Communications collectives et ordonnancement en régime permanent sur plates-formes hétérogènes

Loris Marchal

N° d'ordre : 377
N° attribuée par la bibliothèque : 06ENSL0 377

ÉCOLE NORMALE SUPÉRIEURE DE LYON
Laboratoire de l'Informatique du Parallélisme

THÈSE

pour obtenir le grade de

Docteur de l'École Normale Supérieure de Lyon
spécialité : Informatique

au titre de l'école doctorale de MathIF

présentée et soutenue publiquement le 17 octobre 2006

par Monsieur Loris MARCHAL

Communications collectives et ordonnancement en régime permanent sur plates-formes hétérogènes

Directeurs de thèse :	Monsieur	Olivier	BEAUMONT
	Monsieur	Yves	ROBERT
Après avis de :	Monsieur	Pierre	FRAIGNIAUD
	Madame	Alix	MUNIER-KORDON

Devant la commission d'examen formée de :

Monsieur	Olivier	BEAUMONT, Directeur de Thèse
Monsieur	Franck	CAPPELLO
Monsieur	Pierre	FRAIGNIAUD, Rapporteur
Madame	Alix	MUNIER-KORDON, Rapporteur
Monsieur	Yves	ROBERT, Directeur de Thèse
Monsieur	Laurent	VIENNOT

Remerciements

Ma première collaboration scientifique avec Yves Robert date d'une rencontre un peu fortuite à San Diego. J'effectuais un stage de maîtrise chez Henri Casanova, avec qui Yves venait travailler, accompagné d'Arnaud Legrand. J'étais alors un peu impressionné de devoir le côtoyer et le tutoyer, lui que je ne connaissais que comme professeur d'algorithmique à l'ENS, tout en me demandant si j'aurais un jour la chance de travailler avec lui. Cinq années de collaboration plus tard, mon appréhension et ma réserve ont quelque peu diminué, mais non l'estime et la gratitude pour m'avoir accueilli dans son équipe. Sa notoriété et son expérience ne sont jamais des obstacles à des échanges enrichissants, voire à de franches confrontations d'idées.

C'est grâce à Yves que j'ai fait la connaissance d'Olivier Beaumont, qui est également devenu mon directeur de thèse. Bien que ne le connaissant pas avant le début de mon DEA, et malgré la distance séparant son laboratoire bordelais du LIP (et la non moindre distance séparant son cœur stéphanois des lyonnais), j'ai eu un grand plaisir à travailler avec lui. J'ai également beaucoup apprécié son accueil chaleureux à Bordeaux ainsi que la curiosité qu'il a suscité en moi pour des sujets scientifiques a priori accessoires à cette thèse, mais tout aussi intéressants.

Pendant mes années au LIP, j'ai également beaucoup apprécié travailler avec Arnaud Legrand, dont la sagacité n'a d'égale que la gentillesse, ainsi qu'avec Frédéric Vivien chez qui la critique aiguisée et pointilleuse cache en réalité une grande rigueur intellectuelle. Mais ne vous méprenez pas, ces compliments ne sont pas gratuits puisque j'espère bien avoir la chance de collaborer à nouveau avec Yves, Olivier, Arnaud ou Fredo !

J'aimerais également remercier ici Pierre Fraigniaud et Alix Munier pour le travail de rapporteur qu'ils ont effectué sur mon manuscrit, ainsi que Franck Cappello et Laurent Viennot pour avoir accepté de faire partie de mon jury.

Je tiens aussi à remercier tous les membres du LIP qui m'ont permis de passer d'agréables années de thèse, en particulier les directeurs Jean-Michel Muller et Frédéric Desprez et les secrétaires Sylvie, Isabelle et Corinne. Merci également à tous les membres de GRAAL pour la bonne ambiance de l'équipe, en particulier aux thésards pour les interminables parties de frozen-bubble, de spong, les discussions de geek et autres sorties nocturnes : merci à Arnaud et Martin pour leurs conseils linuxiens et frozen-bubblesques, à Abdou pour ses appositions de mains sur ma debian, à Hélène pour ses histoires inimaginables, à Antoine pour sa disponibilité aux pauses cafés, à Pushpinder pour ses repas indiens et à Raphaël pour accepter de perdre si régulièrement et si tragiquement à frozen-bubble. Merci également aux nouveaux, Emmanuel,

Jean-François, Cédric, Jean-Sébastien, Mathieu, Veronika, pour nous supporter et nous apporter leur bonne humeur. Bon courage à vous !

Un grand merci également à mes parents pour m'avoir supporté et soutenu pendant ces années de thèse, ainsi qu'aux amis pour leur présence, à commencer par les inestimables Lazos : David, Couchoud, Caro, Charles, Arnaud et Charlotte. Merci également aux anciens de l'ENS, Julien, Greg, Jérémie, Emmanuelle, Benoît et Jérémie. Merci enfin à Alan et Holly avec qui j'ai aimé partagé de longs moments, et à tous les amis lyonnais, dont le soutien à l'aide d'interminables soirées à la Mi-Graine a certainement été déterminant, en particulier Arnaud, Élise, Raphaële et (encore) David. Enfin un immense merci à Bénédicte pour ses encouragements et bien plus encore.

Table des matières

Chapitre 1

Introduction

Depuis les premiers développements de l'informatique, la puissance de calcul des ordinateurs n'a jamais cessé de croître. Étonnamment, il s'est toujours trouvé des utilisateurs pour utiliser ces machines de plus en plus rapides, et même pour pousser au développement de machines encore plus puissantes, contredisant ainsi un relecteur de l'article de C.E. Shannon "A Mathematical Theory of Communication." [90] qui, en 1948, déclarait :

> *The author mentions computing machines, such as the recent ENIAC. Well, I guess one could connect such machines, but a recent IBM memo stated that a dozen or so such machines will be sufficient for all the computing that we'll ever need in the foreseeable future, so there won't be a whole lot of connecting going on with only a dozen ENIACs !* [1]

En fait, prenant l'exact contre-pied de l'opinion de ce relecteur peu visionnaire, les processeurs ont été rassemblés et connectés pour créer des machines plus puissantes, jusqu'à utiliser aujourd'hui des milliers d'unités de calcul. Les applications qui aujourd'hui sont les plus consommatrices de puissance de calcul sont principalement dans le domaine du calcul scientifique, en particulier les simulations dans le domaine de la chimie, de la physique, de la climatologie, etc., pour lesquelles on recherche une grande précision et une fiabilité accrue, et le traitement des données produites par les détecteurs des accélérateurs de particules comme l'américain CDF [105] ou le futur européen LHC [107].

Des machines parallèles, aussi appelées «super-calculateurs» ont donc été créées en assemblant un grand nombre de processeurs identiques et en les reliant par un puissant réseau de communication. Cependant, ces machines ne sont pas toujours dans les moyens de ceux qui ont besoin de puissance de calcul. Une solution moins coûteuse consiste à rassembler des machines existantes réparties sur des sites distants. On parle alors de «grille de calcul». L'hétérogénéité de la plate-forme de calcul ainsi obtenue rend délicate leur utilisation efficace et réserve leur usage pour des applications parallèles relativement simples, qui ne nécessitent pas une forte synchronisation entre unités de calcul. Les problèmes d'ordonnancement, qui étaient déjà difficiles pour des plates-formes homogènes, deviennent excessivement complexes sur des grilles de calcul où les vitesses des processeurs et des liens de communication sont hétérogènes, le réseau irrégulier, les performances difficiles à prédire et la disponibilité des machines parfois imprédictible.

[1] Publié dans [87].

Notre but est de concevoir des algorithmes d'ordonnancement optimaux ou garantis pour ces nouvelles plates-formes, ce qui est déjà difficile pour des plates-formes homogènes, avec des modèles très simples. C'est pourquoi nous voulons tirer parti de la régularité des applications qui sont exécutées sur ces plates-formes. En effet, sur des plates-formes distribuées relativement instables, l'exécution d'une application parallèle complexe nécessitent de vérifier qu'aucune défaillance n'a lieu, ni aucune perte de messages, et de redémarrer les tâches ayant subi cette défaillance, ou de resynchroniser l'application avant la perte de message. La majorité des applications utilisant ces grilles sont donc simples et naturellement résistantes. Une application typique consiste en un grand nombre de tâches de calcul de même taille, indépendantes entre elles : si une des tâches est interrompue, ou que la transmission de son résultat échoue, il suffit de recommencer cette tâche plus tard, sur un autre processeur, sans que le reste de l'application en pâtisse. Les algorithmes itératifs asynchrones sont autre un type d'application caractéristique des grilles de calcul : même s'ils nécessitent des communications entre les différentes tâches, l'algorithme est naturellement conçu pour résister à la défaillance d'une communication.

Pour un problème présentant une régularité aussi forte que celle des tâches indépendantes, plutôt que d'optimiser l'ordonnancement de ces tâches depuis l'envoi des données de la première tâche jusqu'à la terminaison de la dernière tâche, nous pensons qu'il est plus efficace de se concentrer sur le cœur de l'exécution, en cherchant une utilisation optimale des ressources dans la phase de régime permanent. Nous cherchons ainsi un ordonnancement pour le régime permanent qui réalise un débit optimal, c'est-à-dire un nombre maximal de tâches traitées par unité de temps. Il faut certes également prendre en compte un phase d'initialisation et une phase de terminaison, mais celles-ci sont de taille négligeable dès que le nombre total de tâches à traiter est important. Cette relaxation nous permet de résoudre des problèmes d'ordonnancements plus complexes, pour lesquels la minimisation du temps total d'exécution est souvent inaccessible.

Parmi les problèmes d'ordonnancement relatifs à l'exécution d'applications sur des plates-formes hétérogènes, on s'intéresse plus particulièrement à l'ordonnancement des communications. Lors de l'exécution d'un ensemble de tâches indépendantes, il est nécessaire de distribuer des données aux machines participantes ; ce peut être des données différentes pour chaque machine, ou bien la même donnée doit être diffusée à toutes les machines. Après le traitement de ces données, les résultats devront être rassemblés, éventuellement en les combinant en un résultat global. Ces opérations de communications collectives (distribution de donnés, diffusion, réduction,...) représentent une grande partie des communications nécessaires à tout programme parallèle. Elles ont été étudiées et optimisées pour des environnements homogènes et réguliers. Nous nous intéressons ici leur optimisation pour des réseaux hétérogènes, en nous concentrant sur leur régime permanent.

Dans la suite de ce chapitre, nous présentons l'article fondateur pour l'étude du régime permanent, puis nous étudierons les différents modèles de communications existants et ceux utilisés dans cette étude. Enfin, nous résumerons l'ensemble des sujets étudiés et des résultats obtenus au cours de cette thèse.

1.1 Ordonnancement en régime permanent

L'idée d'étudier le régime permanent en ordonnancement s'inspire d'un article de Bertsimas et Gamarnik [26], qui étudie le problème du routage des paquets dans un réseau. Dans cette

partie, nous rappelons la méthode introduite dans cet article, que nous adapterons ensuite aux problèmes qui nous intéressent.

1.1.1 Routage de paquets

Dans cette étude, un réseau est représenté par un graphe (V, E) où les sommets de V sont les machines, cherchant à transmettre des paquets de données de même taille en utilisant des liens de communication représentés par les arêtes de E. Pour toute paire de nœuds $k, l \in V$, il y a un nombre de paquets $n_{k,l}$ à transmettre du nœud k au nœud l, ce qu'on appelle des paquets de type (k, l). Le réseau est supposé homogène : tous les liens de communication sont identiques, et transmettre un paquet par un lien de communication prend un temps unité, pendant lequel aucun autre paquet ne peut être transmis sur ce lien. En revanche, tous les liens de communication peuvent être utilisés simultanément ; il n'y a pas de contention au niveau des nœuds.

En suivant les notations de [26], on note $\mathcal{P}$ l'ensemble des types de paquets :

$$\mathcal{P} = \big\{(k, l) \text{ tels que } n_{k,l} > 0\big\}$$

Le routage est libre : l'ordonnanceur doit choisir quelle route chaque paquet va emprunter pour rejoindre sa destination, puis à quels instants le paquet utilisera les liens de communication de cette route. Le but du problème est de construire un ordonnancement qui minimise le temps total nécessaire pour que tous les paquets soient acheminés vers leur destination. On appelle C^* le temps d'exécution optimal.

Comme la recherche du temps d'exécution optimal est un problème NP-complet [94], Bertsimas et Gamarnik proposent de rechercher une utilisation du réseau suffisamment efficace pour que le temps d'exécution soit proche de l'optimal. L'algorithme qu'ils proposent s'exécute en temps $C^* + O(\sqrt{C^*})$.

L'idée principale introduite dans cette étude est de s'intéresser aux quantités moyennes de paquets qui transitent sur chaque arête en régime permanent, en se rapprochant d'un problème de multi-flot. On considère un ordonnancement valide quelconque. On peut définir $x_{i,j}^{k,l}$ comme le nombre total de paquets de type (k, l) qui sont transmis par le lien (i, j) dans cet ordonnancement. Ces quantités doivent vérifier certaines contraintes que nous exprimons maintenant.

On peut supposer sans perte de généralité que les paquets (k, l) ne repassent jamais par le nœud k et ne sont pas réémis par le nœud l, ce qui s'exprime par :

$$x_{i,k}^{k,l} = 0 \text{ et } x_{l,j}^{k,l} = 0 \quad \forall i, j, k, l \in V$$

On définit pour chaque arête (i, j) la quantité totale de paquets transmis par cette arête :

$$C_{i,j} = \sum_{(k,l)\in P} x_{i,j}^{k,l},$$

ce qui est également le temps total d'occupation de l'arête (i, j). On appelle $C_{\max}$ leur maximum :

$$C_{\max} = \max_{i,j} C_{i,j}.$$

Le temps total d'exécution de l'ordonnancement est clairement plus grand que $C_{\max}$.

Considérons le programme linéaire suivant :

$$
\begin{array}{l}
\text{MINIMISER } C_{\max}, \\
\text{SOUS LES CONTRAINTES} \\
\left\{
\begin{array}{ll}
\text{(1.1a)} & \displaystyle\sum_{i:(k,i)\in E} x_{k,i}^{k,l} = n_{k,l}, \text{ pour } (k,l) \in \mathcal{P}, \\
\text{(1.1b)} & \displaystyle\sum_{i:(i,l)\in E} x_{i,l}^{k,l} = n_{k,l}, \text{ pour } (k,l) \in \mathcal{P}, \\
\text{(1.1c)} & \displaystyle\sum_{j:(j,i)\in E} x_{j,i}^{k,l} = \sum_{r:(i,r)\in E} x_{i,r}^{k,l}, \text{ pour } (k,l) \in \mathcal{P}, i \neq k, i \neq l \\
\text{(1.1d)} & C_{i,j} = \displaystyle\sum_{(k,l)\in P} x_{i,j}^{k,l}, \text{ pour } (i,j) \in E \\
\text{(1.1e)} & C_{i,j} \leqslant C_{\max}, \text{ pour } (i,j) \in E \\
\text{(1.1f)} & x_{i,j}^{k,l}, C_{i,j} \geqslant 0, \text{ pour } (i,j) \in E, (k,l) \in \mathcal{P}.
\end{array}
\right.
\end{array}
\tag{1.1}
$$

On vérifie que pour tout ordonnancement, les variables $x_{i,j}^{k,l}$ définies précédemment vérifient les contraintes de ce programme linéaire :

- les trois premières contraintes expriment que les $x_{i,j}^{k,l}$ définissent un flot entre k et l de valeur $n_{k,l}$: tous les messages sont émis par la source k (1.1a), tous sont reçus par la destination (1.1b) et il y a conservation du nombre de messages de type (k,l) en un nœud i qui n'est ni source ni destination (1.1c) ;
- les deux contraintes suivantes reprennent la définition du temps d'occupation des arêtes, et définissent la fonction objective comme leur maximum.

Comme tout ordonnancement doit vérifier les contraintes de ce programme linéaire, la valeur optimale de la fonction objective $C_{\max}$ est une borne inférieure sur le temps d'exécution optimal C^* d'un ordonnancement.

Ce programme linéaire (en nombres rationnels) peut être résolu par un des nombreux programmes de résolution (comme lp-solve [24], Maple [39] ou mupad [96]) et on utilise les valeurs des variables $x_{i,j}^{k,l}$ ainsi que la valeur de la fonction objective pour construire un ordonnancement s'approchant de l'optimal. Cet ordonnancement est composé de deux phases : une phase périodique et une phase finale de nettoyage. La première phase est organisée en périodes de tailles Ω, définies par les intervalles $[m\Omega;(m+1)\Omega]$ pour $m = 0 \dots \left\lceil \frac{C_{\max}}{\Omega} \right\rceil - 1$. La taille de la période sera calculée plus loin, pour minimiser le temps d'exécution. Pendant chacune de ces périodes, le nombre $a_{i,j}^{k,l}$ de paquets de type (k,l) transportés par l'arête (i,j) est choisi pour être proportionnel à $x_{i,j}^{k,l}$:

$$a_{i,j}^{k,l} = \left\lfloor \frac{x_{i,j}^{k,l}\,\Omega}{C_{\max}} \right\rfloor$$

En réalité, il se peut qu'un tel nombre de paquets ne soient pas disponibles sur le nœud k au début de la période. Dans ce cas, l'arête (i,j) transporte autant de paquets que disponible, mais jamais plus de $a_{i,j}^{k,l}$.

Après la fin de cette phase, c'est-à-dire au temps $T = \left\lceil \frac{Cmax}{\Omega} \right\rceil \Omega$, il se peut que des paquets ne soient pas encore arrivés à destination. Ces paquets sont alors routés séquentiellement (donc de façon très inefficace).

On peut vérifier simplement la validité de cet ordonnancement : pendant une période, le nombre total de paquets transmis par l'arête (i, j) est :

$$\sum_{(k,l)\in\mathcal{P}} a_{i,j}^{k,l} = \sum_{(k,l)\in\mathcal{P}} \left\lfloor \frac{x_{i,j}^{k,l}\Omega}{C_{\max}} \right\rfloor \leqslant \sum_{(k,l)\in\mathcal{P}} \frac{x_{i,j}^{k,l}\Omega}{C_{\max}} = \frac{C_{i,j}}{C_{\max}}\Omega \leqslant \Omega$$

(la dernière inégalité venant de 1.1e). Dans la seconde phase, comme un seul paquet est traité à chaque instant, il n'y a pas non plus de congestion.

Le temps d'exécution de la seconde phase peut être borné par $M \times L$, où M est le nombre de paquets restant dans le réseau après la première phase, et L la longueur du plus long chemin d'une source à une destination, en particulier $L \leqslant |V|$.

