceptions, une requête en réception globale est postée pour recevoir toutes les collectives, faire suivre les données aux autres nœuds et finalement rediriger les données vers la requête postée par l'utilisateur. Tous ces éléments sont détaillés dans la suite de cette partie.

2.1.1 Détecter les communications collectives

La difficulté pour lancer le mécanisme de communication collective est de détecter qu'une même donnée va être envoyée à plusieurs nœuds. Il est en effet difficile de savoir avant l'exécution de StarPU si un envoi collectif sera nécessaire, puisque c'est en ordonnançant les tâches à exécuter que StarPU crée des communications, éventuellement collectives. De plus, l'ordonnancement des tâches peut changer au cours du fonctionnement de StarPU, ce qui rend les communications collectives encore moins prédictibles. Cette détection à la volée des collectives rend impossible l'indication explicite de communications collectives. Ce mécanisme est donc transparent pour l'utilisateur.

Lorsqu'un envoi est demandé par StarPU, il est possible que la donnée à envoyer ne soit pas encore disponible (parce que l'exécution d'une tâche produisant cette donnée est encore en cours, par exemple). Si, pendant le temps d'attente que la donnée devienne disponible, d'autres envois de cette donnée vers d'autres nœuds sont demandés, alors les nœuds destinataires sont stockés dans une liste de destinataires de la première requête d'envoi postée. C'est actuellement la seule façon dont StarPU détecte les communications collectives.

Ce mécanisme de détection des communications collectives pose encore question, car il n'est pas forcement optimal : peut-être qu'il est préférable d'attendre un peu entre le moment quand la donnée devient disponible et celui où la donnée va être envoyée, pour s'assurer qu'aucune autre requête d'envoi n'est postée. En effet, peut-être que le temps d'attente supplémentaire sera com-

pensé par le temps gagné grâce à la diffusion en arbre. Être sûr que StarPU détecte bien les opérations collectives fait partie des éléments qui seront testés plus tard.

Lorsque la donnée devient disponible, StarPU procède à l'envoi : si un unique nœud se trouve dans la liste des destinataires, alors la donnée est simplement envoyée vers le destinataire. En revanche, si cette liste possède plusieurs destinataires, un arbre de diffusion est mis en place. StarPU passe alors la main à NewMadeleine.

2.1.2 Construire un arbre de diffusion et envoyer les données

La donnée doit être envoyée à plusieurs nœuds.

Des fonctions sont disponibles pour construire l'arbre binomial correspondant à l'envoi aux nœuds destinataires. Ces fonctions permettent de savoir vers quels nœuds chaque nœud doit envoyer la donnée, pour que lorsque tous les envois soient faits, tous les destinataires aient bien reçu la donnée.

Le nœud initiant l'envoi peut donc savoir vers quels nœuds il doit réellement envoyer la donnée et vers quels nœuds ces nœuds devront faire suivre la donnée. Par exemple, l'arbre de la Figure 3, permet au nœud 0 de savoir :

— qu'il doit envoyer la donnée au nœud 4. Le nœud 4 devra ensuite transmettre la donnée aux nœuds 5 et 6.
— qu'il doit envoyer la donnée au nœud 2. Le nœud 2 devra ensuite transmettre la donnée au nœud 3.
— qu'il doit envoyer la donnée au nœud 1.

Finalement, le nœud 0 n'envoie la donnée qu'aux nœuds 1, 2, 4, ce qui est bien moins que les 6 nœuds avec un envoi séquentiel.

L'envoi des données se fait en ajoutant la liste des nœuds auxquels le nœud destinataire devra faire suivre les données. Un bit est activé dans le tag pour indiquer que cette requête fait partie d'une communication collective. De cette façon, lorsque la donnée arrivera au destinataire, elle ne sera pas traitée par la requête postée par l'utilisateur, mais par une requête propre à NewMadeleine qui se chargera de faire suivre les données à d'autres nœuds si cela s'avère nécessaire.

Devoir passer par un canal spécifique aux communications collectives est nécessaire pour trois raisons. La première, déjà évoquée, est que ce type de requête peut nécessiter un traitement spécifique s'il faut faire suivre les données. La deuxième, c'est que la structure des données est différente que celle à laquelle s'attend l'utilisateur : les données transmises contiennent en plus les données relatives à la communication collective. Finalement, dans tous les cas, il est impossible de recevoir les données directement sur la demande de réception qu'a postée l'utilisateur, car les données peuvent provenir d'un nœud différent de celui spécifié par l'utilisateur (dans l'exemple de la Figure 3, le nœud 3 recevra la donnée qu'il attend du nœud 0 par le nœud 2).

2.1.3 Recevoir (et faire suivre) les données

Tous les nœuds sont susceptibles de recevoir des données d'une communication collective, donc avec le tag spécifique. Puisque les communications collectives sont un mécanisme caché de l'uti-

lisateur, ce tag n'est pas accessible par l'utilisateur, il n'en a pas connaissance. Pour recevoir les données provenant de communications collectives, NewMadeleine poste en interne, lors de son démarrage, une requête en réception sur ce tag avec tous les nœuds comme origine possible.

Lorsqu'un message est reçu sur ce canal, les données relatives à la communication collective sont lues par NewMadeleine. S'il est nécessaire de le transmettre à d'autres nœuds, le même mécanisme est mis en place que lorsqu'un nœud envoie initialement des données : un arbre de diffusion est à nouveau construit avec les nœuds auxquels il faut transmettre les données.

Une fois que les données ont été transmises aux autres nœuds si cela était nécessaire, NewMadeleine notifie la requête en réception postée par l'utilisateur que les données sont arrivées ; par extension, cela permet d'avertir StarPU que les données sont arrivées. Pour savoir à quelle requête de l'utilisateur les données correspondent, chaque requête en réception postée par l'utilisateur est stockée dans une table de hachage avec comme clé le nœud source et le tag. Retrouver les requêtes initialement postées par l'utilisateur se fait donc en temps constant. Si la requête postée par l'utilisateur n'est pas trouvée (elle n'est pas dans la table de hachage), la requête collective qui vient d'arriver est mise en attente jusqu'à ce que l'utilisateur poste la requête correspondante.

Utiliser ainsi une table de hachage soulève deux problèmes. Le premier est qu'il est ainsi impossible d'avoir en même temps deux requêtes en réception qui attendent des données provenant d'un même nœud avec le même tag. Deuxièmement, ces algorithmes font doublon, puisque c'est ainsi que fonctionne en interne NewMadeleine pour gérer les requêtes, et ce, de manière très efficace [14]. À terme, il serait pertinent de pouvoir interagir directement avec les mécanismes internes de NewMadeleine pour retrouver la requête correspondant à un couple nœud/tag et éviter d'utiliser une sur-couche propre aux communications collectives. Pour le moment, cette façon de faire est suffisante pour réaliser des tests sur les communications collectives.

2.2 Synchronisation d'horloges

Pour faciliter le débogage d'applications parallèles et la visualisation de ce qui s'est passé pendant l'exécution, il est possible de lancer les applications qui utilisent StarPU avec un mode qui va générer des **traces**. Ces traces sont des fichiers qui rassemblent des informations sur l'exécution du programme, comme les transferts réseau et mémoire, les différents processus et threads et leurs états, *etc.* Il est ensuite possible de visualiser ces traces à l'aide du programme Vite [15]. La Figure 4 donne un exemple de la visualisation qu'il est possible d'obtenir. Les flèches blanches représentent les transferts de données entre nœuds. Ce sont elles qui vont principalement nous intéresser, puisqu'il s'agit d'un moyen de visualiser comment se déroulent les communications collectives. Précision importante, le temps est situé sur l'axe des abscisses.