Dans [26], Bertsimas et Gamarnik montrent que l'on peut borner le nombre de paquets résiduels à l'issue de la première phase (nous ne détaillons pas ici leur calcul) :

$$M \leqslant \left(\frac{C_{\max}}{\Omega} + 1\right) |E|\,|\mathcal{P}| + |E|\,\Omega$$

Pour minimiser cette quantité, on choisit une longueur de période $\Omega = \sqrt{C_{\max}\,|\mathcal{P}|}$. Le nombre de paquets résiduels est alors borné par

$$M \leqslant 2\,|E|\,\sqrt{C_{\max}\cdot|\mathcal{P}|} + |E|\,|\mathcal{P}|$$

L'ordonnancement proposé s'exécute en un temps C_H tel que

$$C_{\max} \leqslant C^* \leqslant C_H \leqslant C_{\max} + O(\sqrt{C_{\max}})$$

L'ordonnancement ainsi réalisé est *asymptotiquement optimal* quand le nombre de paquets à router augmente :

$$\frac{C_H}{C^*} \leqslant \frac{C_H}{C_{\max}} \longrightarrow 1 \quad \text{quand} \quad \sum_{(k,l)\in\mathcal{P}} n_{k,l} \longrightarrow \infty.$$

Pour un grand nombre de paquets à router, la solution proposée est donc d'une performance équivalente à la solution optimale.

1.1.2 Intérêt du régime permanent

Au vu de l'article de Bertsimas et Gamarnik, on peut mettre en évidence plusieurs avantages d'une approche centrée sur l'étude du régime permanent :

1. Quand la minimisation du temps d'exécution d'un problème est difficile, étudier le régime permanent et chercher à maximiser son débit est parfois plus abordable.
2. Dans l'étude précédente, la relaxation en régime permanent est justifiée dès que le nombre de paquets à router est important. Nous faisons une hypothèse semblable pour les communications collectives ou les problèmes d'ordonnancement que nous étudions : le nombre de messages ou de tâches à traiter est suffisamment grand pour qu'on puisse négliger l'impact des phases d'initilisation et de terminaison de l'ordonnancement, en se concentrant sur le régime permanent qui représente la phase prépondérante.

3. L'ordonnancement fondé sur le régime permanent se caractérise par sa simplicité, et sa description compacte : dans le cas du routage des paquets, il suffit de connaître combien de paquets transitent sur chaque lien pendant une période (avec également un mécanisme simple pour router les paquets résiduels).

L'étude de Bertsimas et Gamarnik montre que le choix de la durée de la période est important. Prendre une période qui est de l'ordre de la racine carrée du temps total d'exécution ($C_{\max}$), permet à la fois (i) d'avoir des périodes suffisamment longues pour que le fait d'arrondir les valeurs rationnelles en valeur entières n'ait pas d'incidence notable sur les performances, et (ii) d'avoir un grand nombre de périodes de telle sorte que la phase de routage des paquets résiduels soit relativement petite par rapport au temps total de l'ordonnancement.

Bien qu'ingénieuse, cette méthode conduit à utiliser une période de plus en plus grande quand le nombre de messages à router augmente. Nous n'allons pas utiliser cette méthode pour déterminer la taille de la période : nous fixerons la taille de la période en fonction des paramètres de la plate-forme et de la solution du programme linéaire, et nous construirons un ordonnancement périodique qui réalise exactement le débit optimal (sans arrondir les valeurs rationnelles comme proposé ci-dessus). Ainsi nous obtenons une période indépendante du nombre de messages à router (qui peut ne pas être connu à l'avance) et un algorithme en $C_{\max} + O(1)$, au lieu d'un algorithme en $C_{\max} + O(\sqrt{C_{\max}})$ (notre $O(1)$ dépend des caractéristiques de la plate-forme, et éventuellement de l'application, mais en aucun cas du nombre de messages à router, ou de tâches à ordonnancer). En revanche, la méthode de Bertsimas et Gamarnik pour choisir la période permet de s'adapter à un nombre de messages quelconque, alors que nous nous concentrons sur le cas d'un grand nombre de messages.

Pour le problème du routage de paquets, la relaxation en régime permanent conduit à résoudre un programme linéaire, correspondant à un problème de multi-flot. Nous utiliserons la même approche, en exprimant nos différents problèmes sous forme de programmes linéaires en nombres rationnels. Résoudre un programme linéaire en rationnels est de complexité polynomiale, et il existe de nombreux logiciels permettant de le faire de façon efficace. Le premier algorithme qui a permis de montrer que cette complexité était polynomiale est la méthode des ellipsoïdes [66, 88], dont l'intérêt est surtout théorique. Nous l'utiliserons cependant pour établir la complexité de certains problèmes de communications collectives car elle présente des propriétés combinatoires intéressantes : elle nous permettra en particulier d'utiliser des programmes linéaires avec un nombre exponentiel de contraintes.

1.2 Modèles et travaux précédents

Dans cette partie, nous étudions les différents modélisations de plates-formes qui ont déjà été proposées, en discutant leurs avantages et leurs inconvénients, puis nous décrivons les modèles que nous utiliserons dans ce manuscrit.

1.2.1 Modèles homogènes

Dans le modèle utilisé par Bertsimas et Gamarnik pour étudier le routage de messages, le transfert d'un message sur une arête du graphe de communication prend un temps unité, et toutes les arêtes peuvent être utilisées simultanément. Au contraire, dans le «modèle du

téléphone» (ou *telephone call model*), un processeur ne peut participer qu'à une seule communication, soit en tant qu'émetteur, soit en tant que récepteur. Ce modèle est homogène car toutes les communications prennent un temps unité. Élaborer un ordonnancement pour un ensemble de communications dans ce modèle revient à trouver un ensemble de couplages entre les nœuds du graphe. C'est par exemple ce que font Khuller et al. [67] pour des problèmes de communications collectives.

Des modèles plus élaborés ont été imaginés pour prendre en compte les différents paramètres intervenant dans la modélisation des machines parallèles. On peut citer le modèle LogP [46, 45], et son extension LogGP au messages de grande taille [3].

Ces modèles homogènes ont permis d'étudier l'ordonnancement de communication sur des topologies structurées, comme les grilles ou les tores à n dimensions, et les hypercubes [97, 59].

Les algorithmes de communications collectives de l'implantation MPICH du standard MPI s'appuient également sur une modélisation homogène des grappes de calcul, prenant en compte latence et bande-passante et faisant une distinction entre les messages de petite taille (pour lesquels la latence est prépondérante) et les messages de grande taille [95].

1.2.2 Modèles hétérogènes

Les modèles homogènes ont ensuite été adaptés pour prendre en compte l'hétérogénéité des réseaux d'interconnexion. Une variante hétérogène du modèle LogP a été proposée : le modèle LogP paramétré [68], qui possède de nombreux paramètres difficiles à mesurer.

Banikazemi et al. [8] introduisent un modèle simple dans lequel l'hétérogénéité est caractérisée par des temps d'envoi de messages différents pour chaque processeurs. Dans ce modèle, le réseau d'interconnexion est un graphe complet, chaque processeur P_i envoie un message en un temps t_i à n'importe quel autre processeur. Les auteurs expliquent que ce modèle simple d'hétérogénéité suffit à décrire les différents coûts de communication dans une grappe hétérogène. Les communications collectives, et en particulier la diffusion, ont été étudiées avec ce modèle [58, 77, 76].

Ce modèle de coût de communication a aussi été utilisé pour étudier des réseaux constitué de deux types de nœuds : les commutateurs du réseau (ou *switch*) et les processeurs. Dans ce cas, on suppose qu'une communication entre deux processeurs occupe tous les liens sur la route entre ces processeurs (routage *wormhole*). Des résultats ont été obtenus pour les communications collectives sous ce modèle [74, 75].

Un autre modèle de communication est encore introduit dans [31, 30] pour les communications collectives : le temps nécessaire au transfert d'un message entre P_i et P_j est composé d'une latence fixe $T_{i,j}$ et d'une partie dépendant de la taille m du message $\frac{m}{B_{i,j}}$. Dans le cas de la diffusion et de nombreuses autres communications collectives, la taille des messages est la même, le temps de communication d'un message de P_i à P_j est donc $C_{i,j} = T_{i,j} + \frac{m}{B_{i,j}}$.

1.2.3 Modèles multi-port

Les modèles précédents sont une généralisation au cas hétérogène du modèle du téléphone, en ce qu'ils n'autorisent un processeur à communiquer qu'avec un seul autre processeur à un instant donné. On leur donne la caractéristique de modèle *un-port*, en considérant qu'une machine

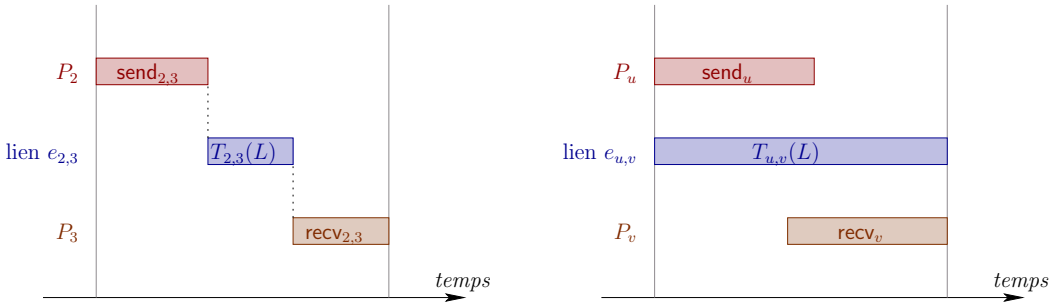
possède un seul port pour gérer les communications. Il existe également des modèles multi-port qui supposent que plusieurs communications peuvent avoir lieu simultanément sur le même processeur, sous certaines contraintes.

Dans le modèle proposé par Banikazemi et al. [9], le coût d'une communication d'un message de taille L du processeur P_i au processeur P_j être décomposé en trois parties consécutives :

$$\mathsf{send}_{i,j} + T_{i,j}(L) + \mathsf{recv}_{i,j}$$

L'occupation du processeur émetteur, du lien de communication, et du processeur récepteur est représentée sur la figure 1.1(a). Le processeur émetteur P_i est en particulier occupé pendant $\mathsf{send}_{i,j}$ unités de temps, mais peut ensuite commencer une autre émission de message alors que le premier transfert n'est pas terminé.

Bar-Noy propose un modèle très proche de celui de Banikazemi dans [11]. La seule différence est le recouvrement entre les différentes parties du temps de transfert du message, qui correspond à la figure 1.1(b) : ici, le lien de communication est occupé pendant tous le transfert, alors que dans le modèle précédent, il n'était occupé qu'après le temps d'initialisation $\mathsf{send}_{i,j}$ par le processeur émetteur et avant le temps de finalisation $\mathsf{recv}_{i,j}$ par le récepteur.



(a) Modèle de Banikazemi [9] (b) Modèle de Bar-Noy [11]

FIG. 1.1 – Décomposition du transfert d'un message pour des modèles multi-port.

A. Rosenberg propose également un modèle décomposant l'envoi d'un message en plusieurs partie, prenant en compte le processeur émetteur, les interfaces de communication et le processeur récepteur dans [85, 1]. Cependant, la complexité d'un tel modèle rend délicat l'analyse d'ordonnancements.

B. Hong et V. Prasanna proposent dans [63] un modèle beaucoup plus simple pour le multi-port : dans leur modèle *multi-port borné*, un nombre illimité de transferts de messages peuvent coexister sur un même lien, ou être initiés par un même processeur, pourvu que des contraintes de capacités soient respectées : la bande-passante totale sur un lien est limitée, de même que la bande-passante totale en entrée ou en sortie d'un processeur, ce qui correspond à la limitation de l'interface réseau de chaque machine.

1.2.4 Autres travaux précédents

Les principaux travaux théoriques d'optimisation de communications collectives sur les modèles homogènes, hétérogènes et multi-port ont été cités dans les parties précédentes. Il convient néanmoins de citer quelques autres travaux se rapportant à notre étude.

Techniques d'optimisation. Les techniques d'optimisation par programmation linéaire que nous utilisons sont principalement inspirées par l'étude de Bertsimas et Gamarnik présenté dans la partie précédente. Nous utilisons également au chapitre 7 une technique d'arrondi d'une solution rationnelle d'un programme linéaire vers une solution entière qui s'inspire des travaux de Coudert et Rivano [44]. Bar-Noy et al.[12] proposent également une technique d'arrondi vers une solution entière pour l'ordonnancement de tâches avec date d'arrivée et date limite d'exécution.

L'utilisation concurrente de plusieurs arbres afin d'optimiser le débit d'une diffusion a d'abord été étudiée sur les topologies homogènes. On peut par exemple citer le cas de l'hypercube de dimension n, dans lequel on peut construire n arbres couvrant arêtes-disjoints enracinés en un processeur, ce qui permet une diffusion très efficace. Plus récemment dans le contexte des réseaux pair-à-pair, ce principe a été utilisé dans Splitstream [38], en cherchant à minimiser l'occupation des nœuds et leur degré sortant, afin de maximiser le débit de la diffusion. L'étude utilisant ce principe d'arbres de diffusion concurrents qui est la plus proche de la notre est sans doute celle de den Burger et al.[32], qui utilise plusieurs arbres pour effectuer une diffusion sur des grilles de calcul. Remarquant que le nombre d'arbres de diffusion à considérer est exponentiel, les auteurs de cette étude commencent par sélectionner un ensemble d'arbres intéressants, générés par plusieurs heuristiques de recherche d'arbre de meilleur débit. Ils utilisent ensuite cet ensemble restreint d'arbres d'une façon très proche de la notre, en écrivant un programme linéaire qui a pour variables les débits des différents arbres, et pour contraintes les limitations de bande-passante mesurées dans le réseau. Ces travaux sont validés par des expérimentations, montrant que le débit total obtenu par leur implantation est supérieur à celui des bibliothèques de communication actuelles.

Bibliothèques de communication. Il existe de nombreuses bibliothèques de communication implantant des primitives de communications collectives, plus ou moins adaptées à un environnement hétérogène.

PVM est une bibliothèque de communication populaire [52], mais elle ne comprend que peu de communications collectives. Leur implémentation est laissée à la charge du programmeur, qui peut ainsi les adapter à l'environnement de calcul qu'il vise, mais ceci représente une grosse charge de travail.

Standard MPI Il définit précisément de nombreuses primitives de communications. Une implémentation récente, nommée MPICH-G2 [64] est adaptée à un environnement de calcul distribué, de type grille : on commence par regrouper les nœuds de calcul qui sont sur les mêmes sous-réseaux en se basant sur leurs possibilités de communication, puis on utilise ce découpage en niveaux pour une effectuer les communications de façon hiérarchique, d'abord entre les sous-réseaux différents, puis ensuite à l'intérieur d'un même sous-réseau. Cependant la segmentation des messages en vue d'un pipeline des opérations est toujours en projet pour une prochaine version.

ECO est une bibliothèque dédiée aux communications collectives sur environnement hétérogène [78]. Comme la précédente, elle commence par regrouper les machines en sous-réseaux (proches des réseaux locaux) en mesurant la latence entre deux machines, puis effectue également un traitement hiérarchique sur ces deux niveaux. À l'intérieur des sous-réseaux, le traitement est différencié selon les caractéristiques de ceux-ci, ce qui conduit par exemple à des arbres de diffusion de largeur différente selon le type de réseau.

MagPIe est une autre bibliothèque qui implémente des communications collectives pour MPI sur un environnement hétérogène [69]. De la même façon que les précédentes, elle utilise un traitement en couches. La limitation à deux couches est justifiée par le fait que les communications longue-distance apportent toujours une latence importance et que le gain d'une approche avec plus de couches, par exemple dans le cas d'un ensemble de grappes de machines parallèles (donc à 3 niveaux), est négligeable devant la latence incompressible d'un transfert sur un lien longue-distance. Les auteurs cherchent donc à effectuer le minimum de transferts sur ce type de lien, et obtiennent ainsi des gains significatifs par rapport à l'implémentation commune MPICH.

1.2.5 Choix d'un modèle

Le choix d'un modèle de communication résulte d'un compromis entre réalisme, instanciabilité et maniabilité pour la conception d'algorithmes. Le modèle de Rosenberg est probablement très proche de la réalité, mais très difficile à instancier, et concevoir des algorithmes pour des schémas de communications complexes parait extrêmement difficile dans ce modèle. En revanche, le modèle homogène du téléphone, sans difficulté d'instanciation, manque cruellement de réalisme pour les plates-formes que nous voulons étudier. Cependant, le fait que l'optimisation du temps de diffusion d'un message soit NP-complet dans ce modèle met en évidence le caractère intrinsèquement difficile de cette opération.

Le modèle multi-port borné utilisé par Hong et Prasanna [63] paraît intéressant car très simple, mais ne nous semble pas assez réaliste en ce qui concerne l'existence de plusieurs transferts simultanés : Saif et Parashar [86] ont montré que les envois asynchrones de MPI étaient effectués de façon sérialisée dès que la taille des données excédait quelque méga-octets, et ce pour deux implémentations populaire de MPI : MPICH et le MPI d'IBM sur le SP2. Nous préférerons donc utiliser des modèles un-port, mais nous utiliserons occasionnellement ce modèle multi-port.

Modèles utilisés dans ce manuscrit

Nous utiliserons par la suite principalement deux variantes du modèles un-port hétérogène, sans latence : les modèles *un-port bidirectionnel avec recouvrement*, et *un-port unidirectionnel avec recouvrement*.

Nous considérons que la plate-forme est modélisée par un graphe $G = (V, E)$ dont les sommets sont des machines, assimilées à des processeurs. Nous notons P_i les sommets de V, et nous préférerons souvent la notation (i, j) à la place de (P_i, P_j) pour désigner les arêtes du graphe, afin de ne pas alourdir les notations.

Dans tous les cas, nous choisissons de modéliser la durée du transfert d'un message de taille N entre les processeurs P_i et P_j par

$$N \times c_{i,j},$$

où $c_{i,j}$ représente l'inverse de la bande passante du lien de communication (i, j), c'est-à-dire le temps nécessaire à la transmission d'un message de taille unité sur le lien (i, j)

Dans le modèle *un-port bidirectionnel avec recouvrement*, l'interaction entre différentes communications simultanées est modélisée comme suit :

- Si le processeur P_i commence au temps t à envoyer un message de taille N au processeur P_j, alors il ne peut initier une autre émission avant la fin de la communication, c'est-à-dire avant $t + N \times c_{i,j}$, et le processeur P_j ne peut commencer à recevoir un autre message avant cette date (modèle *un-port*).
- Par contre, pendant ce temps, le processeur P_i peut tout à fait recevoir un message d'un autre processeur, et P_j peut également envoyer un message à un autre processeur (modèle *bidirectionnel*). On peut imaginer que cela représente un processeur muni de deux ports de communication, un pour les envois et un pour les réceptions.
- Pendant la durée de la communication, P_i et P_j peuvent continuer leur activité de calcul sans être ralenti par la communication (*recouvrement total* entre les calculs et les communications).

Dans le modèle un-port *unidirectionnel* avec recouvrement, on considère qu'il n'y a qu'une seule interface de communication pour chaque processeur, en charge des communications sortantes et entrantes. Ainsi un processeur P_i ne peut pas simultanément envoyer un message à un processeur P_j et recevoir un message d'un processeur P_k, mais ces communications doivent être sérialisées.

Calculs Pour étudier des problèmes mêlant communications et calculs, nous modélisons le temps d'exécution d'une tâche T sur un processeur P_i par

$$z_i \times \delta_T,$$

où z_i est le temps de calcul d'une tâche de taille unitaire sur le processeur P_i, et δ_t est la taille de la tâche T.

Latence Bien que la latence puisse être un paramètre important sur les réseaux de communication, notre modèle ne la prend pas en compte. Nous faisons ce choix principalement afin de pouvoir concevoir des algorithmes de communication de débit optimal. Cependant, pour des séries d'un grand nombre de même opérations, tous les messages ont une même taille, qui est fixée par l'application, et non par l'ordonnancement : par exemple dans le cas de tâches indépendantes, si un ordonnancement décide d'envoyer trois tâches sur un même lien, ces envois seront effectués de manière séquentielle, et la latence devra elle aussi être comptée trois fois. Dans ce cas, nous supposons que le coût $c_{i,j}$ de transmission d'un message sur cette arête prend déjà en compte cette latence. Comme nous supposons que la granularité des messages est fixée par l'application, nous pouvons rassembler les différents composants du coût de communication (latence, bande-passante) dans ce seul paramètre $c_{i,j}$.

Autres modèles utilisés Nous utiliserons également le modèle *multi-port borné* dans le chapitre 5, que nous présenterons à cette occasion. D'autre part, nous étudierons au chapitre 7 l'ordonnancement d'applications sur une plate-forme distribuée organisée comme une collection de grappes de calcul, et nous élaborerons à cette occasion un modèle plus spécifique.

1.3 Contribution et résumé des résultats obtenus

La principale contribution de cette thèse est le développement d'un cadre général d'étude pour l'ordonnancement en régime permanent sur plates-formes hétérogènes. Nous présentons

ce schéma général en l'utilisant pour étudier diverses opérations de communications collectives (diffusion, multicast, distribution, réduction, calcul des préfixes) selon plusieurs modèles de communication (un-port mono- et bi-directionnel, multi-port borné). Nous avons pu d'une part établir la complexité des différentes primitives étudiées selon différents modèles de communication, et d'autre part concevoir des algorithmes efficaces pour résoudre ces problèmes dans certains cas (modèle de communication un-port bidirectionnel).

Dans la suite de cette partie, nous résumons les différents résultats obtenus au cours de cette thèse.

1.3.1 Communication collectives en régime permanent

On s'intéresse ici aux communications induites par l'exécution d'une application distribuée sur une plate-forme de calcul hétérogène. Ces communications peuvent souvent être rassemblées sous la forme de primitives de communications collectives, comme par exemple la diffusion de données : une des machines transmet à toutes les autres une copie d'une donnée qu'elle possède. Pour qu'une application «mérite» d'être exécutée sur une plate-forme à grande échelle, il est probable que le volume de données à communiquer soit important. Pour effectuer ces transferts d'importants volumes de données, nous les découpons en une série d'un grand nombre de communications de taille plus modeste : la diffusion d'une donnée de grande taille sera alors transformée en une série d'un grand nombre de diffusion de messages de petite taille, que l'on cherche à effectuer de façon pipelinée.

Les différents schémas de communications collectives que nous étudions sont les suivants :

Distribution de données Un processeur P_{source} distribue p données à un ensemble V_{cibles} de p processeurs, chacun recevant une donnée différente.

Diffusion de données La plus commune des opérations de communications collectives : un processeur P_{source} doit communiquer une même valeur à tous les autres processeurs de la plate-forme.

Diffusion restreinte Plus générale que la précédente : un processeur P_{source} doit communiquer une même valeur à un sous-ensemble V_{cibles} des processeurs de la plate-forme.

Réduction On définit un ensemble de processeurs participants $\mathcal{R} = \{P_{r_1}, \ldots, P_{r_N}\}$. Chaque processeur P_i de cet ensemble possède une valeur v_i, obtenue par exemple comme résultat d'un calcul local. On cherche à calculer la réduction $v_1 \oplus v_2 \oplus \cdots \oplus v_N$, où $\oplus$ est une opération associative et non-commutative. Le résultat doit au final se trouver sur un seul processeur cible P_{cible}.

Calcul de préfixes Comme pour la réduction chaque processeur participant $P_{r_i} \in \mathcal{R}$ possède une valeur v_i. On cherche à ce que chaque processeur P_{r_i} connaisse la réduction partielle $v_1 \oplus v_2 \oplus \cdots \oplus v_i$ (en particulier P_{r_N} doit connaître le résultat de la réduction totale). De même que pour la réduction, $\oplus$ est une opération associative et non-commutative.

Pour chacun de ces schémas de communication, nous nous intéressons à la version pipelinée du problème. Par exemple dans le problème «série de distributions de données», on considère que le processeur source possède $N \times p$ valeurs $a_{i,j} (i = 1 \ldots p, j = 1 \ldots N)$, qu'il doit distribuer de telle sorte que P_i reçoive les valeurs $a_{i,1}, \ldots a_{i,N}$. Dans notre étude, nous considérons que le nombre N d'opérations dans la série est grand, et nous nous intéresserons surtout au comportement asymptotique quand N tend vers l'infini.

D'autre part, comme nous considérons toujours le problème d'une série d'un même schéma de communication, nous ne préciserons plus «série de distribution de données», mais nous parlerons simplement du problème de distribution de données, de même pour les autres problèmes.

Les résultats obtenus pour l'étude des communications collectives pipelinées, présentés dans la première partie de ce manuscrit, sont les suivants :

- Pour les modèles un-port unidirectionnel et bidirectionnel, nous montrons qu'un ordonnancement réalisant le débit optimal peut être décrit de façon compacte, c'est-à-dire sous la forme d'un ensemble pondéré de schémas d'allocation et d'un ensemble pondéré de couplages de communications, et ce quelque soit le problème considéré (chapitres 2 et 3).
- Pour le modèle un-port unidirectionnel, nous montrons que la complexité des problèmes de diffusion, de distribution de données et de réduction est polynomiale (chapitre 3). Cette méthode s'appuyant sur la méthode d'optimisation d'un programme linéaire par la méthode des ellipsoïdes, elle ne fournit par pour autant d'algorithme efficace en pratique pour résoudre ces problèmes.
- Pour le modèle un-port bidirectionnel, nous proposons des algorithmes efficaces pour résoudre le problème de la diffusion, de la distribution de données, et de la réduction. Nous montrons également que la diffusion restreinte et le calcul parallèle des préfixes sont des problèmes NP-complets, et que ces résultats de NP-complétude s'étendent au modèle un-port unidirectionnel (chapitre 4).
- Nous avons illustré ces résultats par des simulations pour le problèmes de la réduction (partie 4.3 du chapitre 4), et par des expérimentations sur des plates-formes réelles pour la diffusion (chapitre 5).

Ces différents travaux sur les communications collectives ont fait l'objet des publications [1, 3, 5, 13, 15, 16, 17].

1.3.2 Tâches indépendantes et graphes de tâches en régime permanent

Nous avons également étudié des problèmes d'ordonnancement plus classiques sous l'angle de l'optimisation du débit en régime permanent. Nous avons en particulier étudié l'exécution de tâches indépendantes sur plate-forme hétérogène, pour lequel nous sommes également capables de calculer le débit optimal, et de décrire un ordonnancement réalisant ce débit.

Il était ensuite naturel de vouloir prendre en compte des tâches avec dépendances, c'est pourquoi nous nous sommes intéressés à l'exécution d'une série de graphes de tâches. Nous avons montré que dans le cas général, le calcul du débit optimal pour ce problème est NP-complet. Cependant, pour le cas particulier des graphes de tâches avec une profondeur de dépendance bornée, nous sommes capables de calculer le débit optimal et de reconstruire un ordonnancement qui le réalise.

Ces travaux, sur les tâches indépendantes et les graphes de tâches, ont été réalisés en collaboration avec Arnaud Legrand, et ont été présentés en détail dans son manuscrit de thèse [72], ainsi que dans l'Habilitation à diriger des recherches d'Olivier Beaumont [15]. Nous ne reproduirons donc pas ces résultats ici, préférant nous concentrer sur l'étude des communications collectives.

Ces travaux ont fait l'objet des publications [4, 6, 18, 20].

1.3.3 Extension aux applications multiples

Nous nous sommes également intéressés à l'ordonnancement pour des plates-formes de calcul distribuées à grande échelle. Les applications utilisant de telles plates-formes présentent en général un fort degré de parallélisme, demandent une grande puissance de calcul et un temps d'exécution assez long. Ce sont donc de très bons candidats pour la mise en pratique des techniques d'ordonnancement fondées sur l'optimisation du régime permanent. D'autre part, une caractéristique de ces plates-formes est que leur puissance est partagée entre plusieurs applications concurrentes. Nous présentons deux études de cas pour l'ordonnancement d'applications multiples sur de telles plates-formes.

- Nous étudions dans un premier temps l'ordonnancement de plusieurs applications composées de tâches indépendantes sur une plate-forme de type «maître-esclaves». Nous étudions l'éventualité d'un ordonnanceur centralisé, en montrant comment calculer le débit optimal et construire un ordonnancement périodique qui le réalise. Nous proposons également des méthodes heuristiques pour élaborer des ordonnanceurs décentralisés. Ces travaux, présentés au chapitre 6, font l'objet de la publication [12].
- Dans un second temps, nous abandonnons le modèle de plate-forme en «maître-esclaves» pour élaborer un modèle de plate-forme distribuée prenant en compte les spécificités des réseaux d'interconnexion des grilles de calcul. Sur ce modèle, nous étudions l'ordonnancement d'applications constituées de tâches divisibles. Après avoir montré que le problème d'ordonnancement correspondant est NP-complet, nous proposons des méthodes heuristiques pour le résoudre, que nous testons par simulation. Nous présentons ces travaux dans le chapitre 7, qui font également l'objet des publications [2, 7].

1.3.4 Autres problèmes d'ordonnancement abordés

Au cours de cette thèse, nous avons été amenés à étudier d'autres problèmes d'ordonnancement. Comme ils s'écartent du thème principal de cette thèse, nous ne les détaillerons pas dans ce manuscrit.

Tâches indépendantes avec contraintes de mémoire Nous nous sommes intéressés à l'ordonnancement de tâches indépendantes en présence de mémoire limitée : dans les travaux précédemment évoqués, nous supposons avoir à notre disposition sur chaque processeur une mémoire pouvant contenir une nombre illimité de tâches à traiter. Nous montrons que si cette hypothèse n'est pas vérifiée, alors un grand nombre de problèmes d'ordonnancement que l'on savait résoudre, en particulier l'optimisation du débit, deviennent NP-complets. Par contre, nous présentons des résultats de simulation montrant que lorsqu'un seuil dans la taille de la mémoire est franchi, alors on peut espérer atteindre le débit optimal calculé sans contrainte de mémoire. Ces travaux font l'objet de la publication [14].

Tâches divisibles avec messages de retour Nous avons également apporté une contribution à la théorie des tâches divisibles, en étudiant l'ordonnancement de telles tâches avec messages de retour. La plupart des études précédentes dans ce domaine prennent en compte les communications impliquées par les données nécessaires aux calculs, mais pas les messages de retour contenant les résultats. Or il n'est pas rare que les résultats soient d'une taille comparable aux données, et doivent être rassemblés après le calcul. Nous montrons que prendre en compte

les messages de retour rend les problèmes d'ordonnancement significativement plus difficiles. En particulier, nous exhibons le premier cas (à notre connaissance) d'ordonnancement optimal pour des tâches divisibles sous un modèle sans latence, où tous les processeurs ne prennent pas part au calcul.

Ces travaux font l'objet des publications [8, 10].

Ordonnancement pour les réseaux En collaboration avec l'équipe RESO du LIP, nous avons également étudié la réservation de ressources dans les réseaux d'interconnexion des grilles de calcul, en particulier pour les problèmes liés à la contention des connexions au niveau de points d'entrée et de sortie du cœur du réseau. Nous montrons que le problème d'optimisation associé est NP-complet, et nous proposons des stratégies heuristiques pour le résoudre.

Ces travaux ont fait l'objet des publications [9, 11].

Pair-à-Pair En collaboration avec Anne-Marie Kermarrec et Étienne Rivière de l'IRISA, nous nous sommes intéressés à un tout autre sujet : l'élaboration d'un réseau pair-à-pair centré sur les objets, dans lequel chaque objet est décrit par un ensemble d'attributs. Les objets sont placés dans un espace euclidien. Ils sont liés de manière à ce que les objets avec des attributs proches soient eux-mêmes proches dans le réseau. Ceci est réalisé par le calcul d'un diagramme de Voronoï. De plus, des mécanismes de routage efficace de type «petit-monde» généralisent le modèle de Kleinberg sur la grille à notre réseau. Cet ébauche de protocole est destinée à être le support de mécanismes de recherche complexes, comme des requêtes par plage de valeurs.

Ces travaux ont fait l'objet de la publication [21].

En résumé, nous développerons dans ce manuscrit l'étude des communications collectives en régime permanent (de la partie 1.3.1), ainsi que l'étude de l'ordonnancement d'applications multiples (de la partie 1.3.3), en renvoyant aux publications pour les autres sujets abordés.

Première partie

Communications collectives

Chapitre 2

Approche générale

Dans ce chapitre, nous présentons les notions qui interviennent dans la recherche d'ordonnancements efficaces pour des problèmes de communications collectives pipelinées. Nous nous écartons temporairement de la méthode introduite par Bertsimas et Gamarnik. Nous verrons dans le chapitre suivant que la méthode présentée ici est une façon de généraliser leur approche. Bien que cette cette méthode générale ne conduise pas à une résolution efficace, nous la présentons dans un premier temps car elle permet de décrire simplement et d'étudier la complexité de nombreux problèmes de communications collectives, pour différents modèles de communication.

Dans ce chapitre, nous choisissons de présenter les notions utilisées d'abord de façon assez informelle, en nous aidant d'exemples. La formalisation sera introduite au long de ce chapitre et dans le suivant, avec les méthodes de résolution associées. Nous espérons que cette présentation aura l'avantage de la clarté.

2.1 Structure des communications : schémas d'allocation

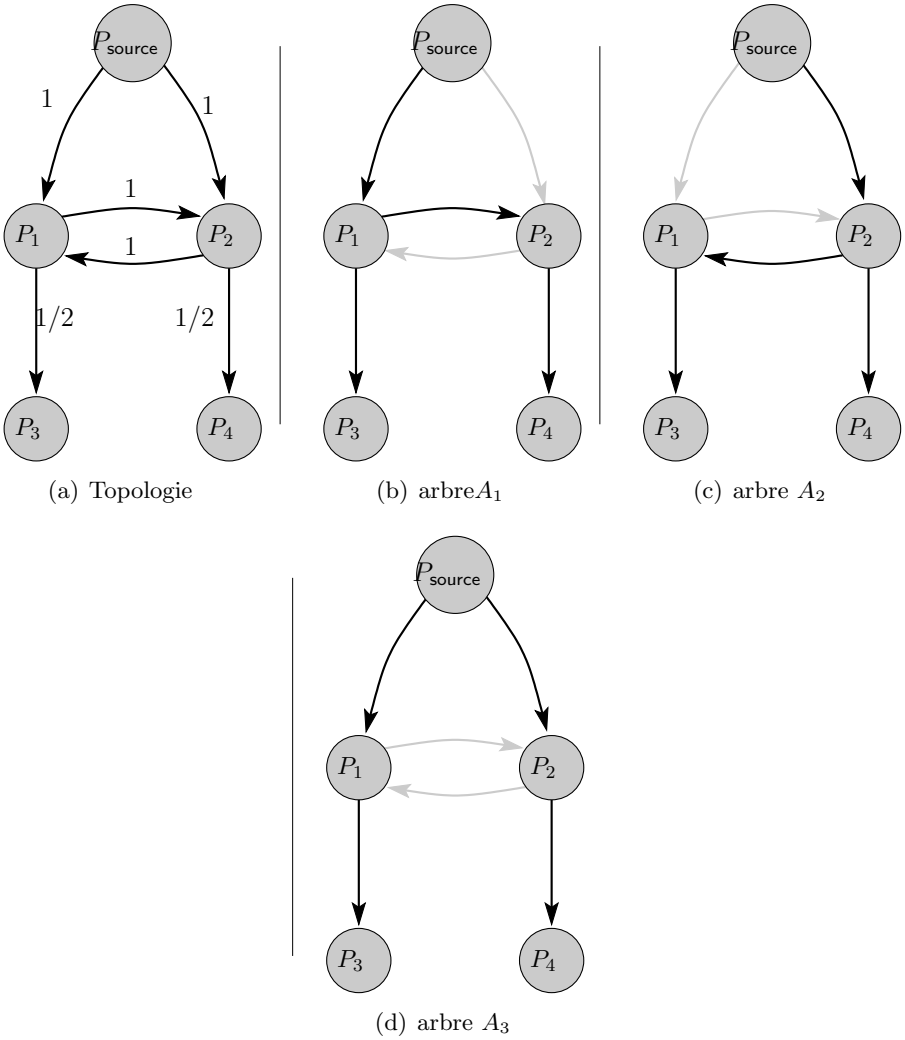
2.1.1 Pour la diffusion

Nous allons utiliser tout au long de ce chapitre l'exemple suivant : étant donné le graphe de plate-forme décrit par la figure 2.1(a), nous souhaitons effectuer une diffusion pipelinée d'un grand nombre de messages depuis le processeur P_{source} vers tous les autres processeurs. Nous étudions ce problème sous le modèle un-port bidirectionnel (un processeur peut simultanément émettre et recevoir) et sous le modèle un-port unidirectionnel (un processeur ne peut qu'émettre ou recevoir à un instant donné).

Considérons un message m de la série de messages que la source doit diffuser. On s'intéresse au parcours qu'effectue ce message dans le réseau, afin de caractériser la structure de l'ordonnancement sous la forme de schémas d'allocation.

Définition 2.1 (Schéma d'allocation)**.** *On appelle schéma d'allocation pour une primitive de communications collectives donnée l'ensemble des tâches (transferts ou calculs) qui interviennent pour le traitement d'une opération de la série.*

Tout ce qu'on sait pour l'instant c'est que le message m est émis par la source et que chaque processeur le reçoit. Les processeurs qui ont reçu ce message sont capables de le renvoyer,



(a) Topologie (b) arbreA_1 (c) arbre A_2

(d) arbre A_3

FIG. 2.1 – Exemple de diffusion sur une petite plate-forme : topologie et arbres de diffusion possibles. Les arêtes du graphe de topologie sont étiquetées par les coûts de communication $c_{i,j}$.