2.2.1 Présentation du problème

Après avoir manipulé les traces de nombreuses exécutions, il s'est avéré que sur certaines d'entre elles, les communications allaient vers le passé (les flèches allaient vers la gauche), comme l'illustre la Figure 5.

Cette incohérence a permis de mettre en évidence un problème de synchronisation des horloges.

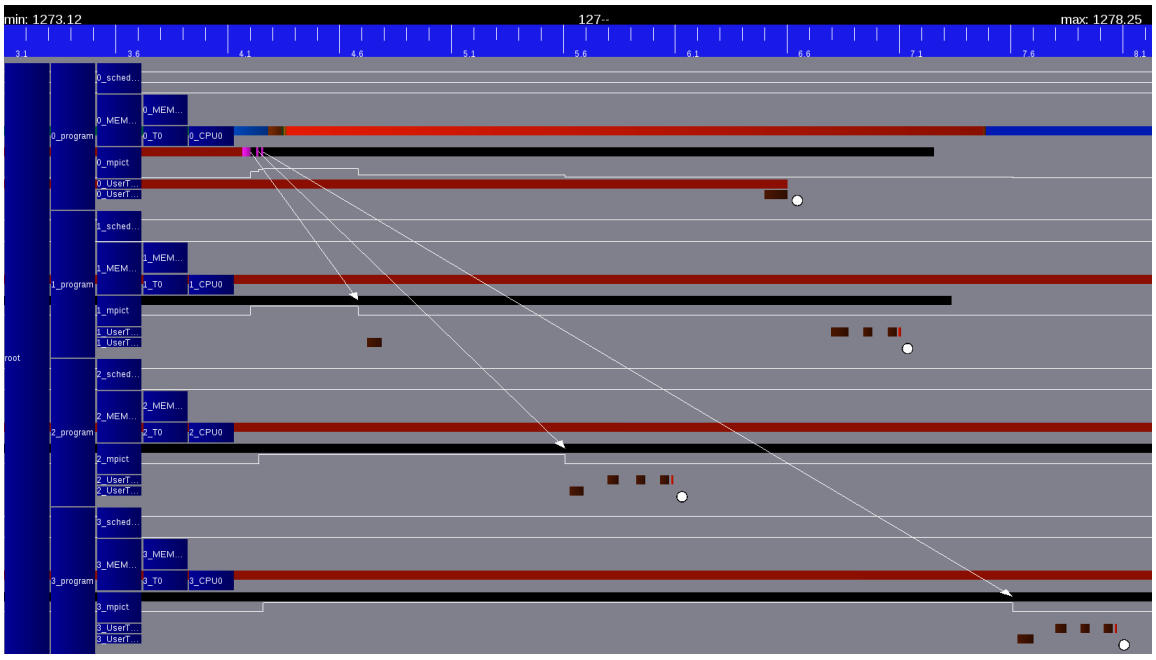


FIGURE 4 – Visualisation avec VITE de la trace d'un envoi séquentiel de données vers 3 nœuds.

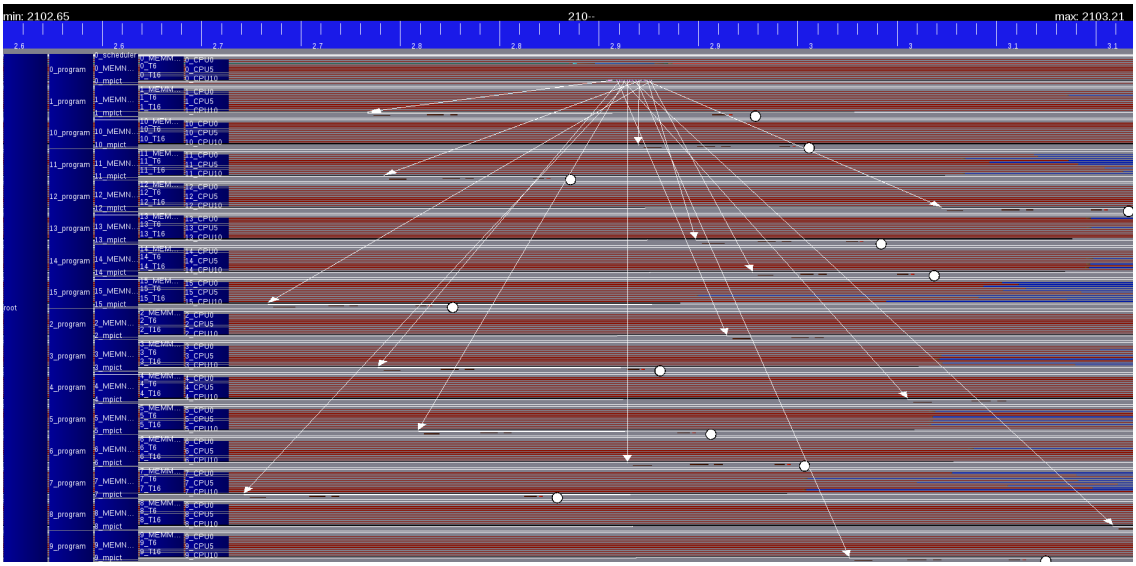


FIGURE 5 – Trace boguée de l'envoi séquentiel vers 15 nœuds.

Lorsque des événements provenant de différents nœuds sont sauvegardés, il faut s'assurer que les horloges qui donnent le moment d'apparition des événements aient bien la même origine (l'origine de chaque horloge peut être le démarrage du programme sur chaque nœud, par exemple). Les horloges sont dites *synchronisées* lorsqu'elles ont toutes la même origine. Si l'origine des temps n'est pas la même pour tous les événements, des erreurs comme celle visible sur la FIGURE 5 peuvent apparaître. Ce problème n'affecte pas seulement la visualisation des traces, puisque si les temps sont mal mesurés, il est également possible que cela fausse les tests de performances que nous ferons par la suite. À noter que la synchronisation d'horloges distribuées n'est pas un problème

évident et fait encore l'objet de recherches [16].

Dans ce cas, la synchronisation des horloges était faite en considérant que tous les nœuds sortaient d'une barrière MPI (permet de s'assurer que tous les nœuds sont dans une même portion de code à un moment donné : tous les nœuds envoient une donnée à un nœud, puis ce nœud répond à tous les autres nœuds) en même temps. Seulement, ce n'est pas nécessairement le cas, comme l'illustre la FIGURE 6. Le module qui génère les traces considérait que les temps t_1, t_2 et t_3 étaient égaux à t_0, d'où le décalage provoqué.

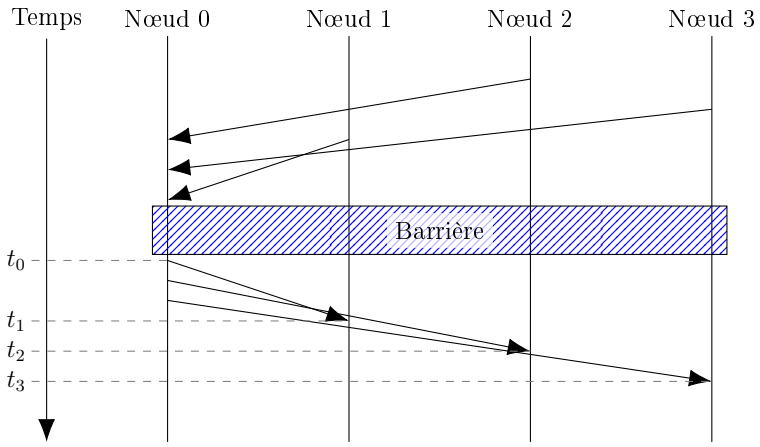


FIGURE 6 – Exemple de barrière : tous les nœuds ne sortent pas en même temps de la barrière.

2.2.2 Solution

La solution mise en œuvre pour synchroniser les horloges consiste à comparer les horloges locales de deux nœuds pour obtenir la différence et ajouter à cette différence la latence de communication entre ces nœuds. Prenons l'exemple de la synchronisation de l'horloge d'un nœud B sur celle d'un nœud A (voir FIGURE 7) :

1. le nœud A relève le temps (t_A) ;
2. A envoie un message à B ;
3. à la réception du message, B relève le temps (t_B), et envoie la valeur à A ;
4. en recevant t_B, A peut calculer la différence entre t_B et t_A. À cette valeur, il faudra également soustraire la latence de la communication entre A et B ;
5. A envoie à B la valeur de son décalage δ avec l'horloge de A.
6. B stocke la valeur de son décalage dans son fichier de trace.

Lors du pré-traitement du fichier de traces pour le visualiser, le décalage de chaque nœud sera appliqué sur les temps que contient sa trace. Il n'est pas possible d'appliquer le décalage lors de la génération des traces, car des informations de la trace sont déjà écrites, avant même que la synchronisation des horloges ait eu lieu.

Le calcul de la latence entre deux nœuds A et B se fait à l'aide de nombreux échanges (voir FIGURE 8) :

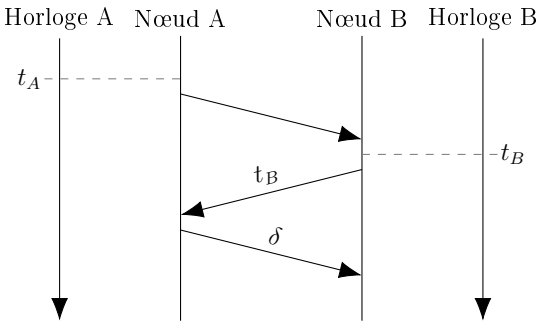


FIGURE 7 – Protocole pour calculer le décalage d'horloge entre A et B.

1. A relève le temps (t_1) ;
2. une boucle de N itérations a lieu dans laquelle A envoie un message à B et B lui répond ;
3. A relève le temps (t_2).

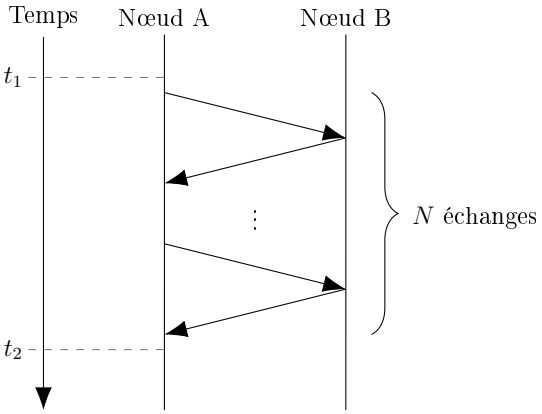


FIGURE 8 – Protocole pour calculer la latence entre A et B.

La latence entre A et B est donnée par la formule $\frac{t_2-t_1}{2N}$.

Pour vérifier que les horloges entre deux nœuds A et B sont bien synchronisées, il est possible d'exécuter le protocole suivant (illustré par la FIGURE 9) :

1. A relève le temps (t_{A1}) ;
2. A envoie un message à B ;
3. à la réception du message, B relève le temps, en appliquant le décalage calculé auparavant (t_B*) ;
4. B envoie à A un message avec la valeur de t_B* ;
5. à la réception du message, A relève le temps (t_{A2}) ;

Les horloges sont bien synchronisées si on a $t_{A1} < t_B* < t_{A2}$. Cela signifie que le décalage entre les horloges des nœuds A et B est inférieur à la latence du réseau, ce qui est suffisant pour nos besoins, puisque les traces nous servent à représenter des évènements réseau.

Les temps sont relevés à l'aide de la fonction `clock_gettime()`, utilisée avec le paramètre `CLOCK_MONOTONIC_RAW` pour ne pas avoir les ajustements NTP.

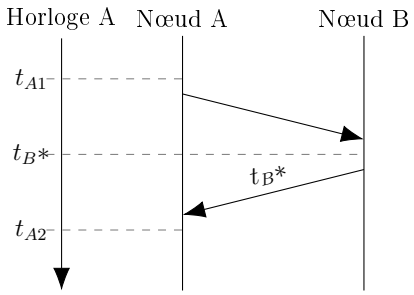


FIGURE 9 – Protocole pour s'assurer que les horloges de A et B sont bien synchronisées.

La solution présentée ne fait qu'aligner les origines des horloges. En revanche, elle ne prend pas en compte l'éventuelle différence de vitesse d'écoulement du temps sur les différents processeurs. Expérimentalement, il est possible de remarquer que la synchronisation n'est plus correcte après une trentaine de secondes.

3 Évaluation des performances

Une fois l'algorithme de communications collectives avec des arbres de diffusion implémenté, il a fallu réaliser des tests de performances pour s'assurer que le gain obtenu par cette nouvelle fonctionnalité était semblable aux prédictions théoriques.

L'efficacité de l'algorithme est principalement mesurée temporellement, même si d'autres métriques comme le volume de données envoyées par chaque nœud pourrait aussi être intéressant.

Cette partie présente les différents tests de performances réalisés et leur interprétation.

3.1 Microbenchmark NewMadeleine

Les premiers tests réalisés sont des *microbenchmarks* : ils mesurent les performances des collectives dynamiques seules, sans autre contexte. Le microbenchmark présenté ici n'utilise que NewMadeleine et réalise des transferts de 8 Mo de données en augmentant progressivement le nombre de nœuds dans la collective. Les temps mesurés correspondent à la durée de la collective : le temps entre le premier envoi de données et la réception par le dernier nœud. Après avoir synchronisé les horloges de tous les nœuds avec le nœud initiateur de la collective, chaque nœud renvoie au nœud source le moment où il a reçu les données. Le nœud source peut ainsi calculer quel est le dernier nœud à avoir reçu les données et connaître la durée totale de la collective.

Sur le graphe de la FIGURE 10, quatre courbes sont tracées : elles correspondent à quatre algorithmes de collectives différents. Nous nous intéressons principalement à la courbe bleue, représentant l'algorithme naïf et la courbe rouge, représentant les collectives dynamiques. Il est clair que l'algorithme naïf suit une complexité linéaire, tandis que l'algorithme des collectives dynamiques a une complexité logarithmique (les paliers correspondant aux changements de hauteur de l'arbre sont même visibles). Les implémentations de ces deux algorithmes sont donc conformes à la théorie.