à destination d'un ou plusieurs autres processeurs voisins. Sans perte de généralité, on peut supposer qu'un processeur P_i ne reçoit ce message m qu'une seule fois ; sinon cela signifie qu'un processeur P_i reçoit le message depuis P_j, puis ensuite depuis P_k, on peut alors supprimer le transfert $P_k \rightarrow P_i$ sans perturber les autres communications et pour une occupation des ressources strictement plus petite. Si on considère le trajet du message m dans le graphe, il forme donc un arbre dirigé, enraciné en P_{source}. Les figures 2.1(b), 2.1(c) et 2.1(d) représentent les trois arbres de diffusion que le message peut emprunter sur la plate-forme précédente. Un arbre de diffusion est un sous-graphe du graphe de plate-forme, puisque les messages ne peuvent être transmis que par des arêtes du graphe initial et il couvre tous les sommets puisque chaque processeur doit recevoir une copie du message. Ainsi, les schémas d'allocation pour le problème de la diffusion sont des arbres couvrants dirigés, enracinés en P_{source} et sous-graphes du graphe de plate-forme.

Le problème souvent étudié pour les communications collectives et en particulier pour la diffusion, est celui de la minimisation du temps d'exécution d'une opération. Pour la diffusion d'un message, on cherche ainsi généralement à trouver le meilleur arbre, c'est-à-dire l'arbre qui conduit à minimiser le temps entre le moment où la source émet la première copie du message et le moment où le dernier processeur reçoit une copie. Ce problème est connu pour être difficile, même sous le modèle simple "du téléphone", lorsque les coûts de communication sont homogènes [51, problème ND49]. Notre objectif est différent : nous cherchons à optimiser le débit de la diffusion d'une série de messages. Il est tout à fait possible que deux messages de la série empruntent des arbres différents et ceci est même souhaitable : on imagine qu'en utilisant des arbres à arêtes disjointes, on pourra diffuser plusieurs messages simultanément et donc améliorer le débit.

Reprenons notre exemple décrit sur la figure 2.1. On peut par exemple vouloir utiliser l'arbre A_1 de la figure 2.1(b) pour les messages impairs et l'arbre A_2 de la figure 2.1(c) pour les messages pairs. On appelle $\mathcal{A} = \{A_1, \ldots, A_{|\mathcal{A}|}\}$ l'ensemble des schémas d'allocation possibles, donc l'ensemble des arbres de diffusion pour notre problème. On note x_a le débit de messages diffusés en régime permanent en utilisant l'arbre A_a. En étudiant le parcours des messages à diffuser, on peut ainsi caractériser la structure générale d'un ordonnancement sous la forme d'une collection pondérée de schémas d'allocation, $\{(A_a, x_a), A_a \in \mathcal{A}\}$ avec $\sum x_a = \rho$ le débit total obtenu.

Notons également que $\mathcal{A}$ est potentiellement de grande taille : il peut exister un nombre exponentiel en n d'arbres de diffusion dans une plate-forme à n nœuds. Nous verrons au chapitre suivant comment nous restreindre à des ordonnancements utilisant un petit nombre d'arbres et qui peuvent ainsi être décrits de façon plus compacte.

2.1.2 Pour les autres primitives

Avant d'étudier comment s'organisent les transferts intervenants lors de l'utilisation des schémas d'allocation, étudions ce que que deviennent ces schémas pour les autres primitives de communications collectives.

Diffusion restreinte. Le cas le plus immédiat est celui de la diffusion restreinte : le problème est identique, sauf que les destinations des messages sont un sous-ensemble strict V_{cibles} de l'ensemble de processeurs. En appliquant le même raisonnement, on peut caractériser les schémas

d'allocation pour cette opération : ce sont des arbres dirigés enracinés en P_{source}, sous-graphes du graphe de plate-forme et couvrant les sommets de V_{cibles}. Ces arbres couvrant un sous-ensemble donné de sommets sont appelés arbres de Steiner et la recherche d'un tel arbre de poids minimal[1] est un problème NP-complet [65]. Même si ce résultat ne peut être directement transcrit pour notre problème, nous verrons au chapitre suivant que l'optimisation du débit de la diffusion restreinte est également un problème difficile.

Distribution de données. Dans le cas de la distribution de données, le processeur source P_{source} souhaite envoyer des messages distincts à un ensemble de processeurs cibles V_{cibles}. Pour $k \in V_{\mathsf{cibles}}$ nous appelons messages de type k les messages que la source envoie à destination du processeur P_k. Pour étudier la forme des schémas d'allocation, nous devons nous intéresser au parcours d'un message émis par la source vers chaque destination, soit $|V_{\mathsf{cibles}}|$ messages au total. Sans perte de généralité, nous pouvons considérer qu'aucun de ces messages n'est dupliqué ; si ce n'est pas le cas pour une destination P_i, nous sélectionnons parmi les transferts de messages de type i uniquement ceux qui forment une route sans cycle de P_{source} à P_i (une telle route existe nécessairement puisque P_i reçoit le message). Un schéma d'allocation pour la distribution de données est donc une collections de routes de P_{source} à chaque processeur destination de V_{cibles}, empruntant les arêtes du graphe de plate-forme. Chaque schéma d'allocation comporte exactement une route de P_{source} à P_i, pour toute destination P_i : $\mathcal{A} = \mathcal{R}_1 \times \mathcal{R}_2 \times \cdots \times \mathcal{R}_{V_{\mathsf{cibles}}}$ en notant $\mathcal{R}_i$ l'ensemble des routes sans cycles de P_{source} à P_i, pour $i \in V_{\mathsf{cibles}}$.

Notons que deux de ces routes peuvent emprunter le même lien de communication : nous ne nous intéressons pas ici aux contraintes de ressources mais à la forme générale des solutions.

Réduction. Le cas de la réduction est un peu plus complexe. Comme pour les autres communications, considérons une seule opération de réduction : chaque processeur $P_{r_i} \in \mathit{Reds} = \{P_{r_0}, \ldots, P_{r_N}\}$ possède initialement une valeur v_i. Nous cherchons à calculer la valeur réduite $v = v_0 \oplus v_1 \oplus \cdots \oplus v_N$, où $\oplus$ est associatif, mais pas nécessairement commutatif. Le résultat doit au final se trouver sur le processeur P_{cible}. Cette opération est plus complexe que les précédentes à cause des calculs qui interviennent lors de la réduction. Pour $0 \leqslant k \leqslant m \leqslant N$, nous notons $v_{[k,m]}$ le résultat partiel issu de la réduction des valeurs $v_k, \ldots, v_m$:

$$v_{[k,m]} = v_k \oplus \cdots \oplus v_m.$$

Les valeurs initiales $v_i = v_{[i,i]}$ vont être assemblées en résultats partiels jusqu'à ce que la valeur finale $v = v_{[0,N]}$ soit atteinte. Comme $\oplus$ est un opérateur associatif, $v_{[k,m]}$ peut être calculé comme suit :

$$v_{[k,m]} = v_{[k,l]} \oplus v_{[l+1,m]} \quad \text{pour } 0 \leqslant k \leqslant l < m \leqslant N$$

Nous notons $T_{k,l,m}$ la tâche de calcul associée à l'opération $v_{[k,l]} \oplus v_{[l+1,m]}$.

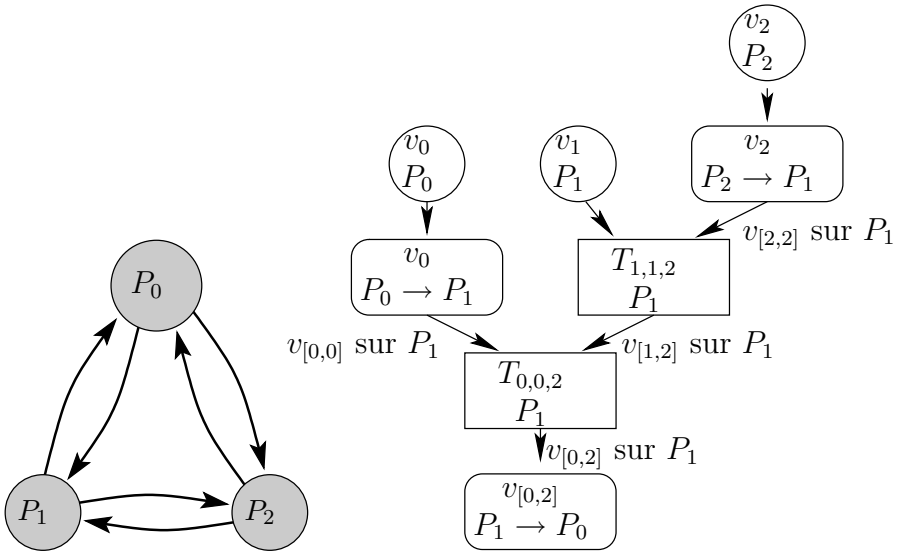
Considérons une petite plate-forme constituée de trois processeurs P_0, P_1 et P_2 et complètement connectée (voir figure 2.2(a)). Tous les processeurs possèdent une valeur initialement ($P_{r_i} = P_i$) et le résultat doit se trouver au final sur $P_{\mathsf{cible}} = P_0$. Une méthode pour réaliser la réduction de $\{v_0, v_1, v_2\}$ consiste en les opérations suivantes :

1. P_2 envoie sa valeur v_2 à P_1
2. P_1 calcule la réduction partielle $v_{[1,2]} = v_1 \oplus v_2$ (tâche $T_{1,1,2}$)

[1] Le poids d'un arbre est la somme des poids des arêtes de l'arbre.

3. P_0 envoie sa valeur v_0 à P_1,
4. P_1 calcule le résultat final $v_{[0,2]} = v_0 \oplus v_{[1,2]}$ (tâche $T_{0,0,2}$),
5. P_1 envoie le résultat final $v = v_{[0,2]}$ à P_0.

Ces tâches s'organisent naturellement en un arbre, comme illustré sur la figure 2.2(b) : le résultat d'une ou de deux d'entre elles est utilisé comme entrée de la tâche suivante.



(a) Topologie (b) Un arbre de réduction

FIG. 2.2 – Exemple de réduction sur trois processeurs

On peut définir plus formellement un schéma d'allocation pour la réduction comme
– un ensemble de messages localisés sur des processeurs :

$$\mathcal{M} \subset \{v_{[k,m]},\ 0 \leqslant k \leqslant m \leqslant N\} \times V$$

– et un ensemble de taches localisées (calcul ou transfert) :

$$\mathcal{T} \subset \underbrace{\{T_{k,l,m},\ 0 \leqslant k \leqslant l < m \leqslant N\} \times V}_{\text{calcul sur un processeur}} \cup \underbrace{\{v_{[k,m]},\ 0 \leqslant k \leqslant m \leqslant N\} \times E}_{\text{transfert sur un lien}}$$

tels que des contraintes de précédence sont satisfaites :
– si un message est présent sur un nœud, alors il existe une tâche qui l'a produit :

$$\begin{aligned}&\text{si } (v_{[k,m]}, P_i) \in \mathcal{M} \text{ et } k \neq m \text{ ou } k = m \neq i,\\ &\quad \text{alors il existe } (T_{k,l,m}, P_i) \in \mathcal{T}(\text{ pour un } l \text{ tel que } k \leqslant l < m)\\ &\quad \text{ou } (v_{[k,m]}, (j,i)) \in \mathcal{T}(\text{ pour un } j \text{ tel que } (j,i) \in E)\end{aligned}$$

– les entrées d'une tâche doivent être disponibles :

$$\begin{aligned}&\text{si } (T_{k,l,m}, P_i) \in \mathcal{T} \text{ alors } (v_{[k,l]}, P_i) \in \mathcal{M} \text{ et } (v_{[l+1,m]}, P_i) \in \mathcal{M}\\ &\text{si } (v_{[k,m]}, (i,j)) \in \mathcal{T} \text{ alors } (v_{[k,l]}, P_i) \in \mathcal{M}\end{aligned}$$

PRIMITIVE DE COMMUNICATION	SCHÉMA D'ALLOCATION CORRESPONDANT
diffusion de données	arbre couvrant dirigé, enraciné en P_{source}, sous-graphe de la plate-forme
diffusion restreinte	arbre dirigé, couvrant les sommet de V_{cibles}, enraciné en P_{source}, sous-graphe de la plate-forme
distribution de données	ensemble de routes dans le graphe de plate-forme : pour toute cible $P_i \in V_{\mathsf{cibles}}$, une route de P_{source} à P_i
réduction	arbre de réduction

TAB. 2.1 – Schémas d'allocation pour les communications collectives

Un couple de tels ensembles vérifiant ces contraintes est appelé **arbre de réduction**.

Les schémas d'allocation de chacune des primitives de communication collective sont résumés dans le tableau 2.1.

On peut remarquer que ce cadre de travail permet de s'intéresser à des primitives de communications complexes, mêlant calculs et communications, du moment que l'on sait décrire les schémas d'allocation associés. Notons également que les schémas d'allocation pour la réduction sont très proches de la notion d'allocation pour un graphe de tâches. Elle est même plus générale puisqu'ici, plusieurs tâches donnent le même message en sortie, alors qu'une seule d'entre elle est nécessaire pour une opération de réduction : pour construire le message $v_{[k,m]}$, il suffit d'effectuer une des tâches $T_{k,l,m}$, pour un $l \in [k, m[$, et tous les $v_{[k,m]}$ ne sont pas calculés lors d'une opération de réduction.

2.2 Organisation des communications : schémas de communication

La description de la structure d'un ordonnancement sous forme de schémas d'allocation pondérés par leur débit nous donne une information de structure et de localisation des transferts de données et des calculs éventuellement impliqués dans l'opération de communication collective. Avec cette information, nous savons *où* sont faites les choses. Nous nous intéressons maintenant à leur organisation dans le temps : à quels instants ces différents transferts et ces différentes tâches ont lieu. L'organisation des transferts est soumise à des contraintes à cause de la contention au niveau des arêtes (due à leur capacité limité) et au niveau des nœuds (due au modèle un-port par exemple).

Pour étudier l'organisation des communications dans le temps, nous nous plaçons en régime permanent, afin de relâcher les contraintes de précédence qui indiquent qu'un transfert doit avoir lieu avant un autre (par exemple qu'un message m doit être arrivé avant d'être retransmis). Nous supposons qu'un schéma d'allocation de débit non nul sera utilisé un grand nombre de fois, car le nombre d'opérations (diffusion de messages ou autre) à effectuer est grand. Dans cette partie, nous cherchons à construire un motif qui sera ensuite utilisé pour élaborer un ordonnancement périodique. On veut donc organiser les communications nécessaires aux schémas d'allocation dans ce motif, et nous verrons dans la partie suivante comment reconstruire un ordonnancement respectant les contraintes de précédence en utilisant ce motif. Pour la suite, nous étudions les contraintes sur les communications en nous appuyant sur le cas de la diffusion, en reprenant l'exemple de la partie précédente (figure 2.1).

2.2.1 Contraintes pour les communications sous le modèle un-port unidirectionnel

Nous nous intéressons tout d'abord au modèle un-port unidirectionnel. Rappelons que dans ce modèle, un processeur peut à tout instant communiquer avec seulement un autre processeur, que ce soit pour un envoi ou pour une réception. L'ensemble des communications qui peuvent être effectuées simultanément est ainsi un sous-ensemble des liens de communication tel qu'un processeur est couvert par au plus un des liens sélectionnés, c'est-à-dire un couplage dans le graphe de la plate-forme. Pour la plate-forme décrite par la figure 2.1(a), il existe ainsi 8 couplages possibles, illustrés sur la figure 2.3.

Définition 2.2 (Schéma de communication)**.** *On appelle schéma de communication un ensemble de communications point à point qui peuvent être effectuées simultanément dans un modèle de communication donné.*

Pour le modèle un-port unidirectionnel, les schémas de communication sont les couplages du graphe de plate-forme. Là encore, il peut exister un grand nombre de schémas de communication ($O(2^n)$ pour une plate-forme à n processeurs) et nous verrons dans le chapitre suivant comment nous restreindre à des ensembles de taille raisonnable. On appelle $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$ l'ensemble des schémas de communication pour un modèle donné.

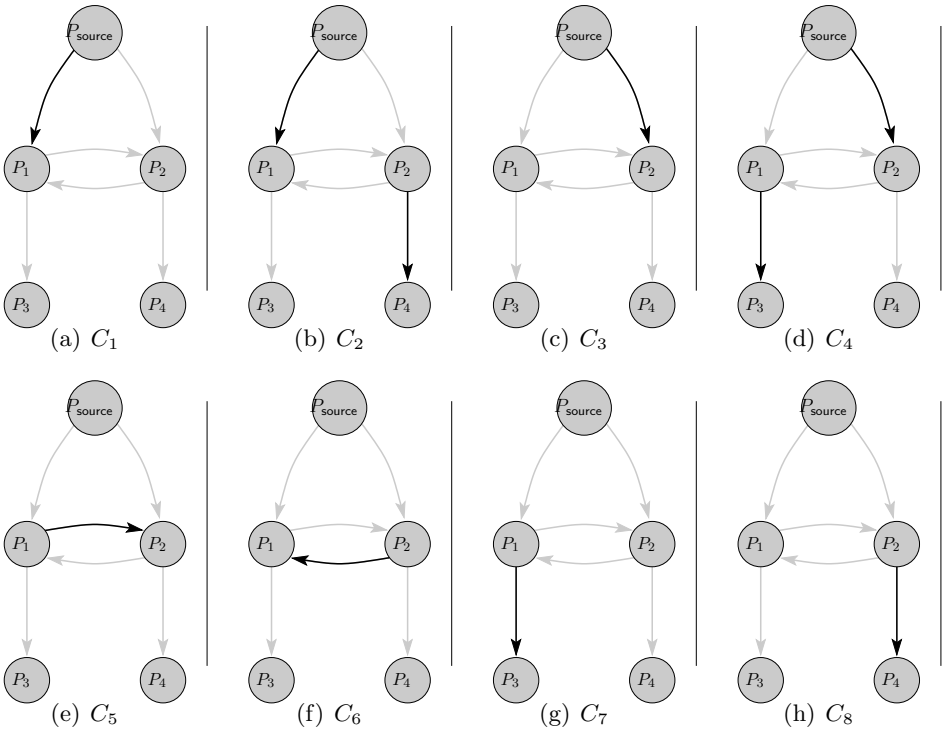
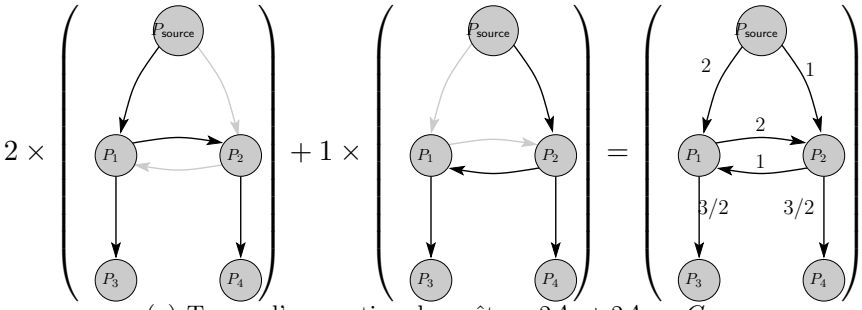


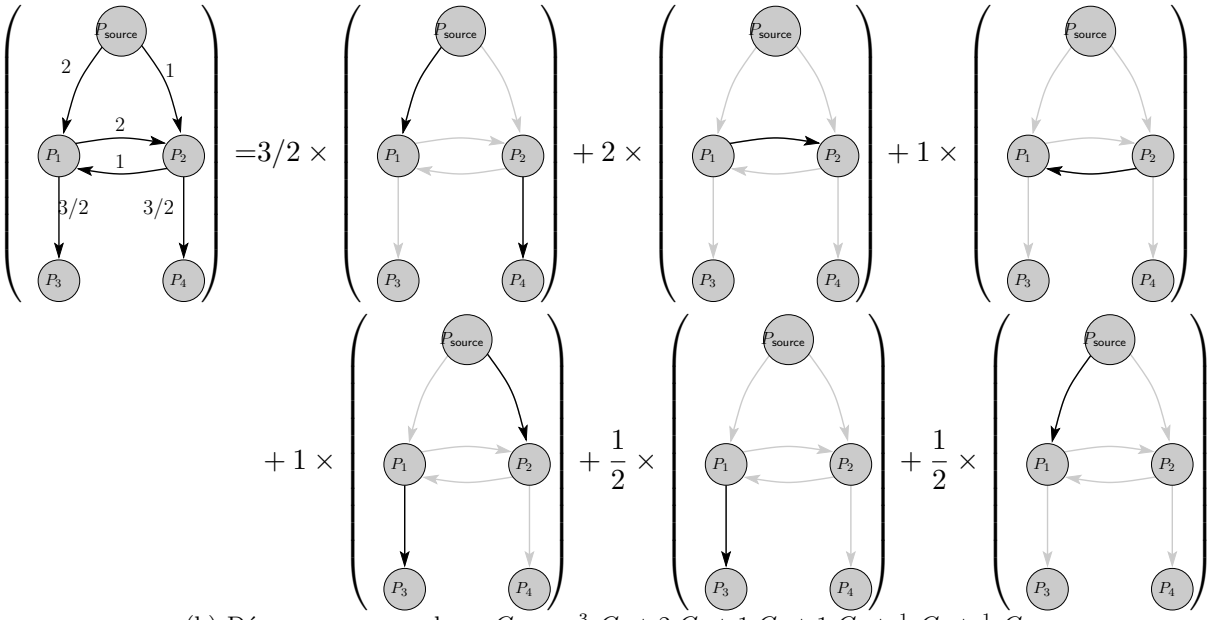
FIG. 2.3 – Couplages possibles pour les communications dans le modèle un-port

Supposons maintenant qu'on veuille utiliser la structure suivante pour une opération de diffusion : dans un motif l'arbre A_1 de la figure 2.1 est utilisé pour diffuser deux messages et l'arbre A_2 pour diffuser un message, en un temps donné T. Par exemple, pour diffuser un grand nombre de messages, les messages m_{3p} et m_{3p+1} utiliseront l'arbre A_1 et les messages m_{3p+2} utiliseront l'arbre A_2. L'ensemble des communications à faire pour un motif est donné par le

graphe pondéré de la figure 2.4(a) : les pondérations représentent les temps d'occupation des arêtes. On suppose ici qu'un message a une taille unité, ainsi le transfert d'un message sur une arête (P_i, P_j) nécessite $c_{i,j}$ unité de temps. Par exemple, dans la figure, l'arête $P_{\text{source}} \to P_1$ est utilisée pendant 2 unités de temps (pour transmettre les deux messages de l'arbre A_1), alors que l'arête $P_1 \to P_3$ est utilisée pendant 3/2 unités de temps pour transmettre 3 messages (deux pour l'arbre A_1 et un pour l'arbre A_2). La figure 2.4(b) montre une façon d'organiser ces communications en utilisant les couplages : le motif est divisé en 6 parties, de durée variable et dans chaque partie un seul couplage est utilisé. Ici le couplage C_2 est utilisé pendant 2 unités de temps, puis le couplage C_5 pendant 2 unités de temps, etc. Au bout de 13/2 unités de temps (la somme des pondérations des couplages utilisés ici), toutes les communications nécessaires ont été réalisées. Ceci signifie en particulier que si le découpage illustré ici est utilisé, le temps nécessaire pour un motif est au moins 13/2.



(a) Temps d'occupation des arêtes : $2A_1 + 2A_2 = G_{\text{occ}}$



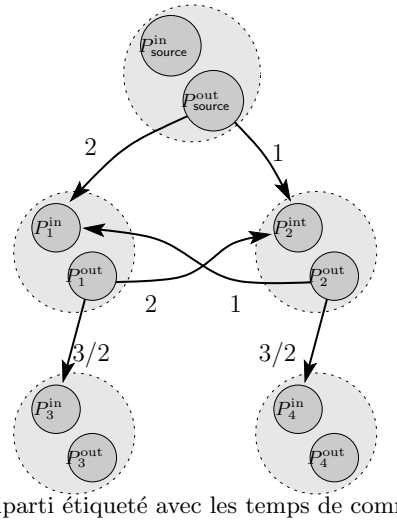
(b) Découpage en couplages $G_{\text{occ}} = \frac{3}{2}\, C_2 + 2\, C_5 + 1\, C_6 + 1\, C_4 + \frac{1}{2}\, C_7 + \frac{1}{2}\, C_1$

FIG. 2.4 – Organisation des communications pour le modèle un-port unidirectionnel

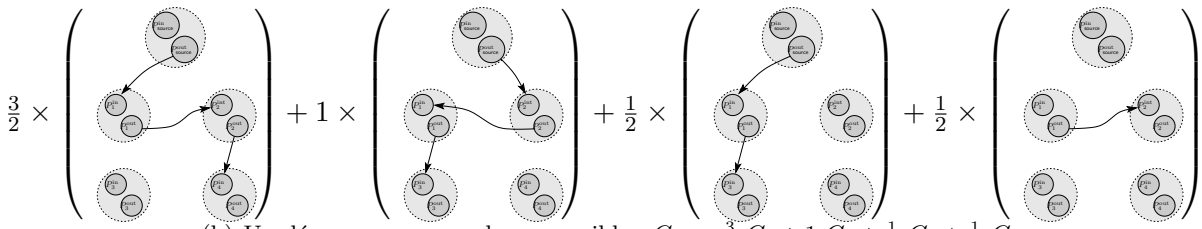
L'organisation des communications dans un motif peut ainsi être décrit par un ensemble de schémas de communication, pondérés par leur temps d'utilisation.

2.2.2 Contraintes pour les communications sous le modèle un-port bidirectionnel

Dans le modèle un-port bidirectionnel, un processeur peut simultanément être impliqué dans une réception et une émission. Pour pouvoir travailler de nouveau avec des couplages, nous construisons le graphe de plate-forme étendu $G_B = (V_{\text{out}} \cup V_{\text{in}}, E_B)$ comme suit : chaque nœud P_i de G est transformé en un nœud de V_{out} et un nœud de V_{in}, l'un $P_i^{\text{out}} \in V_{\text{out}}$ représentant l'activité d'émission et l'autre $P_i^{\text{in}} \in V_{\text{in}}$ représentant l'activité de réception. Pour toute arête (P_i, P_j) dans G, on construit une arête $(P_i^{\text{out}}, P_j^{\text{in}})$ dans E_B. Comme pour le cas unidirectionnel, les arêtes sont ensuite pondérées par leur temps d'occupation pour un motif donné. La figure 2.5(a) représente le graphe G_B obtenu pour notre plate-forme, pondéré par les temps de communication nécessaires à la structure décrite à la partie précédente (figure 2.4(a)), toujours en supposant que les messages sont de taille unité. Les schémas de communication pour le modèle un-port bidirectionnel sont les couplages dans le graphe G_B. Il existe 23 couplages possibles sur notre plate-forme et la figure 2.5(b) propose un découpage de G_B en un ensemble de couplages telle que la somme des poids vaut 7/2.



(a) Graphe biparti étiqueté avec les temps de communication



(b) Un découpage en couplages possible : $G_B = \frac{3}{2}\, C_1 + 1\, C_2 + \frac{1}{2}\, C_3 + \frac{1}{2}\, C_4$

FIG. 2.5 – Organisation des communications pour le modèle un-port bidirectionnel

Les différents schémas de communication vus dans cette partie sont résumés dans la table 2.2.

MODÈLE DE COMMUNICATION	SCHÉMAS DE COMMUNICATION CORRESPONDANTS
un-port unidirectionnel	couplages dans le graphes de plate-forme G
un-port bidirectionnel	couplages dans le graphe biparti G_B

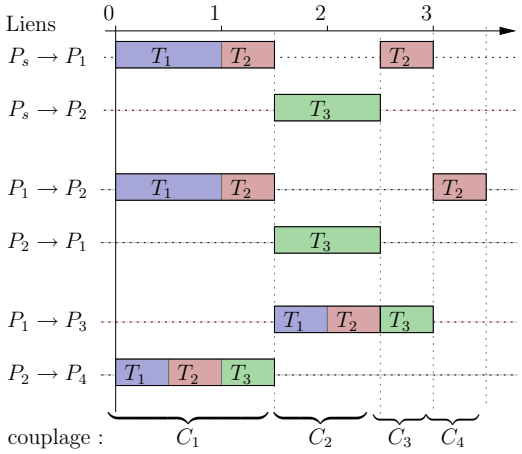
TAB. 2.2 – Schémas de communication pour les différents modèles

2.3 Construction d'un ordonnancement

2.3.1 Exemple de la diffusion sur une petite plate-forme

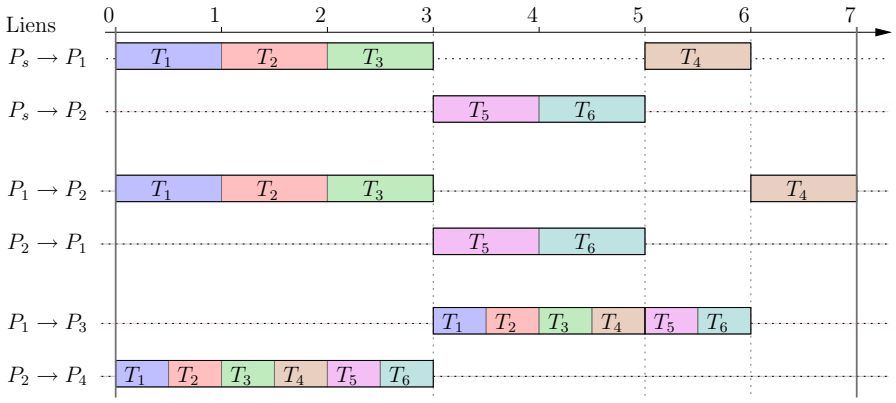
Une fois obtenue une collection pondérée de schémas d'allocation et une collection pondérée de schémas de communication telles que toutes les communications nécessaires aux schémas d'allocation pour un motif peuvent être effectuées selon les schémas de communication, nous pouvons reconstruire un motif dont la durée est la somme des pondérations des schémas de communication.

Nous reprenons notre exemple précédent sous le modèle un port bidirectionnel : sur le graphe de plate-forme de la figure 2.1(a), l'arbre A_1 est utilisé pour diffuser 2 messages et l'arbre A_2 pour diffuser un message. La figure 2.5(b) montre un ensemble pondéré de couplages pour le modèle bidirectionnel. On peut ainsi construire le motif suivant, de durée 7/2. Pour ce faire, on affecte les transferts de messages nécessaires pour les arbres de diffusion, aux emplacements de communication disponible dans les couplages. On n'utilise pas de politique d'allocation spéciale : nous affectons le premier transfert au premier couplage qui convient, en découpant éventuellement un transfert sur plusieurs couplages si le premier couplage disponible n'est pas de longueur suffisante. Dans la figure suivante, T_1 et T_2 représentent les deux instances de l'arbre de diffusion A_1 et T_3 représente l'instance de A_2.



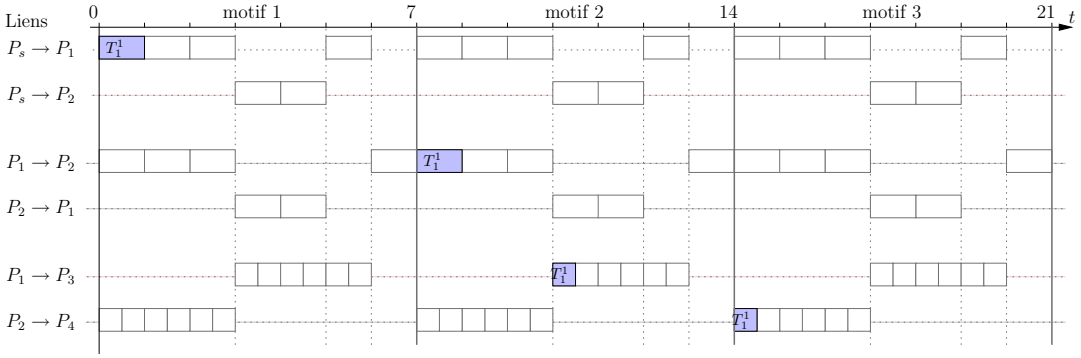
On voit sur cette figure que les transferts de certains messages sont découpés sur plusieurs schémas de communication : par exemple, le transfert $P_{\mathsf{source}} \to P_1$ pour l'arbre T_2 est séparé entre le premier couplage et le troisième. Or notre modèle de communication ne permet pas a priori que que l'on puisse découper un transfert en autant de parties que nécessaire. Pour éviter ce découpage, on calcule le plus petit commun multiple de tous les dénominateurs apparaissant dans les quantités de messages à envoyer par couplage (ici 2) et on multiplie la longueur du

motif par cette quantité. Sur notre exemple, on obtient le motif étendu de longueur 7 suivant :

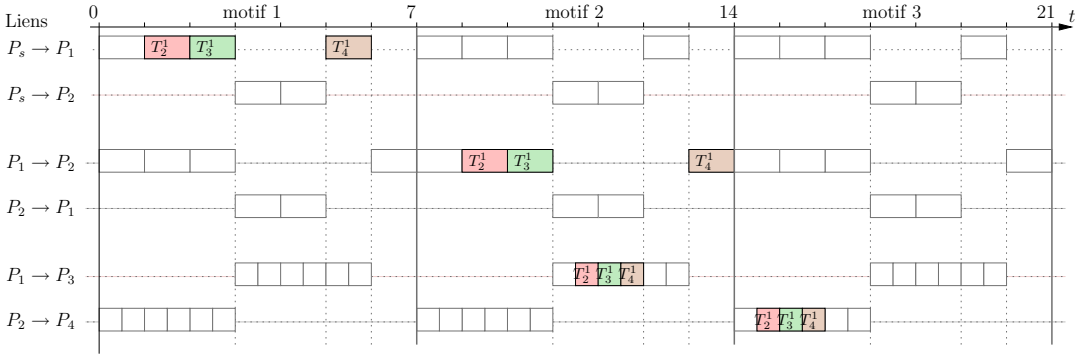


À partir de ce motif, on souhaite reconstruire un ordonnancement respectant les contraintes de précédence. Nous considérons séparément le cas de chaque instance de chaque arbre de diffusion. Pour chacune des instances, on juxtapose plusieurs motifs (au plus autant que de transferts pour cet arbre) et on alloue les communications selon les contraintes de précédence : dans le premier motif, seuls les transferts partant de la source sont alloués, puis dans le second motif, les transferts depuis les processeurs que l'on vient d'atteindre peuvent être alloués, puisque ces processeurs ont déjà reçu le message correspondant à cet arbre et ainsi de suite.

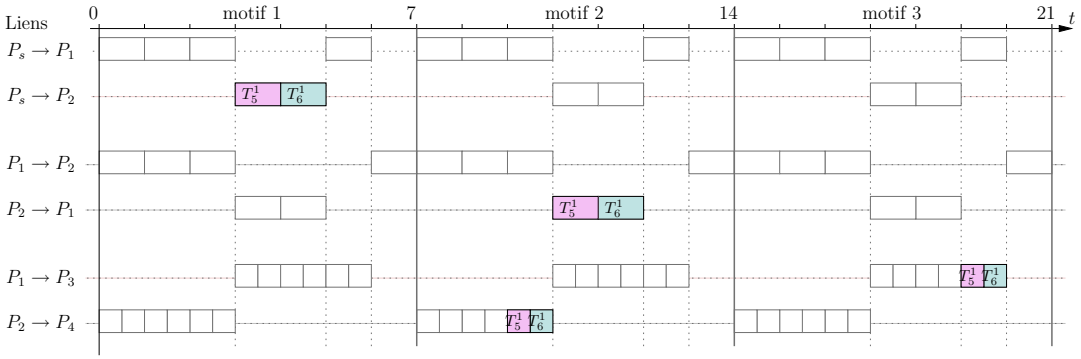
Pour l'arbre T_1 dans notre exemple, en juxtaposant 3 motifs, on alloue $P_{\text{source}} \to P_1$ dans le premier motif, puis comme P_1 a reçu le message à l'issue du premier motif, on alloue $P_1 \to P_2$ et $P_1 \to P_3$ pour l'arbre T_1 dans le second motif et enfin $P_2 \to P_4$ dans le troisième motif. On obtient le schéma suivant pour la diffusion d'un message sur T_1 :



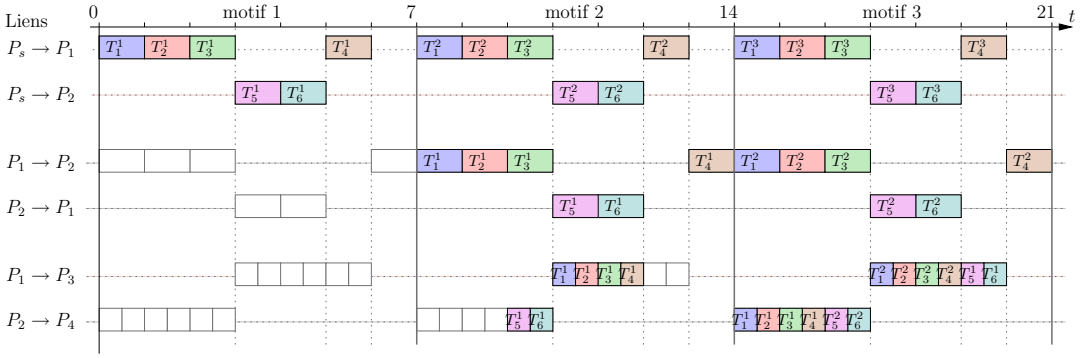
On note T_1^k le parcours du $k^{\text{ème}}$ message diffusé via T_1. Les mêmes allocations peuvent être effectuées pour les arbres T_2, T_3 et T_4 :



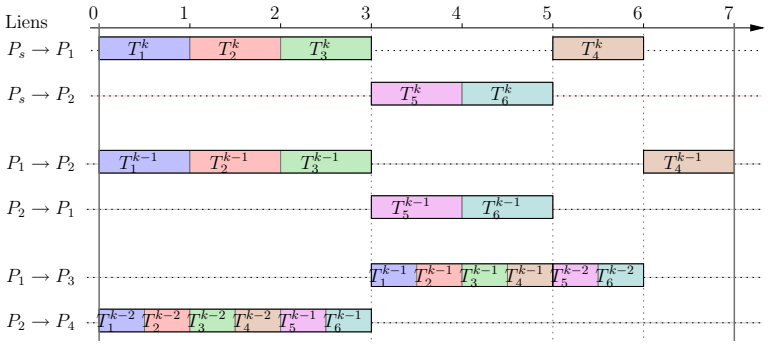
On procède de même pour les arbres T_5 et T_6 : le transfert $P_{\text{source}} \rightarrow P_2$ est alloué dans le premier motif, $P_2 \rightarrow P_1$ et $P_2 \rightarrow P_4$ au second et $P_1 \rightarrow P_3$ au troisième :



Il ne reste plus qu'à répéter les schémas obtenus pour tous les messages empruntant chaque arbre. On obtient l'ordonnancement suivant pour les trois premiers motifs.