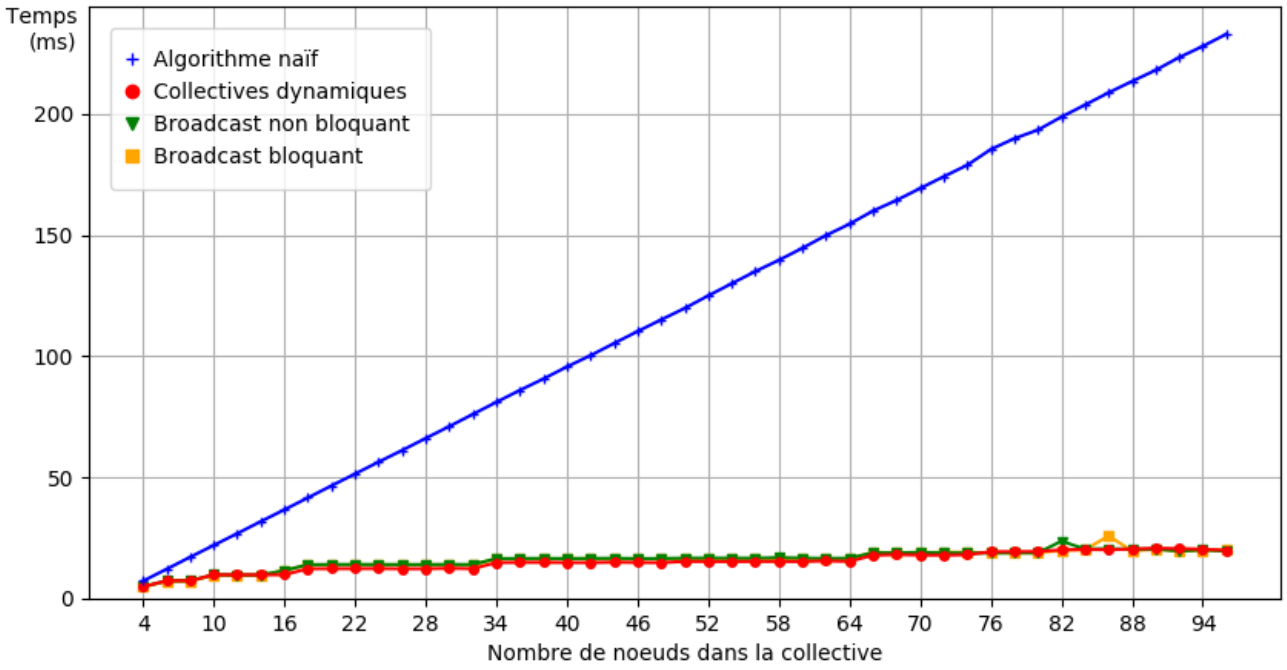


FIGURE 10 – Microbenchmark avec NEWMADELEINE, transmission de 8 Mo de données.

Les deux autres courbes, vertes et jaunes, représentent des algorithmes de collectives correspondant à ce qui se fait en MPI, où l'appel à une collective est explicite pour tous les nœuds de la collective : tous les nœuds savent qu'ils sont dans une collective avant même de commencer la collective et connaissent tous les nœuds qui y participent. La comparaison de l'algorithme des collectives dynamiques avec ces deux algorithmes « classiques » permet de savoir si les collectives dynamiques imposent un surcoût : le fait de transférer des données supplémentaires (la liste des nœuds auxquels il faut faire suivre) et de découvrir la collective au moment de recevoir les données pourrait être pénalisant. La comparaison est possible puisque ces trois algorithmes ont tous recourt à des arbres binomiaux et utilisent sensiblement les mêmes mécanismes internes de NEWMADELEINE. Finalement, il s'avère qu'aucun surcoût n'est mesuré, au contraire, l'algorithme des collectives dynamiques serait même parfois meilleur (voir FIGURE 11).

3.2 Décomposition de CHOLESKY

La décomposition de CHOLESKY est utilisée comme test applicatif. Cet algorithme est adapté car il génère un grand nombre de communications collectives, qui sont détaillées dans l'ANNEXE A. Pour tester cet algorithme, nous avons utilisé CHAMELEON [17], une bibliothèque d'algèbre linéaire développée par l'équipe HIEPACS de l'INRIA. Cette bibliothèque utilise STARPU et fournit déjà des tests pour évaluer les performances de l'algorithme de CHOLESKY.

La décomposition de CHOLESKY permet de déterminer pour toute matrice A symétrique définie positive, une matrice L triangulaire inférieure, telle que $A = LL^t$.

Nous nous intéressons dans notre cas à la factorisation par blocs : les matrices sont scindées en

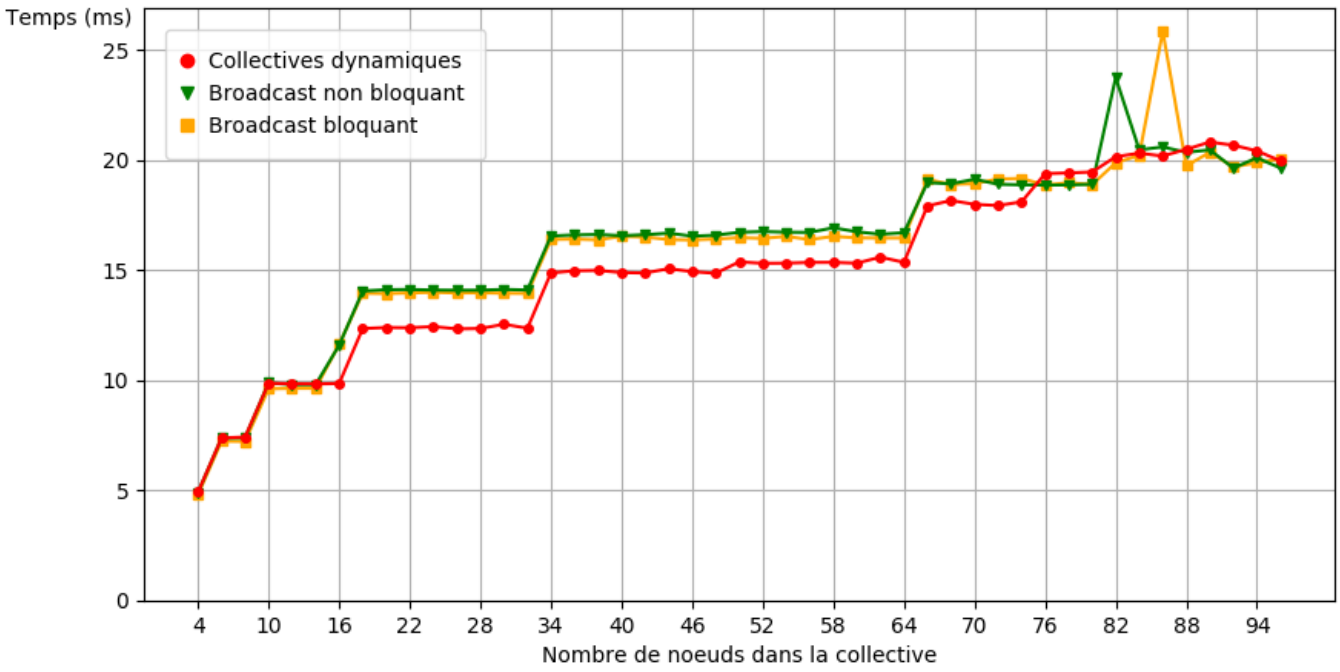


FIGURE 11 – Zoom sur le microbenchmark avec NEWMADELEINE.

blocs et chaque nœud se charge de calculer les valeurs d'un bloc. C'est également ces blocs qui sont transmis entre les nœuds et qui vont créer des communications collectives. Les blocs sont aussi appelés des **tuiles**.