On obtient le finalement le motif complet suivant, annoté par les instances des arbres de diffusion, pour le régime permanent :



2.3.2 Généralisation

La construction précédente peut être généralisée : à partir d'un motif M de longueur T_p, on peut construire un motif étendu de longueur $A \times T_p$, où $A = \text{ppcm}_i\{b_i\}$, en notant $\frac{a_i}{b_i}$ les quantités de messages apparaissant dans le motif M ; puis on peut construire un ordonnancement périodique comme précédemment pour obtenir un motif complet.

La formalisation de ce résultat s'appuie sur les notions d'ordonnancement cyclique et d'ordonnancement périodique introduites par Alix Munier dans [81], que nous adaptons ici au cas des communications collectives. Ces notions ont d'abord été introduites pour l'ordonnancement des graphes de tâches, cependant notre contexte de travail est un peu différent :

- nous ne considérons que rarement les tâches de calcul et nous concentrons sur les communications ;
- la notion d'allocation est différente, puisque nous nous autorisons parfois à dupliquer les messages (en particulier pour la diffusion).

Comme l'extension des notions d'ordonnancement cyclique et d'ordonnancement périodique a déjà été écrite pour les séries de graphes de tâches dans [15], nous choisissons de ne présenter que l'extension de ces notions aux communications collectives, et plus précisément pour l'opération de diffusion : en effet, il serait délicat de présenter des notions s'appliquant à toute primitive de communications collectives et la diffusion est une primitive simple qui rassemble les caractéristiques nouvelles de ces opérations : pas de calcul et duplication des messages.

Rappelons que la plate-forme est modélisée par un graphe $G = (V, E, c)$, où $c(e)$, noté $c_{i,j}$ pour $e = (i, j)$, est le temps nécessaire à la transmission d'un message entre P_i et P_j. Nous considérons ainsi que tous les messages ont une taille unité.

L'équivalent d'une allocation d'un graphe de tâches pour la diffusion de messages est le schéma d'allocation, c'est-à-dire l'arbre de diffusion. Nous l'avons déjà défini dans la partie 2.1.1, il s'agit d'un arbre dirigé enraciné en P_{source}, sous-graphe du graphe de plate-forme G et couvrant tous les processeurs.

Nous pouvons alors définir formellement un ordonnancement pour une opération de diffusion comme suit :

Définition 2.3 (Ordonnancement pour une diffusion)**.** *Un ordonnancement valide pour une opération de diffusion, associé à un arbre de diffusion $A = (V, E^A)$ est une application $t : E^A \to \mathbb{R}$ qui vérifie les contraintes suivantes :*

précédence *Pour toutes arêtes (k, l) et (l, m) de E^A,*

$$t(k, l) + c_{k,l} \leqslant t(l, m)$$

un-port *Les contraintes dépendent du modèle choisi :*

- *Sous le modèle un-port bidirectionnel : remarquons qu'il ne peut y avoir de congestion qu'en sortie d'un processeur, puisqu'un processeur ne reçoit qu'une seule fois le message. Pour tout couple d'arêtes (k, l) et (k, m) de E^A,*

$$t(k, l) + c_{k,l} \leqslant t(k, m) \text{ ou bien } t(k, m) + c_{k,m} \leqslant t(k, l)$$

- *Sous le modèle un-port unidirectionnel : pour tout couple d'arêtes (i, j), (k, l) de E^A,*

$$\text{si } i = k \text{ ou } i = l \text{ ou } j = k \text{ ou } j = l \text{ alors } \begin{cases} t(i, j) + c_{i,j} \leqslant t(k, l) \text{ ou bien} \\ t(k, l) + c_{k,l} \leqslant t(i, j) \end{cases}$$

Définition 2.4 (Durée d'un ordonnancement)**.** *La durée d'un ordonnancement valide t est définie par*

$$\max_{(k,l)\in A}(t(k,l)+c_{k,l}) - \min_{(k,l)\in E^A} t(k,l)$$

Nous adaptons maintenant la notion d'ordonnancement cyclique au cas de la diffusion.

Définition 2.5 (Ordonnancement d'une série de diffusions)**.** *Un ordonnancement valide pour une série de diffusions consiste en*

- *une suite d'arbres de diffusion $(A_n) = (V_n = V, (E_n^A))$,*

on note $E_{\mathbb{N}}^A = \{(n, E_n^A),\ \text{pour } n \in \mathbb{N}\}$

- *une application $t : E_{\mathbb{N}}^A \to \mathbb{R}$, où $t(n,(k,l)$ représente la date de départ du transfert (k,l) de l'arbre A_n, qui vérifie les contraintes suivantes :*

  précédence *Pour tout couple d'éléments $(n,(k,l))$ et $(n,(l,m))$ de $E_{\mathbb{N}}^A$,*

  $$t(n,(k,l)) + c_{k,l} \leqslant t(n,(l,m))$$

  un-port *Les contraintes dépendent du modèle choisi :*

  - *Sous le modèle un-port bidirectionnel : dans ce cas, il faut également considérer la congestion en entrée d'un processeur : pour tout couple d'éléments $(n,(k,m))$ et $(n',(l,m))$ de $E_{\mathbb{N}}^A$,*

    $$t(n,(k,m)) + c_{k,m} \leqslant t(n',(l,m)) \text{ ou bien } t(n',(l,m)) + c_{l,m} \leqslant t(n,(k,m))$$

    puis en sortie : pour tout couple d'éléments $(n,(k,l))$ et $(n',(k,m))$ de $E_{\mathbb{N}}^A$,

    $$t(n,(k,l)) + c_{k,l} \leqslant t(n',(k,m)) \text{ ou bien } t(n',(k,m)) + c_{k,m} \leqslant t(n,(k,l))$$

  - *Sous le modèle un-port unidirectionnel : pour tout couple d'éléments $(n,(i,j))$ et $(n',(k,l))$ de $E_{\mathbb{N}}^A$,*

    $$\text{si } i=k \text{ ou } i=l \text{ ou } j=k \text{ ou } j=l \text{ alors } \begin{cases} t(n,(i,j)) + c_{i,j} \leqslant t(n',(k,l)) \text{ ou bien} \\ t(n',(k,l)) + c_{k,l} \leqslant t(n,(i,j)) \end{cases}$$

Définition 2.6 (Durée d'un ordonnancement d'une série de diffusions)**.** *La durée des N premières diffusions d'un ordonnancement d'une série de diffusions est définie par*

$$D_N = \max_{\substack{n=0\ldots N-1 \\ (n,(k,l))\in E_{\mathbb{N}}^A}} t(n,(k,l)) + c(k,l) \quad - \min_{\substack{n=0\ldots N-1 \\ (n,(k,l))\in E_{\mathbb{N}}^A}} t(n,(k,l))$$

Définition 2.7 (Suite $\boldsymbol{K}$-périodique)**.** *Une suite (u_n) est dite K-périodique si elle est croissante et s'il existe un entier n_0 et un réel T_p strictement positif tels que*

$$\forall n \geqslant n_0 : u_{n+K} = u_n + T_p$$

K est appelé facteur de périodicité et T_p représente la période de la suite. $\frac{K}{T_p}$ est alors la fréquence de la suite. Une suite telle que $n_0 = 0$ est dite strictement *K-périodique.*

Définition 2.8 (Ordonnancement K-périodique)**.** *Un ordonnancement $((A_n), t)$ d'une série de diffusions est dit K-périodique de période T_p si*

- *la suite (A_n) vérifie $\forall n \neq 0, A_{n+K} = A_n$,*
- *pour tout arête $(k,l) \in A_n$ la suite $(t(n,(k,l)))_{n\in\mathbb{N}}$ est strictement K-périodique de période T_p, c'est-à-dire*

$$\forall n \in \mathbb{N} t(n+K,(k,l)) = t(n,(k,l)) + T_p$$

Définition 2.9 (Débit d'un ordonnancement d'une série de diffusions)**.** *Le débit (ou fréquence) d'un ordonnancement est, sous réserve de son existence, la limite*

$$\lim_{N\to\infty} \frac{N}{D_N}$$

Remarque Un ordonnancement K-périodique de période T_p est entièrement caractérisé par les K premières valeurs de la suite (A_n) et des $t(n,(k,l))$. La définition suivante permet ainsi de définir un ordonnancement K-périodique uniquement avec ses K premières valeurs. Notons également que le débit d'un ordonnancement K-périodique de période T_p est égal à $\frac{K}{T_p}$.

Définition 2.10 (K-motif de longueur T_p)**.** *Un K-motif de longueur T_p est une suite $(A_n)_{n=1\ldots K}$ et une application $t : E^A_{[\![1,K]\!]} \to \mathbb{R}$ (en notant $E^A_{[\![1,K]\!]} = \{(n, E^A_n),$ pour $n \in [\![1,K]\!]\}$) telles que, si on construit*

- *la suite (A'_n) telle que $A'_n = A_{n \mod K}$,*
- *et l'application $t' : E^{A'}_{\mathbb{N}} \to \mathbb{R}$ telle que $t'(n,(k,l)) = t(n \mod K,(k,l) + T_p \times (n \operatorname{div} K)$,*

alors $((A'_n), t')$ est un ordonnancement K-périodique de période T_p.

Nous sommes désormais en mesure d'énoncer le résultat qui généralise la construction illustrée sur l'exemple précédent.

Théorème 2.1. *Soit une plate-forme représentée par un graphe pondéré $G = (V, E, c)$ et un processeur P_{source}. Étant donné une collection pondérée d'arbres de diffusion $((A_1, x_1), \ldots, (A_n, x_n)$ et une collection pondérée de schémas de communication $((C_1, y_1), \ldots, (C_p, y_p)$, tels que les schémas de communication permettent d'effectuer toutes les communications nécessaires aux arbres de diffusion, c'est-à-dire tels que*

$$\forall (k,l) \in E, \quad \sum_{\substack{C_i \\ (k,l)\in C_i}} y_i \geqslant \sum_{\substack{A_i \\ (k,l)\in A_i}} x_i \cdot c_{k,l} \tag{2.1}$$

et tels que, avec les notations $c_e = \alpha_e/\beta_e$, $x_i = \gamma_i/\delta_i$ et $y_i = \mu_i/\nu_i$, on a $\log\alpha_e + \log\beta_e \leqslant B$, $\log\gamma_i + \log\delta_i \leqslant B$ et $\log\mu_i + \nu_i \leqslant B$, alors il est possible de construire un ordonnancement K-périodique de période T_p, de débit $(\sum_{i=1}^n x_i)/(\sum_{i=1}^p y_i)$, tel que $\log K \leqslant (|E| + 2n + p)B$.

Démonstration. Nous n'allons pas exhiber ici la preuve formelle de ce résultat, qui repose entièrement sur la méthode décrite dans l'exemple précédent. On reprend brièvement cette construction, en notant $x = \sum_{i=1}^n x_i$ et $y = \sum_{i=1}^p y_i$.

- Nous commençons par construire un motif de durée y, dans lequel toutes les communications des x arbres de diffusion trouvent leur place, grâce à la contrainte 2.1.

– Les transferts peuvent être découpés en plusieurs parties dans le motif précédent. En reprenant les notations du théorème, dans le pire cas, la quantité de messages pour l'arbre i envoyée pendant le schéma de communication j sur une arête e est

$$\frac{1}{\beta_e} \times \frac{1}{\delta_i} \times \frac{1}{\nu_j}$$

Nous calculons ensuite le plus petit commun multiple L des dénominateurs de ces quantités. On peut donc borner L par :

$$L \leqslant \prod_{e=1}^{|E|} \beta_e \times \prod_{i=1}^{n} \delta_i \times \prod_{j=1}^{p} \nu_j$$

En multipliant toutes les quantités par L, nous obtenons un motif de taille $K = L \times x$, de période $T_p = L \times y$. Nous allouons ensuite précisément les instances des transferts des différents arbres, en respectant les contraintes de dépendances.

Nous obtenons alors un K-motif de période T_p, décrivant un ordonnancement K-périodique de débit $K/T_p = x/y$, avec

$$K \leqslant \prod_{e=1}^{|E|} \beta_e \times \prod_{i=1}^{n} \delta_i \times \prod_{j=1}^{p} \nu_j \times x$$

$$\log K \leqslant |E|\, B + nB + pB + \log(nB) \leqslant |E|\, B + 2nB + pB$$

Une preuve formelle de ce résultat existe dans [15], qui s'appuie sur un résultat comparable pour les graphes de tâches ; on utilise ensuite une construction ad-hoc pour produire un graphe de tâches correspondant à l'ensemble pondéré d'arbres $((A_1, x_1), \ldots, (A_n, x_n)$. Cependant, comme cette démonstration est assez lourde du fait du formalisme nécessaire et qu'elle est à la fois trop générale et peu adaptée à notre cas, nous ne la reproduisons pas ici. ■

Extension aux autres communications collectives Nous avons défini formellement les notions d'ordonnancement, de débit, et motif K-périodique pour une série diffusion. Ces notions et le résultat du théorème 2.1 s'étendent naturellement aux autres communications collectives étudiées, comme la distribution de données ou la réduction.

Chapitre 3

Méthodologie pour la résolution

Forts des notions de schémas d'allocation et de schémas de communication que nous avons définies dans le chapitre précédent, nous nous intéressons désormais à la recherche d'ordonnancements pour les problèmes de communications collectives pipelinées, en cherchant à maximiser le débit obtenu.

3.1 Résolution à base de programmation linéaire

Dans cette partie, nous nous appuyons sur la caractérisation d'un ordonnancement sous la forme de schémas d'allocation et de schémas de communication et nous exprimons les contraintes que les pondérations de ces schémas doivent respecter. En rassemblant ces contraintes et en ajoutant l'objectif de maximiser le débit, nous obtenons un programme linéaire, dont la solution optimale nous donne une borne supérieure sur le débit atteignable. Même si une solution peut a priori posséder un très grand nombres de schémas d'allocation et de couplages, nous montrons que nous pouvons nous restreindre à des solutions de taille polynomiale en la taille de la plate-forme. Nous montrons ensuite qu'à partir d'une solution optimale du programme linéaire, nous sommes en mesure de construire un ordonnancement périodique qui réalise le débit optimal.

Restrictions Pour plus de simplicité, nous nous restreignons au cas où tous les messages ont une taille unité, ainsi $N \times c_{i,j}$ représente le temps nécessaire à l'envoi de N messages sur l'arête (i, j). Si on voulait prendre en compte des messages de tailles différentes, par exemple si on souhaitait distribuer des messages de taille variable, il faudrait alors rajouter une pondération aux schémas d'allocation dans ce qui suit. Comme ceci n'a pas de sens pour toutes les primitives de communications collectives, nous préférerons ne pas exposer notre méthode avec cette généralisation, qui induit une complexification inutile des notations. Pour les mêmes raisons, nous nous restreignons aux primitives de communications qui ne nécessitent pas de calcul. Pour pouvoir prendre en compte la réduction de données, il faut donc rajouter les contraintes de calculs à celles présentées ici.

Comme les schémas d'allocation étudiés dans ce chapitre ne sont constitués que de communications, nous confondons par la suite un schéma d'allocation et la liste des arêtes qu'il utilise, et nous noterons $e \in A$ quand le schéma A utilise l'arête e, afin de ne pas alourdir les notations. Par exemple, dans le cas de la diffusion nous considérerons qu'un schéma d'allocation est la liste des arêtes d'un arbre de diffusion. Cependant, comme une arête peut être présente plusieurs

fois dans un schéma (utilisée simultanément par deux routes distinctes pour une distribution de données par exemple), un schéma d'allocation peut contenir plusieurs fois le même élément ; c'est pour cela que l'on considère la notion de liste d'arêtes plutôt que celle d'ensemble d'arêtes). Par exemple dans la somme $\sum_{e\in A} f(e)$, si une arête e est présente plusieurs fois dans le schéma A, elle sera comptabilisée autant de fois dans la somme.

3.1.1 Construction du programme linéaire

Nous considérons ici un ordonnancement quelconque pour le problème de la diffusion sur une plate-forme donnée par un graphe pondéré $G = (V, E, c)$, depuis le processeur P_{source}. On note $T(N)$ le temps nécessaire à cet ordonnancement pour diffuser N messages. Comme vu à la partie précédente, on considère la décomposition de l'ordonnancement en schémas d'allocation et on note $n_a(N)$ le nombre de messages diffusés en suivant l'arbre de diffusion A_a. On considère également la décomposition en schémas de communication de cet ordonnancement et on appelle $T_c(N)$ le temps total pendant lequel le schéma de communication C_c est utilisé pour cet ordonnancement.

Nous pouvons écrire un certain nombre de contraintes sur ces variables n_a, T_c.

– Tout d'abord, ces variables doivent être positives :

$$\forall A_a \in \mathcal{A},\ n_a(N) \geqslant 0, \quad \forall C_c \in \mathcal{C},\ T_c(N) \geqslant 0.$$

– D'autre part, la somme des durées d'utilisation de chaque schéma de communication ne peut être plus grande que la durée totale de l'ordonnancement :

$$\sum_{C_c\in\mathcal{C}} T_c(N) \leqslant T(N).$$

– Enfin, il faut que toutes les communications nécessaire aux schémas d'allocation puissent être effectuées par les schémas de communication. Pour l'arête (i, j), le temps total d'utilisation de cette arête par les schémas de communication doit être au moins égal au nombre de messages communiqués sur cette arête par les schéma d'allocation, multiplié par le coût de transmission d'un message par l'arête :

$$\forall (i,j) \in E, \quad \sum_{\substack{C_c\in\mathcal{C}\\(i,j)\in C_c}} T_c(N) \geqslant c_{i,j} \cdot \sum_{\substack{A_a\in\mathcal{A}\\(i,j)\in A_a}} n_a(N).$$

Notons qu'il pourrait sembler plus naturel d'avoir une égalité entre ces deux quantités : le temps nécessaire au transfert de tous les messages pour les schémas d'allocation est égal au temps d'occupation de ces arêtes dans les couplages. Cependant, cela rajoute une contrainte qui n'est pas indispensable : il suffit que le temps d'occupation des arêtes dans les couplages soit suffisamment long pour transmettre tous les messages. De plus, la formulation du programme linéaire sous forme d'inégalités rend plus lisible la transformation en programme linéaire dual de la partie 3.1.3.