Les tests ont été réalisés sur 64 nœuds sur la machine INTI du CEA avec des tailles de blocs de 320×320.

La FIGURE 12 compare les performances obtenues avec et sans les collectives dynamiques. Les courbes présentées sont la moyenne de plusieurs exécutions et les aires entourant les courbes représentent les marges d'erreur obtenues. Puisque l'unité de mesure de la performance est maintenant le `flops`, plus grands sont les *flops* obtenus, meilleures sont les performances.

L'interprétation principale de ce graphe est que **l'utilisation de collectives dynamiques améliore sensiblement les performances de la décomposition de CHOLESKY**.

Cependant, plusieurs remarques :

— ces décompositions de CHOLESKY ont été exécutées sur 64 nœuds. Cela signifie que les plus grandes collectives en terme de nombre de destinataires contiennent 7 nœuds destinataires (voir l'ANNEXE A). Il serait intéressant d'augmenter le nombre de nœuds sur lesquels les tests de performances sont menés, pour voir à quel point l'écart entre les performances avec et sans les collectives dynamiques se creuse.

— les performances sont mesurées pour des tailles de matrices allant jusqu'à 67200×67200. Il serait intéressant de voir les performances pour des matrices encore plus grandes. En effet, les graphes de l'article [14], qui présentent les performances de la décomposition de CHOLESKY sans les collectives dynamiques mais dans le même contexte que celui des performances mesurées ici, montrent que les performances atteignent un plafond à partir d'une certaine

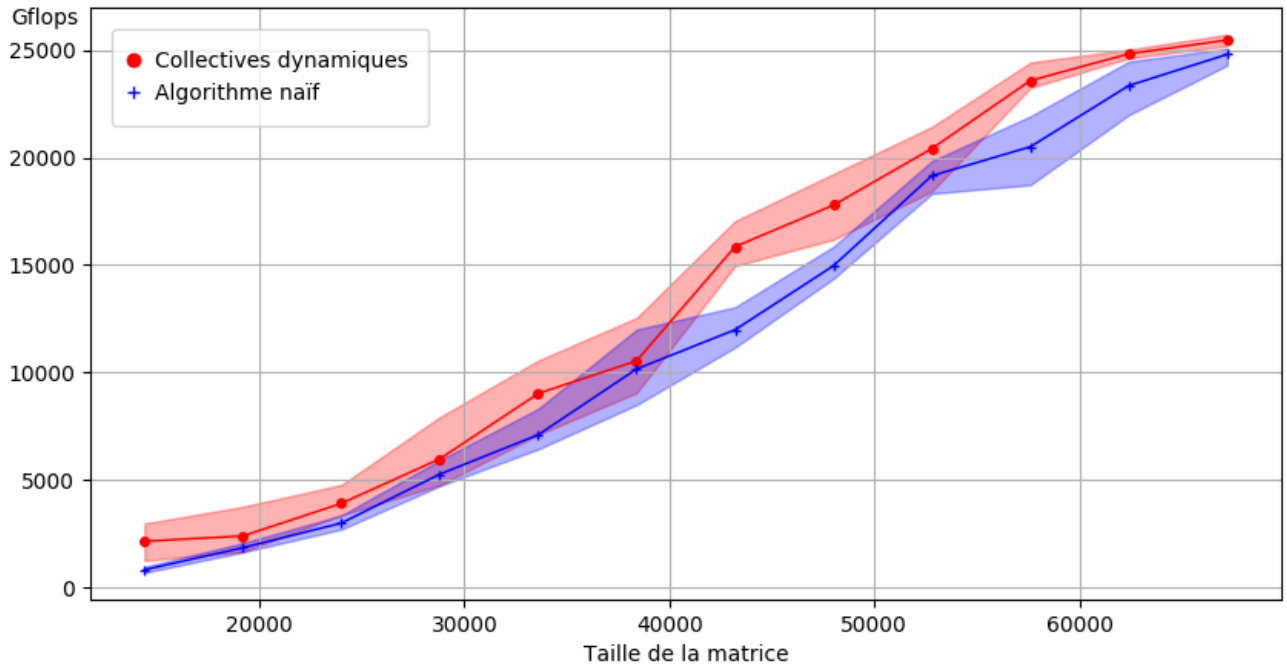


FIGURE 12 – Test de performance de la décomposition de CHOLESKY avec et sans les collectives dynamiques.

taille de matrice. Où se situerait la courbe rouge des collectives dynamiques lorsque la courbe bleue de l'algorithme naïf atteindrait ce plafond ?

— les résultats obtenus en pratique montrent une amélioration des performances, mais il peut être pertinent de savoir si ce gain de performance correspond au gain théorique. Pour le savoir, il est nécessaire de compter le nombre de collectives avec leurs nombres de nœuds destinataires. L'ANNEXE A présente quelques éléments pour faciliter le décompte.

— comme évoqué précédemment, la détection par STARPU des communications collectives est une tâche qui peut s'avérer difficile. D'où l'intérêt de savoir si toutes les communications collectives de la décomposition de CHOLESKY sont bien détectées comme telles lors de l'exécution. Encore une fois, le décompte de l'ANNEXE A et les statistiques affichées par STARPU nous aident à répondre à la question. Expérimentalement, il a été remarqué que les collectives sont bien détectées, à part parfois des collectives qui sont scindées en deux, c'est-à-dire une collective vers n nœuds destinataires est détectées comme deux collectives vers $\frac{n}{2}$ destinataires.

Ces points montrent qu'il reste encore du travail pour analyser plus en profondeur les résultats obtenus, le temps imparti au stage n'ayant pas suffit...

Pour résumer, les collectives dynamiques permettent d'améliorer les performances de la décomposition de CHOLESKY. Cependant, il reste encore à savoir si l'utilisation des collectives dynamiques est exploitée au maximum. Finalement, la décomposition de CHOLESKY n'est qu'une application qui nous intéresse car elle génère beaucoup de collectives, mais il en existe d'autres pour lesquelles il serait possible de mesurer le gain de performance obtenu avec les collectives dynamiques. Durant mon stage, il a notamment été question d'essayer la multiplication matricielle par blocs.

Conclusion

Les communications réseau des applications HPC est un facteur qui peut rapidement empêcher un passage à l'échelle efficace. Pour permettre un meilleur passage à l'échelle des applications distribuées utilisant StarPU, les communications collectives étaient un mécanisme qu'il était possible d'améliorer. L'utilisation d'arbres binomiaux pour répartir la diffusion des données à travers la collective a permis de grandement améliorer la vitesse d'exécution des communications collectives. En rendant ces collectives dynamiques, nous avons adapté l'algorithme aux besoins de StarPU, pour que les nœuds participant à une collective ne le découvrent qu'à la réception des données et ne soient plus dépendants des autres nœuds.

Les tests ont montré que les communications collectives dynamiques offrent des performances semblables à celles attendues en théorie et permettent d'accélérer les applications provoquant des communications collectives.

Ce travail sur les communications collectives a aussi été l'occasion de mettre en place une horloge globale distribuée, nécessaire pour la visualisation des traces et pour les mesures des performances des microbenchmarks.

Au-delà des contributions que j'ai pu apporter à NewMadeleine et StarPU, ce stage m'a permis de découvrir le monde du calcul haute performance, ses enjeux et ses problématiques. J'ai également beaucoup appris en programmation système et en déverminage. J'ai aussi pu améliorer mon expérience en développement logiciel en contribuant à divers logiciels et bibliothèques qui m'étaient auparavant inconnus. Ce stage montre également l'une des capacités les plus souvent citées à propos du métier d'ingénieur : l'adaptation à de nouveaux domaines. Malgré mon semestre de spécialisation en cyber-sécurité, ce stage s'est très bien déroulé et j'y ai appris beaucoup de nouvelles choses.

Bien que cette conclusion marque la fin de ce stage de fin d'études, elle marque également le début d'une thèse. En effet, je continuerai le travail commencé lors de ce stage par une thèse sur le passage à l'échelle des communications de StarPU avec NewMadeleine. Ce doctorat permettra d'améliorer les interactions entre NewMadeleine et StarPU, en exploitant au maximum les possibilités de NewMadeleine par StarPU et en donnant à NewMadeleine toutes les informations dont dispose StarPU qui pourraient aider dans la gestion des communications. Il sera question de la caractérisation des communications irrégulières de StarPU pour adapter NewMadeleine en conséquence ; des communications collectives : analyser plus en profondeur les performances obtenues sur l'algorithme de Cholesky, mais aussi tester d'autres applications et d'autres algorithmes de collectives ; changer certains paramètres de NewMadeleine à la volée, en fonction de la charge de communication imposée par StarPU ; et finalement exploiter le graphe de tâches que se construit StarPU, pour pouvoir prédire les communications à venir.

Références

[1] InfiniBand Trade Association. https://www.infinibandta.org, 2019.

[2] Top500. https://www.top500.org/, 2019.

[3] MPI Forum. https://www.mpi-forum.org/, 2019.

[4] Open MPI : Open Source High Performance Computing. https://www.open-mpi.org/, 2019.

[5] Open MP : The OpenMP API specification for parallel programming. https://www.openmp.org/, 2019.

[6] MPICH : High-Performance Portable MPI. https://www.mpich.org/, 2019.

[7] MVAPICH : MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE. http://mvapich.cse.ohio-state.edu/, 2019.

[8] NewMadeleine : An Optimizing Communication Library for High-Performance Networks. http://pm2.gforge.inria.fr/newmadeleine/, 2019.

[9] MPC. http://mpc.hpcframework.paratools.com/, 2019.

[10] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. StarPU : A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Euro-Par 2009*, LNCS, August 2009.

[11] Olivier Aumage, Elisabeth Brunet, Nathalie Furmento, and Raymond Namyst. NEW MADELEINE : a Fast Communication Scheduling Engine for High Performance Networks. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–8, Long Beach, CA, USA, 2007. IEEE.

[12] Guillaume Beauchamp. Portage de StarPU sur la bibliothèque de communication NewMadeleine. Technical report, 2017.

[13] Udayanga Wickramasinghe and Andrew Lumsdaine. A survey of methods for collective communication optimization and tuning. *CoRR*, abs/1611.06334, 2016.

[14] Alexandre Denis. Scalability of the NewMadeleine Communication Library for Large Numbers of MPI Point-to-Point Requests. CCGrid 2019 - 19th Annual IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing, May 2019.

[15] ViTE. http://vite.gforge.inria.fr/, 2019.

[16] Sascha Hunold and Alexandra Carpen-Amarie. On the Impact of Synchronizing Clocks and Processes on Benchmarking MPI Collectives. pages 1–10, 09 2015.

[17] Chameleon. https://gitlab.inria.fr/solverstack/chameleon, 2019.

A Communications collectives dans la décomposition de CHOLESKY

Avoir des informations sur les collectives (nombre de collectives émises par chaque nœud, nombre de nœuds destinataires dans chaque collective) dans la décomposition de CHOLESKY permet deux choses :

— s'assurer que STARPU reconnaît bien toutes les collectives et n'omet pas de nœud destinataire ;

— pouvoir calculer le temps mis par l'ensemble des collectives et ainsi comparer les résultats mesurés avec les résultats théoriques.

A.1 Rappel de l'algorithme

Nous nous intéressons à la décomposition de CHOLESKY **par blocs**. Découper la matrice en blocs facilite la répartition des calculs sur différents nœuds.

En utilisant la répartition par blocs cycliques, la tuile de coordonnées (i, j) se situe sur le nœud $(j \bmod Q) \cdot P + (i \bmod P)$ où P et Q sont les dimensions de la grille de nœuds.

Dans l'algorithme suivant, A désigne la matrice dont on souhaite obtenir une triangulaire inférieure. C'est elle qui va être transformée pour contenir cette triangulaire. $A[i][j]$ désigne le bloc de la ligne i et de la colonne j de la matrice A. Cet algorithme reposant sur la symétrie de A, la triangulaire supérieure de A est ignorée.

for $j = 0\,;\, j < N\,;\, j{+}{+}$ **do**
  `POTRF`(`RW`, $A[j][j]$)
  for $i = j{+}1\,;\, i < N\,;\, i{+}{+}$ **do**
    `TRSM`(`RW`, $A[i][j]$, `R`, $A[j][j]$)
  end for
  for $i = j{+}1\,;\, i < N\,;\, i{+}{+}$ **do**
    `SYRK`(`RW`, $A[i][i]$, `R`, $A[i][j]$)
    for $k = j{+}1\,;\, k < i\,;\, k{+}{+}$ **do**
      `GEMM`(`RW`, $A[i][k]$, `R`, $A[i][j]$, `R`, $A[k][j]$)
    end for
  end for
end for

Les fonctions utilisées correspondent à celles définies par les BLAS (*Basic Linear Algebra Subprograms*) :

— `POTRF` : effectue la factorisation de CHOLESKY matricielle ;

— `TRSM` : résout une équation matricielle triangulaire de la forme $AX = B$ et stocke la solution X dans B ;

— `SYRK` : place dans C le résultat de l'opération $AA^t + C$;

— `GEMM` : effectue une multiplication matricielle .

`R` signifie un accès en lecture au bloc et `RW` un accès en lecture et écriture.

L'algorithme peut être vu comme récursif (voir FIGURE 13) : lors de l'itération j de la première boucle, les colonnes d'indices inférieurs à j ne sont plus utilisées, elles contiennent le résultat définitif de la décomposition de CHOLESKY.

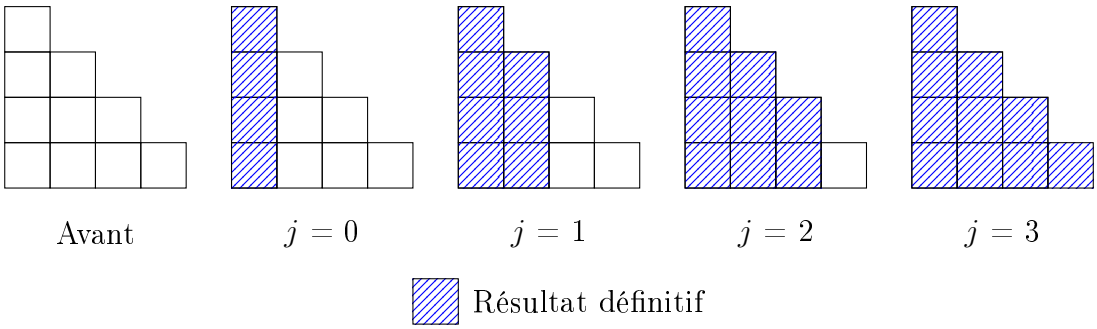


FIGURE 13 – Progression de l'algorithme de la décomposition de CHOLESKY

Pour calculer le nombre de collectives, il suffit donc de les compter pour une itération de la première boucle et de sommer ensuite.