Nous rassemblons ces contraintes en un programme linéaire, en changeant légèrement les variables : nous utilisons x_a le nombre moyen de messages utilisant le schéma d'allocation A_a par unité de temps et y_c le temps moyen d'utilisation du schéma de communication C_c par unité de temps. On rappelle que $\mathcal{A}$ représente l'ensemble des schémas d'allocation valides pour

l'opération considérée et $\mathcal{C}$ l'ensemble des schémas de communication valides sous le modèle de communication choisi.

$$\begin{array}{l}\text{MAXIMISER } \rho_{\text{opt}} = \sum_{A_a \in \mathcal{A}} x_a, \\ \text{SOUS LES CONTRAINTES} \\ \left\{ \begin{array}{ll} \text{(3.1a)} & \displaystyle\sum_{C_c \in \mathcal{C}} y_c \leqslant 1 \\ \text{(3.1b)} & \forall (i,j) \in E, \displaystyle\sum_{\substack{C_c \in \mathcal{C} \\ (i,j) \in C_c}} y_c \geqslant \sum_{\substack{A_a \in \mathcal{A} \\ (i,j) \in A_a}} x_a \cdot c_{i,j} \\ \text{(3.1c)} & \forall A_a \in \mathcal{A},\ x_a \geqslant 0 \\ \text{(3.1d)} & \forall C_c \in \mathcal{C},\ \ y_c \geqslant 0 \end{array} \right. \end{array} \tag{3.1}$$

Les variables $x_a = \frac{n_a(N)}{T(N)}$ et $y_c = \frac{T_c(N)}{T(N)}$ vérifient les contraintes de ce programme linéaire, de sorte que pour tout ordonnancement et pour toute taille de message N, on vérifie que

$$\sum_{A_a \in \mathcal{A}} \frac{n_a(N)}{T(N)} = \frac{N}{T(N)} \leqslant \rho_{\text{opt}}.$$

Le programme linéaire précédent permet donc d'obtenir une borne supérieure sur le débit d'une opération de communication collective donnée.

Exemple de la diffusion en un-port unidirectionnel Nous reprenons l'exemple décrit à la figure 2.1 du chapitre précédent, avec les couplages de communication pour le modèle un-port illustré sur la figure 2.3. Pour ce problème, le programme linéaire s'écrit de la façon suivante :

$$\begin{array}{l}\text{MAXIMISER } \rho_{\text{opt}} = x_1 + x_2 + x_3, \\ \text{SOUS LES CONTRAINTES} \\ \left\{ \begin{array}{ll} \text{(3.2a)} & y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7 + y_8 \leqslant 1 \\ \text{(3.2b)} & \begin{array}{ll} \text{pour l'arête } (P_{\mathsf{source}}, P_1) : & y_1 + y_2 \geqslant x_1 + x_3 \\ \text{pour l'arête } (P_{\mathsf{source}}, P_2) : & y_3 + y_4 \geqslant x_2 + x_3 \\ \text{pour l'arête } (P_1, P_2) : & y_5 \geqslant x_1 \\ \text{pour l'arête } (P_2, P_1) : & y_6 \geqslant x_2 \\ \text{pour l'arête } (P_1, P_3) : & y_4 + y_7 \geqslant x_1 + x_2 + x_3 \\ \text{pour l'arête } (P_2, P_4) : & y_2 + y_8 \geqslant x_1 + x_2 + x_3 \end{array} \\ \text{(3.2c)} & x_1 \geqslant 0,\ x_2 \geqslant 0,\ x_3 \geqslant 0, \\ \text{(3.2d)} & y_1 \geqslant 0,\ y_2 \geqslant 0,\ y_3 \geqslant 0,\ y_4 \geqslant 0,\ y_5 \geqslant 0,\ y_6 \geqslant 0,\ y_7 \geqslant 0,\ y_8 \geqslant 0. \end{array} \right. \end{array} \tag{3.2}$$

La contrainte (3.2a) indique que la somme des temps d'occupation des couplages doit être inférieure à une unité de temps, puisqu'on travaille sur une période de cette longueur. Les contraintes (3.2b) précisent l'utilisation de chaque arête : par exemple l'arête (P_1, P_3) est utilisée par les arbres de diffusion A_1,A_2 et A_3 et elle est présente dans les couplages C_4 et C_7. Elle est donc utilisée pour transmettre $x_1 + x_2 + x_3$ messages en un temps unité, ce qui prend $(x_1 + x_2 + x_3)/2$ unités de temps ($c_{1,3} = 1/2$). Ceci doit être fait en utilisant les couplages

qui comprennent cette arête, dont le temps d'occupation total est $y_4 + y_7$, d'où la contrainte $y_4 + y_7 \geqslant x_1 + x_2 + x_3$. Les dernières contraintes (3.2c et 3.2d) indiquent simplement que les variables sont positives.

3.1.2 Existence d'une solution optimale compacte

Pour simplifier les notations que l'on va introduire, on récrit le programme linéaire 3.1 sous la forme suivante :

$$(\mathcal{P}) \quad \begin{array}{l} \text{Maximiser } {}^t c \cdot X \\ \text{Sous les contraintes} \\ \begin{cases} A \cdot X \leqslant b, \\ X \geqslant 0 \end{cases} \end{array}$$

avec A matrice de taille $n \times m$ dont les coefficients sont dans $\mathbb{Q}$, $b \in \mathbb{Q}^n$ et $c \in \mathbb{Q}^m$. Ce programme linéaire est équivalent au programme linéaire 3.1 si on pose :

- $m = |\mathcal{A}| + |\mathcal{C}|$, c'est le nombre de variables ;
- $n = |E| + 1$, c'est le nombre de contraintes non triviales, c'est-à-dire autres que $x_a \geqslant 0$ ou $y_c \geqslant 0$;
- Pour $a = 1, \ldots, |\mathcal{A}|$, $X_a = x_a$ et pour $c = 1, \ldots, |\mathcal{C}|$ $X_{c+|\mathcal{A}|} = y_c$;
- $b \in \mathbb{Q}^n$ est défini par : $b_1 = 1$ et pour $i = 1, \ldots, |E|$, $b_{i+1} = 0$.
- $c \in \mathbb{Q}^m$ est défini par :

$$\begin{aligned} &\text{pour } 1 \leqslant l \leqslant |\mathcal{A}|\,, \quad c_l = 1, \\ &\text{pour } 1 \leqslant l \leqslant |\mathcal{C}|\,, \quad c_{l+|\mathcal{A}|} = 0. \end{aligned}$$

- A est la matrice des contraintes (3.1a) et (3.1b), elle est donc composée de zéros, de uns et de coefficients $c_{i,j}$. Plus précisément, en notant $E = \{e_1, e_{|E|}\}$ l'ensemble d'arête de G :

$$\begin{aligned} &\text{pour } 1 \leqslant l \leqslant |\mathcal{A}|\,, && A[1, l] = 0 \\ &\text{pour } 1 \leqslant l \leqslant |\mathcal{C}|\,, && A[1, l + |\mathcal{A}|] = 1 \\ &\text{pour } 1 \leqslant l \leqslant |\mathcal{A}|\,, \text{ et } 1 \leqslant k \leqslant |E|\,, && A[k+1, l] = \begin{cases} c_{i,j} \text{ si } e_k \in A_l \\ 0 \text{ sinon} \end{cases} \\ &\text{pour } 1 \leqslant l \leqslant |\mathcal{C}|\,, \text{ et } 1 \leqslant k \leqslant |E|\,, && A[k+1, l+|\mathcal{A}|] = \begin{cases} -1 \text{ si } e_k \in C_l \\ 0 \text{ sinon} \end{cases} \end{aligned}$$

On obtient la matrice représenté ci-dessous :

	allocation A_l pour $l = 1, \ldots, \lvert\mathcal{A}\rvert$	couplage C_l pour $l = 1, \ldots, \lvert\mathcal{C}\rvert$
	0,0,. . . ,0	1,1,. . . ,1
arête $e_k \in E$	($c_{i,j}$ si $e_k \in A_l$, 0 sinon)	(-1 si $e_k \in C_l$, 0 sinon)

En écrivant $c_{i,j} = \frac{\alpha_{i,j}}{\beta_{i,j}}$ sous forme irréductible, on pose $c_{\max} = \max_{i,j}\{\alpha_{i,j}, \beta_{i,j}\}$, ainsi la taille du codage de chaque $c_{i,j}$ est au plus $2\log c_{\max}$. Le codage de chaque coefficient de A, b et c est donc de taille au plus $O(\log c_{\max})$. Comme m est de grande taille, le nombre de contraintes et la taille du résultat X peuvent a priori être exponentiels en la taille de la plate-forme, c'est-à-dire $O(\max(|V|, |E| \times \log c_{\max}))$. Cependant parmi ces contraintes, $|\mathcal{A}| + |\mathcal{C}|$ sont des simples contraintes $X \geqslant 0$. Ainsi si on nous fournit une solution comportant P schémas d'allocation et Q schémas de communication, il est facile de vérifier en temps polynomial en $P+Q$ (et la tailles des coefficients X_i) que les contraintes sont bien vérifiées et donc que la solution est valide.

Plus formellement, on suppose qu'on dispose d'un oracle $\mathcal{O}$ qui pour toute donnée X décrite par ses k coefficients non nuls sous la forme $(i_1, x_{i_1} = a_{i_1}/b_{i_1}), \ldots, (i_l, x_{i_k} = a_{i_k}/b_{i_k})$ (on suppose implicitement que $X_i = 0$ sur les autres composantes) et pour tout $K \in \mathbb{Q}$ permet de déterminer si les contraintes

$$\begin{cases} A \cdot X \leqslant b, \\ X \geqslant 0 \\ {}^t c \cdot X \geqslant K \end{cases}$$

sont vérifiées en temps $O(k^\gamma \log(\max_l(a_{i_l}, b_{i,l}))$ pour un γ fixé. La dernière contrainte permet de vérifier si le débit de la solution décrite par X est supérieure à une borne K, comme celle qui apparaît dans le problème de décision associé au programme linéaire précédent, que nous définissons maintenant :

Définition 3.1 (DEC-PROG-LIN)**.** *existe-t-il un $X \in \mathbb{Q}^m$ qui vérifie les contraintes du programme linéaire $(\mathcal{P})$ et tel que la valeur de l'objectif ${}^t c \cdot X \geqslant K$?*

Étant donné le grand nombre de contraintes à vérifier dans le programme linéaire $(\mathcal{P})$, ce problème n'est a priori pas dans NP. Toutefois le nombre de contraintes non triviales (c'est-à-dire différentes de $X \geqslant 0$) est de l'ordre du nombre d'arêtes dans le graphe de plate-forme. On considère le problème de décision réduit suivant :

Définition 3.2 (DEC-PROG-LIN-RED)**.** *existe-t-il un $X \in \mathbb{Q}^m$ tel que :*
- *X possède au plus n composantes non nulles,*
- *les composantes non nulles de X, $X_i = a_i/b_i$, vérifient*

$$\log(a_i) + \log(b_i) \leqslant 2n(\log n + 2n \log c_{\max}) + n \log c_{\max}$$

- *X satisfait les contraintes du programme linéaire $(\mathcal{P})$ et ${}^t c \cdot X \geqslant K$?*

Nous montrons que ces deux problèmes sont équivalents, mais que le second est dans NP.

Théorème 3.1. *DEC-PROG-LIN-RED appartient à la classe NP et s'il existe une solution X de DEC-PROG-LIN, alors il existe une solution Y de DEC-PROG-LIN-RED et Y est également solution de DEC-PROG-LIN.*

Démonstration. Commençons par l'appartenance de DEC-PROG-LIN-RED à la classe NP. Nous utilisons les composantes non nulles de la donnée X comme certificat, décrites par $(i_1, x_{i_1} = a_{i_1}/b_{i_1}), \ldots, (i_n, x_{i_n} = a_{i_n}/b_{i_n})$.
- On peut facilement vérifier les deux premières contraintes de DEC-PROG-LIN-RED, c'est-à-dire qu'il y a au plus n composantes non nulles et que $\log(a_i) + \log(b_i) \leqslant 2n(\log n + 2n \log c_{\max}) + n \log c_{\max}$.

– On peut alors utiliser l'oracle $\mathcal{O}$ pour vérifier les contraintes linéaires en temps $O\big(n^\gamma \times (2n(\log n + 2n \log c_{\max}) + n \log c_{\max})\big)$

On peut donc vérifier si X est solution de DEC-PROG-LIN-RED en temps polynomial en la taille des données, donc DEC-PROG-LIN-RED est dans NP.

Soit X une solution de DEC-PROG-LIN. Alors X est une solution du programme linéaire 3.1 de débit supérieur à K. On sait [89] qu'il existe une solution optimale X^{opt} du programme linéaire en un sommet du polyèdre défini par les contraintes de $(\mathcal{P})$:

$$\begin{cases} A \cdot X \leqslant b, \\ X \geqslant 0 \end{cases}$$

X^{opt} est donc obtenu en résolvant un système linéaire de taille $m \times m$ de plein rang extrait de

$$\begin{pmatrix} A \\ I_m \end{pmatrix} \cdot X = \begin{pmatrix} b \\ 0_m \end{pmatrix},$$

sachant que $\left(\begin{smallmatrix} A \\ I_m \end{smallmatrix}\right)$ est nécessairement de rang m puisqu'elle contient I_m. Comme A est de taille $n \times m$, le système linéaire $m \times m$ contient au plus n lignes de A, donc au minimum $m - n$ lignes sont du type $X_i = 0$; c'est-à-dire qu'au plus n composantes de X^{opt} sont non nulles.

Considérons A', matrice du système $n' \times n'$ ($n' \leqslant n$) extrait du système linéaire $m \times m$ dans lequel les variables contraintes à zéros (et les contraintes triviales correspondantes) ont été éliminées. On note y le vecteur des composantes non contraintes à 0 de X^{opt}. Ainsi y est solution d'un système $A' \cdot y = b'$, où A' est extraite de A, de taille $n' \times n'$, et b' est un vecteur de taille n' extrait de $\left(\begin{smallmatrix} b \\ 0_m \end{smallmatrix}\right)$. A' est à coefficients rationnels, ses coefficients étant 0,1 ou $c_{i,j}$. En notant de nouveau $c_{i,j} = \frac{\alpha_{i,j}}{\beta_{i,j}}$ sous forme irréductible, on calcule $L = \text{ppcm}_i\{b_{i,j}\}$. Comme $\beta_{i,j} \leqslant c_{\max}$, on a $L \leqslant {c_{\max}}^{|E|} \leqslant {c_{\max}}^{n}$.

On appelle A'' la matrice obtenue à partir de A' en multipliant chaque coefficient par L. Ainsi A'' est à coefficients entiers, ses coefficients sont majorés par $\max_{i,j} \alpha_{i,j} \times L \leqslant L^2$ et $y' = \frac{1}{L}\, y$ est solution du système $A'' \cdot y' = b'$.

Les coefficients de y' peuvent être obtenus en utilisant les formules de Cramer [55] sous la forme ${y'}_k = \frac{\det(N^{(k)})}{\det(D^{(k)})}$, où $N^{(k)}$ et $D^{(k)}$ sont des matrices $n' \times n'$ extraites de (A'', b'). De plus on peut majorer la valeur de ces déterminants comme suit :

$$\begin{aligned}
\det\left(N^{(k)}\right) &= \prod_1^{n'} \lambda_j && \lambda_j \text{ valeur propre de } N^{(k)} \\
&\leqslant \left(\left|\left|N^{(k)}\right|\right|_2\right)^2 \\
&\leqslant \left(n' \max_{i,j} N^{(k)}_{i,j}\right)^{n'} && \text{d'après [55]} \\
&\leqslant (n' L^2)^{n'} \\
&\leqslant (nL^2)^n \\
&\leqslant (nL^2)^n
\end{aligned}$$

La même propriété est évidemment vérifiée par $\det\left(D^{(k)}\right)$. On vérifie donc que y'_k s'écrit sous forme irréductible :

$$y'_k = \frac{a'_k}{b'_k}, \text{ où } \begin{cases} \log(|a'_k|) \leqslant n(\log n + 2\log L) \leqslant n(\log n + 2n\log c_{\max}) \\ \log(|b'_k|) \leqslant n(\log n + 2\log L) \leqslant n(\log n + 2n\log c_{\max}) \end{cases}$$

Puis, on peut écrire y_k sous forme irréductible :

$$y_k = \frac{a'_k}{L \times b'_k} = \frac{a_k}{b_k}, \text{ où } \begin{cases} \log(|a_k|) \leqslant n(\log n + 2n\log c_{\max}) \\ \log(|b_k|) \leqslant (\log n + 2n\log c_{\max}) + \log L \leqslant n(\log n + 2n\log c_{\max}) + n\log c_{\max} \end{cases}$$

Donc

$$\log(|a_k|) + \log(|b_k|) \leqslant 2n(\log n + 2n\log c_{\max}) + n\log c_{\max}$$

Considérons maintenant X^{opt}, vecteur de taille m construit en complétant y par des 0. On vérifie alors que

- X^{opt} possède au plus n composantes non nulles
- Si ${X^{\text{opt}}}_k = \frac{a_k}{b_k} \neq 0$, alors $\log(|a_k|) + \log(|b_k|) \leqslant 2n(\log n + 2n\log c_{\max}) + n\log c_{\max}$.
- $A \cdot X^{\text{opt}} \leqslant b$, $X^{\text{opt}} \geqslant 0$ et ${}^t c \cdot X^{\text{opt}} \geqslant K$,

de sorte que X^{opt} est bien une solution de DEC-PROG-LIN-RED (et par construction également une solution de DEC-PROG-LIN). ■

Remarque Le résultat précédent est valable pour tous les problèmes étudiés, indépendamment de leur complexité, pourvu qu'il existe un oracle qui vérifie les contraintes non triviales du programme linéaire en temps polynomial. Comme cet ensemble de contraintes est de petite taille ($|E|+1$ pour le modèle un-port), ceci est toujours possible. Cela justifie notre formulation du problème sous la forme d'un programme linéaire : malgré le nombre important de contraintes et de variables, on peut se restreindre à étudier des «petites» solutions, qu'on peut vérifier en temps polynomial. En particulier, même pour les problèmes de communications collectives qui n'admettent pas d'algorithmes polynomiaux, on sait qu'il existe des solutions optimales qui sont décrites avec un «petit» nombre de schémas d'allocation et de schémas de communication.