A.2 Premier cas : nombre de tuiles égal au nombre de nœuds

Dans ce cas, chaque nœud a en charge une tuile et donc lorsqu'un nœud fait une opération sur une tuile qui a besoin d'une autre tuile, il est certain qu'un transfert entre deux nœuds va avoir lieu.

A.2.1 Décompte des collectives

Pour compter les collectives d'une itération de la boucle principale, intéressons-nous à son contenu. Puisque nous considérons pour l'instant une unique itération de cette boucle, j est donc constant. Nous pouvons le définir valant 0 puisque, l'algorithme étant récursif, il suffit de considérer que lors de la prochaine itération de la boucle, N vaut $N - 1$.

`POTRF` Le `POTRF` n'a pas besoin d'autres tuiles pour s'exécuter : il n'y a donc pas de transfert entre nœuds.

`TRSM` Le résultat du `POTRF`, le bloc $A[j][j]$, est ensuite transmis à tous les nœuds de sa colonne (boucle sur les `TRSM`, voir FIGURE 14). Cela fait donc une collective avec $N - 1$ destinataires.

`SYRK` et `GEMM` Les communications réseau pour les boucles englobant les `SYRK` et les `GEMM` sont un peu plus difficiles à compter. Réécrivons cette partie de l'algorithme en mettant mieux en évidence les transferts entre nœuds et en remplaçant j par 0 :

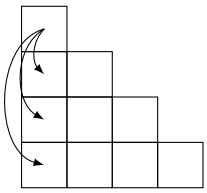


FIGURE 14 – Transferts liés aux `TRSM`

for $i = 1\,;\, i < N\,;\, i{+}{+}$ **do**
  $A[i][0] \rightarrow A[i][i]$
  for $k = 1\,;\, k < i\,;\, k{+}{+}$ **do**
    $A[i][0] \rightarrow A[i][k]$
    $A[k][0] \rightarrow A[i][k]$
  end for
end for

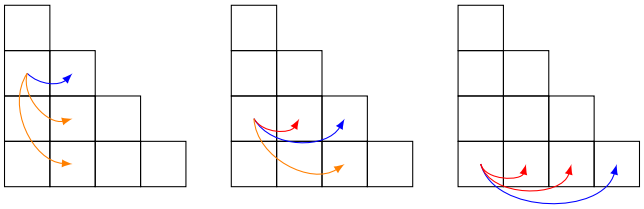


FIGURE 15 – Transferts liés aux `SYRK` et `GEMM`

La FIGURE 15 nous aide à comprendre que chaque tuile de la première colonne (excepté celle de la première) produit une collective vers $N-1$ nœuds. Il y a donc $N-1$ collectives chacune contenant $N-1$ destinataires.

Total pour une itération Pour cette itération de la boucle sur j, il y a donc N collectives vers $N-1$ nœuds. j n'étant en réalité pas constant – c'est lui qui influe sur la taille de la matrice lors de la récursion –, **il y a $N-j$ collectives vers $N-j-1$ nœuds**.

Chaque nœud s'occupant d'une unique tuile, il est possible de remarquer également que chaque nœud s'occupant d'une tuile située sur la triangulaire inférieure ne produit qu'une collective. Les autres nœuds ne font pas de collectives.

Ainsi, pour s'assurer que les collectives sont bien détectées, il est utile de savoir que **le nœud n va générer une collective vers $N - \lfloor \frac{n}{P} \rfloor - 1$ nœuds**. Cette information pour chaque nœud nous est utile car STARPU peut nous afficher pour chaque nœud des statistiques relatives aux communications.

A.2.2 Durée des communications collectives

Nous allons maintenant comparer la durée de toutes les communications collectives d'une exécution de la décomposition de CHOLESKY en utilisant l'algorithme avec les collectives dynamiques ou l'algorithme naïf.

La durée des communications collectives peut être donnée par la formule suivante :

$$T(N) = \sum_{k=0}^{N-2} (N-k) f(N-k-1)$$

avec N la racine du nombre de tuiles et f une fonction qui donne pour un nombre de nœuds destinataires, la durée de la collective. Cette somme résulte du fait qu'à chaque itération k de la décomposition de CHOLESKY, il y a $N-k$ collectives vers $N-k-1$ nœuds. La somme va seulement jusqu'à $N-3$ car les deux dernières itérations ne génèrent pas de communications collectives et le décalage d'indice emmène la somme seulement jusque $N-1$.

Algorithme naïf Pour l'algorithme linéaire, la complexité temporelle est linéaire, on a donc $f : x \mapsto \alpha x$ avec α la durée du transfert d'une tuile d'un nœud à un autre. Après simplification de la somme, nous obtenons alors :

$$T_{seq}(N) = \alpha \frac{N^3}{3} + \mathcal{O}(N^3)$$

Algorithme avec arbre binomial Pour cet algorithme, la complexité temporelle est logarithmique, on a donc $f : x \mapsto \alpha \log(x)$. Après simplification de la somme, nous obtenons :

$$T_{bino}(N) = \alpha \frac{N^2 \log(N)}{2} + \mathcal{O}(N^2)$$

Gain obtenu En faisant le rapport des deux formules trouvées, le facteur de gain devrait être de $\frac{2N}{3\log(N)}$ avec N la racine carrée du nombre de tuiles. Ce facteur de gain concerne uniquement la durée des communications collectives et ne se ressentira probablement pas complètement dans les valeurs mesurant la performance des algorithmes à cause du recouvrement calculs / communications.

Le cas où le nombre de tuiles correspond précisément au nombre de nœuds ne correspond malheureusement pas vraiment à la réalité : si chaque nœud ne s'occupe que d'une tuile, les performances chutent puisque le parallélisme de l'application est réduit. De plus, pour augmenter la taille des matrices, il est nécessaire soit d'augmenter la taille des blocs, soit d'augmenter le nombre de nœuds ; dans tous les cas l'exploitation du parallélisme n'est pas optimale. Et c'est sans compter avec les nœuds qui ont en charge la triangulaire supérieure de la matrice : la matrice étant symétrique, ils ne font aucun calcul !

A.3 Second cas : nombre de tuiles différent du nombre de nœuds

Ce cas-ci est problématique pour compter théoriquement les collectives, car un nœud peut s'occuper de plusieurs tuiles. Ainsi, certains transferts ne se feront pas, car deux tuiles nécessaires pour une opérations peuvent se trouver sur le même nœud ou bien les données auront déjà été transmises lors d'une opération sur une autre tuile. Par exemple, la FIGURE 16 montre une matrice de $N = 5$ avec une grille de 3×3 nœuds : le transfert de $(0,0)$ vers $(0,3)$ ne se fait pas car ces

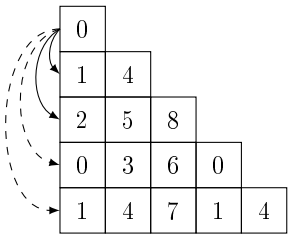


FIGURE 16 – Transferts liés aux `TRSM` : les transferts en pointillés ne se font pas.

deux tuiles sont sur le même nœud et le transfert de $(0, 0)$ vers $(0, 4)$ ne se fait pas car cette tuile a déjà été transférée vers le nœud 1 lors du transfert de $(0, 0)$ vers $(0, 1)$.

Compter les collectives pour avoir une formule en fonction du nombre de tuiles est bien plus compliqué que dans le cas où chaque nœud s'occupe d'une unique tuile. En revanche, ce qui est faisable, c'est de simuler l'algorithme de la décomposition de CHOLESKY et relever ainsi les transferts effectifs.

Opérations collectives dynamiques dans StarPU / NewMadeleine

Philippe SWARTVAGHER

Maître de stage: Alexandre DENIS

Tuteur: Mathieu FAVERGE

Bordeaux INP ENSEIRB MATMECA

Informatique

RSR

HPC: High Performance Computing

- Utiliser des ordinateurs pour faire d'importants calculs numériques : calcul scientifique, simulations (météorologie, mécanique, physique, chimie, ...)
- **Objectif:** utiliser la puissance maximale des ordinateurs: performance mesurée en `flops` (*floating-point operations per second*)
- Un supercalculateur (superordinateur, cluster, ...) se compose de:
  - ordinateurs connectés entre eux par un réseau haute performance...
  - ... possédant plusieurs unités de calcul (plusieurs CPU, GPU, ...)...
  - ... avec chacun plusieurs cœurs.
- Les programmes sont parallélisés et répartis sur toutes les unités de calcul pour gagner en performance.
- **Les communications réseau sont un des goulots d'étranglement pour les performances.**

StarPU

- Bibliothèque logicielle de programmation par tâches pour architectures hétérogènes
- L'utilisateur fournit:
  - la décomposition de son programme en tâches,
  - les tâches écrites pour chaque type d'unité de calcul,
  - le graphe de dépendances des tâches.
- StarPU s'occupe du reste:
  - ordonnancement des tâches sur les unités de calcul,
  - transferts mémoire nécessaires entre les unités de calcul
- L'objectif est de faciliter l'utilisation simultanée de différentes unités de calcul (CPU, GPU, ...).
- Les communications réseau sont laissées à la bibliothèque NewMadeleine, ou toute autre bibliothèque supportant le standard MPI.
- Codée en C, sous licence libre

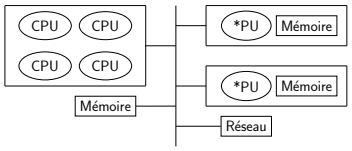


Figure: Éléments constituant un nœud de supercalculateur

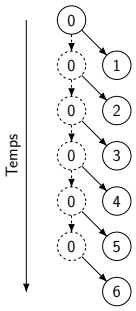
NewMadeleine

- Bibliothèque de communication pour le HPC
- Optimisation à la volée des flux de paquets (agrégation, ordonnancement, ...)
- Programmation asynchrone : permet le recouvrement calcul / communication

Communications collectives: broadcast

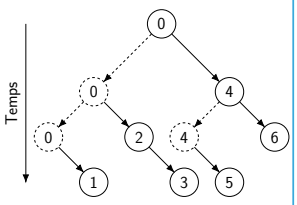
- Un nœud envoie une même donnée vers différents nœuds.
- Utilisées dans de nombreux algorithmes numériques
- Pour StarPU: des tâches placées sur différents nœuds (1 à 6, sur les figures ci-dessous) ont comme même dépendance le résultat d'une tâche exécutée sur le nœud 0.
- Équivalent d'un *multicast*, appelé *broadcast* par abus de langage

Algorithme naïf



- Le nœud source envoie séquentiellement à tous les nœuds destinataires.
- **Complexité en nombre de nœuds: linéaire**

Algorithme avec arbre binomial



- Les nœuds qui ont déjà reçu la donnée se mettent également à envoyer aux nœuds qui ne l'ont pas encore reçue.
- **Complexité en nombre de nœuds: logarithmique**
- D'autres algorithmes existent.

Collectives dynamiques pour StarPU

- Dans les implémentations classiques de collectives, tous les nœuds de la collective doivent s'attendre pour exécuter la collective en même temps. Ils doivent également connaître la liste de tous les nœuds dans la collective pour dérouler leur algorithme. De plus, l'utilisateur indique explicitement qu'il s'agit d'une collective.
- Ces contraintes ne conviennent pas à StarPU car l'exécution des tâches n'est pas synchronisée entre les nœuds. **Attendre des nœuds pour faire une collective impacte donc ses performances.**
- **Mise en place de collectives dynamiques**: les nœuds peuvent être impliqués dans des collectives sans le savoir avant de recevoir les données (collectives *implicites*):
  - utilisation d'arbres binomiaux pour profiter de leur complexité en nombre de nœuds destinataires
  - plus besoin d'attendre que tous les nœuds destinataires soient prêts à recevoir la donnée pour mettre en place la collective: les nœuds découvrent qu'ils sont dans une collective lorsqu'ils reçoivent les données

Microbenchmark

- Uniquement avec NewMadeleine
- Comparaison de l'algorithme naïf et des collectives dynamiques
- Transfert de 8 Mo, en faisant augmenter le nombre de nœuds destinataires
- Médianes de 10 valeurs

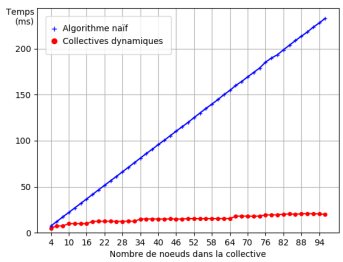


Figure: Microbenchmark avec NewMadeleine - machine Inti du CEA

- Résultats correspondant aux attentes: les complexités des deux algorithmes sont bien visibles.

Test applicatif

- Mesures de performances sur la décomposition de Cholesky
- Opération qui produit beaucoup de collectives
- Utilisation de Chameleon: bibliothèque d'algèbre linéaire pour matrices denses, qui utilise déjà StarPU

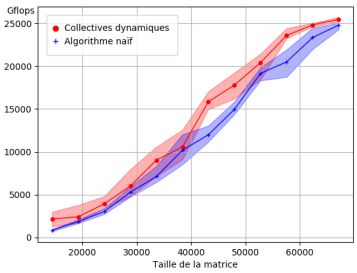


Figure: Performances obtenues sur la factorisation de Cholesky avec 64 nœuds - machine Inti du CEA

- Résultats satisfaisants: amélioration des performances en utilisant les collectives dynamiques

INRIA

- *Institut National de Recherche en Informatique et Automatique*
- 200 équipes réparties dans 8 centres à travers toute la France
- Principalement des équipes s'intéressant au calcul haute performance, à la simulation numérique, à la robotique et à la santé au centre de Bordeaux Sud-Ouest

TADaaM

- *Topology-Aware System Scale Data Management for High Performance Computing Applications*
- Équipe qui m'a accueilli pour mon stage
- A pour objectif la création d'une pile logicielle pour abstraire les caractéristiques matérielles des machines et s'adapter aux besoins des applications HPC

Travaux futurs

- Analyser plus en profondeur les résultats obtenus avec la décomposition de Cholesky, pour s'assurer que les performances théoriques sont atteintes
- Tester d'autres applications numériques et d'autres algorithmes de collectives
- **Thèse**: analyser le schéma de communication de StarPU et comprendre les paramètres impactant les performances, augmenter et exploiter le partage d'informations entre NewMadeleine et StarPU pour améliorer les performances liées aux communications