3.1.3 Résolution à l'aide de la méthode de l'ellipsoïde

Nous allons maintenant nous intéresser à la résolution du problème d'optimisation du débit. Notre résolution est fondée sur la méthode des ellipsoïdes pour la programmation linéaire, introduite par Khachiyan [66] et présentée en détail dans le livre de Schrijver [88]. Pour pouvoir appliquer cette méthode, nous construisons le programme linéaire dual de $(\mathcal{P})$, qui s'écrit :

$$(\mathcal{D}) \quad \begin{array}{l} \text{Minimiser } {}^t b \cdot U \\ \text{Sous les contraintes} \\ \begin{cases} {}^t A \cdot U \geqslant c, \\ U \geqslant 0 \end{cases} \end{array}$$

On détaille les contraintes du programme linéaire dual. La $l^{\text{ème}}$ contrainte s'écrit :

$$\begin{aligned} & [{}^tAU]_l \geqslant c_l \\ \Leftrightarrow & \sum_{k=1}^{|E|+1} A_{k,l}U_k \geqslant c_l \\ \Leftrightarrow & A[1,l]U_1 + \sum_{k=1}^{|E|} A_{k+1,l}U_k \geqslant c_l \end{aligned}$$

Les $|\mathcal{A}|$ premières contraintes peuvent être associées aux schéma d'allocation : au schéma d'allocation A_l correspond la contrainte l (pour $1 \leqslant l \leqslant |\mathcal{A}|$) :

$$\sum_{\substack{e_k=(i,j) \\ e_k \in A_l}} c_{i,j}U_{k+1} \geqslant 1$$

Les $|\mathcal{C}|$ contraintes suivantes sont associées aux schémas de communication : au schéma de communication C_l correspond la contrainte $l + |\mathcal{A}|$ (pour $1 \leqslant l \leqslant |\mathcal{C}|$) :

$$U_1 + \sum_{e_k \in C_l} -U_{k+1} \geqslant 0$$

De plus comme seule la première composante de b est non nulle (et égale à 1), l'objectif est de minimiser U_1. On peut ainsi réécrire le programme linéaire dual comme suit :

$$\begin{array}{l} \text{Minimiser } U_1, \\ \text{sous les contraintes} \\ \left\{ \begin{array}{lll} \text{(3.3a)} & \forall A_a \in \mathcal{A} & \displaystyle\sum_{\substack{e_k=(i,j) \\ e_k \in A_a}} c_{i,j}\ U_{k+1} \geqslant 1 \\ \text{(3.3b)} & \forall C_c \in \mathcal{C} & \displaystyle\sum_{e_k \in C_l} U_{k+1} \leqslant U_1 \\ \text{(3.3c)} & & U \geqslant 0 \end{array} \right. \end{array} \tag{3.3}$$

Pour un problème d'optimisation sur un convexe K de $\mathbb{Q}^k$, on peut définir les deux problèmes suivants. Le premier est un problème optimisation, tel que celui que nous cherchons à résoudre sur $\mathcal{P}$.

Définition 3.3 (Problème d'optimisation forte (SOPT(K, C)))**.** *Soient un convexe K et un vecteur $C \in \mathbb{Q}^k$, trouver un vecteur $x \in K$ qui maximise ${}^tC \cdot x$ ou montrer que K est vide.*

Le second problème est un problème de séparation, consistant à décider si un vecteur appartient ou nom à un polyèdre et sinon, à fournir un hyperplan qui sépare le polyèdre du vecteur.

Définition 3.4 (Problème de séparation forte (SSEP(K, x)))**.** *Étant donnés un convexe K et un vecteur $x \in \mathbb{Q}^k$, décider si $x \in K$ et sinon, trouver un hyperplan qui sépare x de K ; plus précisément, trouver un vecteur $C \in \mathbb{Q}^k$ tel que ${}^tC \cdot x > \max\left\{{}^tC \cdot y \mid y \in K\right\}$.*

La méthode de l'ellipsoïde s'appuie sur l'équivalence entre les problèmes d'optimisation et de séparation, équivalence dont la preuve peut être trouvée dans [57, chapitre 6] et qui est exprimée par le théorème suivant.

Théorème 3.2 (Théorème 6.4.9 dans [57]). *Chacun des deux problèmes*

- *séparation forte,*
- *optimisation forte,*

peut être résolu en temps oracle-polynomial pour tout polyèdre "bien décrit" si on connaît un oracle pour l'autre problème.

Pour montrer que le convexe $K_{\mathcal{D}}$ correspondant au problème d'optimisation $\mathcal{D}$ est un polyèdre "bien décrit", nous utilisons la définition suivante, issue de [57] :

Définition 3.5. *Soit $P \subseteq \mathbb{R}^k$ un polyèdre et ϕ un entier positif,*

(i) On dit que (P) à une "complexité de facette" d'au plus ϕ s'il existe un système d'inégalités à coefficients rationnels dont la solution est (P) et tel que la longueur du codage de chaque inégalité du système est au plus ϕ

(ii) Un polyèdre "bien décrit" est un triplet $(P; k, \phi)$ où $P \subseteq \mathbb{R}^k$ est un polyèdre de "complexité de facette" d'au plus ϕ. La taille de codage $\langle P \rangle$ d'un polyèdre "bien décrit" $(P; k, \phi)$ est $\phi + k$.

Le polyèdre $K_{\mathcal{D}}$ est clairement exprimé par un système d'inégalités à coefficients rationnels. Les coefficients de ces inégalités sont 0, 1 ou un des $c_{i,j}$. Donc $(K_{\mathcal{D}}; k, 2 \log c_{\max})$ est un polyèdre bien décrit. Dans ce cas, le théorème suivant affirme qu'il est possible de reconstruire la solution de $\mathcal{P}$ à partir de l'exécution de la méthode des ellipsoïdes sur $\mathcal{D}$.

Théorème 3.3 (Théorème 6.5.15 dans [57]). *Il existe un algorithme de temps oracle-polynomial qui, pour chaque $c \in \mathbb{Q}^k$ et pour tout polyèdre bien décrit $(P; k, \phi)$, où la taille de ϕ est polynomiale en la taille des entrées et donné par un oracle de séparation forte dont chaque sortie est de taille au plus ϕ*

(i) trouve une solution du problème primal, ou

(ii) prouve que le problème admet des solutions non bornées ou pas de solution du tout.

Afin de résoudre le problème d'optimisation sur $\mathcal{P}$ en utilisant la méthode du théorème précédent, nous allons construire un oracle de séparation pour le problème dual $\mathcal{D}$.

3.1.4 Application au différentes communications collectives

Nous devons maintenant spécifier notre étude en fonction des primitives de communications collectives étudiées. Nous développons en détail le cas de la diffusion, pour lequel nous pouvons obtenir un algorithme polynomial, puis nous décrivons brièvement le cas de la distribution de données, également polynomial. Enfin nous examinons le cas de la diffusion restreinte, pour lequel cette méthode n'aboutit ni à un algorithme polynomial, ni à une preuve de NP-complétude, bien que l'échec de notre approche laisse présager sa complexité.

3.1.4.1 Diffusion de données

Considérons le problème de séparation forte pour le problème de la diffusion (Diffusion-SSEP) dans le dual $\mathcal{D}$:

Définition 3.6 (Diffusion-SSEP(G,P_{source},x))**.** *Étant donnés un graphe de plate-forme G, un nœud source P_{source} et un vecteur $x \in \mathbb{Q}^{|E|+1}$, existe-t-il un vecteur $C \in \mathbb{Q}^{|E|+1}$ tel que ${}^tC \cdot x > \max\left\{{}^tC \cdot y,\ y \in K_{\mathcal{D}}\right\}$, où $\mathcal{D}$ est le programme linéaire dual associé à une opération de diffusion dans G depuis P_{source}.*

Lemme 3.1. *Diffusion-SSEP(G,P_{source},x) peut être résolu en temps polynomial.*

Démonstration. Considérons une instance du problème de séparation forte $\mathcal{D}$, donnée par un vecteur x. Nous cherchons à montrer que $x \in K_{\mathcal{D}}$, ou à exhiber une contrainte qui n'est pas satisfaite en x. $K_{\mathcal{D}}$ est donné par 3 types de contraintes

$$\begin{cases} \text{Contrainte } \ I & x \geqslant 0, \\ \text{Contraintes } \ II & \forall C_c \in \mathcal{C} \quad \displaystyle\sum_{e_k \in C_l} x_{k+1} \leqslant x_1 \\ \text{Contraintes } \ III & \forall A_a \in \mathcal{A} \quad \displaystyle\sum_{\substack{e_k=(i,j) \\ e_k \in A_a}} c_{i,j}\ x_{k+1} \geqslant 1 \end{cases}$$

Nous examinons comment vérifier chaque type de contraintes :

- Étant donné x, la contrainte I peut évidemment être vérifiée en temps polynomial.
- Contraintes de type II : Il s'agit de vérifier pour chaque schéma de communication, si la somme des poids des arêtes intervenant dans le schéma de communication est inférieure à la valeur de x_1. Pour ceci, nous considérons le graphe sur lequel les schémas de communication sont des couplages, pondérés par les x_{k+1}.
  Modèle unidirectionnel : Nous considérons le graphe pondéré $G_{\mathcal{M}} = (V, E, c_{\mathcal{M}})$, où $c_{\mathcal{M}}(e_k) = x_{k+1}$.
  Modèle bidirectionnel : Nous considérons le graphe biparti G_B construit au chapitre précédent pour caractériser les schémas de communication sous le modèle bidirectionnel et nous pondérons les arêtes de ce graphe par les x_{k+1} : dans ce cas, on a $G_{\mathcal{M}} = (V_{\text{out}} \cup V_{\text{in}}, E_B, c_{\mathcal{M}})$, où $c_{\mathcal{M}}(e'_k) = x_{k+1}$ (en notant e'_k l'arête de G_B correspondant à e_k).
  Quelque soit le modèle utilisé, on peut calculer dans $G_{\mathcal{M}}$ un couplage $M_{\max}$ de poids maximal $w_{\max}$ en temps polynomial [43]. Suivant que le poids de ce couplage est supérieur ou inférieur à x_1, nous pouvons conclure.
  - Si $w_{\max} \leqslant x_1$, alors toutes les inégalités de type II sont vérifiées puisque la contrainte est vérifiée pour le couplage de poids maximum ;
  - Si $w_{\max} > x_1$, alors l'inégalité correspondant à $M_{\max}$ n'est pas satisfaite, i.e.

$$\sum_{e_k \in M_{\max}} x_{k+1} > x_1$$

  et $\{x, \sum_{e_k \in M_{\max}} x_{k+1} = x_1\}$ est un hyperplan séparant x de $K_{\mathcal{D}}$.

– Contraintes de type *III* : nous devons vérifier que la sommes des poids des arêtes de tous schéma d'allocation est plus grande que x_1. Rappelons que nous étudions ici uniquement le cas de la diffusion de données, ou les schéma d'allocation sont des arbres couvrant. Considérons le graphe pondéré $G_A = (V, E, c_A$, où $c_A(e_k = (P_i, P_j)) = c_{i,j} \cdot x_{k+1}$. On peut calculer dans G_A un arbre $A_{\min}$ enraciné en P_{source} de poids minimal $w_{\min}$ en temps polynomial [43]. Comme précédemment, nous distinguons deux cas :
  – Si $w_{\min} \geqslant 1$, alors toutes les inégalités de type *III* sont vérifiées, puisque la contrainte est vérifiée pour l'arbre de poids minimal.
  – Sinon, l'inégalité correspondant à $A_{\min}$ n'est pas satisfaite, i.e.

$$\sum_{e_k=(P_i,P_j)\in A_{\min}} c_{i,j} \cdot x_{k+1} < 1$$

  et $\{x, \sum_{e_k=(P_i,P_j)\in A_{\min}} c_{i,j} x_{k+1} = 1\}$ est un hyperplan séparant x de $K_{\mathcal{D}}$.

On peut donc, en temps polynomial, vérifier que x est dans le convexe $K_{\mathcal{D}}$ ou exhiber un hyperplan séparant x de $K_{\mathcal{D}}$. ■

En utilisant les résultats précédents, on peut alors montrer le résultat suivant :

Théorème 3.4. *Étant donnés une plate-forme G et un nœud source P_{source}, il est possible, en temps polynomial en la taille de la plate-forme G,*
– *de calculer le débit optimal d'une opération de diffusion dans G depuis P_{source} (Lemme 3.1),*
– *de trouver l'ensemble des arbres sur lesquels sont réalisées les diffusions et l'ensemble des couplages sur lesquels sont réalisés les échanges de messages, (Théorème 3.3),*
– *et de construire un ordonnancement périodique de débit optimal (Théorème 2.1).*

Cependant, le temps de résolution d'un programme linéaire avec la méthode des ellipsoïdes étant grand en pratique, nous ne pouvons pas considérer que ce résultat fournit une méthode de résolution pratique du problème de la diffusion de données, par contre cette approche permet de connaître la complexité de ce problème pour les modèles de communication un port unidirectionnel et bidirectionnel.

3.1.4.2 Distribution de données

Nous reprenons le raisonnement précédent pour d'autres communications collectives, en commençant par la distribution de données. On peut appliquer la même méthode, en prenant pour schémas d'allocation ceux de la distribution de donnée. On rappelle qu'un schéma d'allocation dans ce cas est un collection de $|V_{\mathsf{cibles}}|$ routes sans cycle : une route $P_{\mathsf{source}} \rightsquigarrow P_i$ pour chaque processeur destination $P_i \in V_{\mathsf{cibles}}$. On rappelle aussi que l'on identifie un schéma d'allocation et la liste de arêtes qu'il utilise. En particulier pur la distribution de données, un schéma peut contenir plusieurs fois la même arête, si elle est présente dans plusieurs routes qui compose le schéma. En donnant cette nouvelle définition à $\mathcal{A}$, l'étude précédente reste valable avec le nouveau programme dual correspondant à la distribution de données, seul le problème de séparation se trouve modifié.

Définition 3.7 (Distribution-SSEP(G,P_{source},V_{cibles},x))**.** *Étant donnés un graphe de plate-forme G, un nœud source P_{source}, un ensemble de destinations V_{cibles} et un vecteur $x \in \mathbb{Q}^{|E|+1}$, existe-t-il un vecteur $C \in \mathbb{Q}^{|E|+1}$ tel que ${}^tC \cdot x > \max\left\{{}^tC \cdot y,\ y \in K_{\mathcal{D}}\right\}$, où $\mathcal{D}$ est le programme linéaire dual associé à une opération de distribution de données depuis P_{source} vers V_{cibles}.*

Lemme 3.2. *Distribution-SSEP(G,P_{source},V_{cibles},x) peut être résolu en temps polynomial.*

Démonstration. Pour montrer ce résultat, nous adaptons la preuve du lemme 3.1. Seules les contraintes de type *III* sont différentes, puisque ce sont les seules à faire apparaître les schémas d'allocation. Comme précédemment, pour ces contraintes, nous construisons le graphe pondéré $G_A = (V, E, c_A$, où $c_A(e_k = (P_i, P_j)) = c_{i,j} \cdot x_{k+1}$. Le but est encore une fois d'extraire un schéma d'allocation de poids minimal et de vérifier si son poids est supérieur à 1.

Le poids d'un schéma d'allocation, c'est-à-dire d'une collection de routes, est la somme des poids de chacune de ces routes. Le poids d'une route est la somme des poids des arêtes qu'elle emprunte. Si une arête est utilisée par deux routes d'un même schéma, son poids est comptabilisé deux fois dans le poids du schéma. Donc les contributions au poids d'un schéma de deux routes qui le composent sont indépendantes. Pour trouver un schéma de poids minimal, on peut ainsi chercher indépendamment des routes de poids minimal reliant la source à chacune des destinations, puis les assembler. Pour chaque cible $P_i \in V_{\text{cibles}}$, on peut calculer dans G_A une route de poids minimum de P_{source} à P_i en temps polynomial. On appelle $A_{\min}$ le schéma d'allocation correspondant à la collection de routes obtenues et $w_{\min}$ son poids. On sait que $w_{\min}$ est le poids minimal d'un schéma d'allocation. Comme précédemment, nous distinguons deux cas :

- Si $w_{\min} \geqslant 1$, alors toutes les inégalités de type *III* sont vérifiées, puisque la contrainte est vérifiée pour le schéma d'allocation de poids minimal.
- Sinon, l'inégalité correspondant à $A_{\min}$ n'est pas satisfaite, i.e.

$$\sum_{e_k=(P_i,P_j)\in A_{\min}} c_{i,j} \cdot x_{k+1} < 1$$

et $\{x, \sum_{e_k=(P_i,P_j)\in A_{\min}} c_{i,j} x_{k+1} = 1\}$ est un hyperplan séparant x de $K_{\mathcal{D}}$. ■

On est alors en mesure de montrer le même résultat que pour la diffusion :

Théorème 3.5. *Étant donnés une plate-forme G, un nœud source P_{source}, et un ensemble de cibles V_{cibles}, il est possible, en temps polynomial en la taille de la plate-forme G,*

- *de calculer le débit optimal d'une opération de distribution de données dans G depuis P_{source} vers V_{cibles} (Lemme 3.2),*
- *et de trouver l'ensemble des routes sur lesquels sont réalisées les diffusions et l'ensemble des couplages sur lesquels sont réalisés les échanges de messages (Théorème 3.3),*
- *et de construire un ordonnancement périodique de débit optimal (Théorème 2.1).*

3.1.4.3 Diffusion restreinte

Voyons maintenant le cas de la diffusion restreinte à un sous-ensemble de processeurs cibles V_{cibles}. Nous pouvons définir comme précédemment le problème de séparation dans le dual associé :

Définition 3.8 (Diffusion-Restreinte-SSEP(G,P_{source},V_{cibles},x))**.** *Étant donnés un graphe de plate-forme G, un nœud source P_{source}, un ensemble de destination V_{cibles} et un vecteur $x \in \mathbb{Q}^{|E|+1}$, existe-t-il un vecteur $C \in \mathbb{Q}^{cardE+1}$ tel que ${}^tC \cdot x > \max\left\{{}^tC \cdot y,\ y \in K_{\mathcal{D}}\right\}$, où $\mathcal{D}$ est le programme linéaire dual associé à une opération de diffusion restreinte depuis P_{source} vers V_{cibles}.*

Là encore, la résolution du problème de séparation est identique sauf pour les contraintes de types *III*. Si on tente d'appliquer la méthode précédente à ces contraintes, on construit le graphe pondéré $G_A = (V, E, c_A$, où $c_A(e_k = (P_i, P_j)) = c_{i,j} \cdot x_{k+1}$ et on cherche à trouver un schéma d'allocation de poids minimal sur G_A. Rappelons que pour la diffusion restreinte, un schéma d'allocation est un arbre enraciné en P_{source} et couvrant les processeurs de V_{cibles}. Malheureusement, trouver un tel arbre de poids minimal est un problème NP-complet, appelé problème de Steiner [65]. Nous ne pouvons donc pas utiliser cette méthode pour résoudre le problème de la diffusion restreinte.

Même si le fait que le problème de séparation dans le dual se ramène à un problème NP-complet, nous ne pouvons conclure à ce stade que le problème de la diffusion restreinte est NP-complet. Nous montrerons plus loin dans ce manuscrit, en utilisant une autre méthode, que la diffusion restreinte de données est NP-complet.

3.1.4.4 Autres primitives de communications collectives

Nous appliquons ici la méthode précédente à deux autres primitives de communications collectives, proche de la distribution et/ou de la diffusion de données. L'adaptation de la méthode est assez simple, grâce à la formulation en schémas d'allocation : seule la signification de l'ensemble $\mathcal{A}$ de ces schémas change dans les programmes linéaires $\mathcal{P}$ et $\mathcal{D}$, et donc dans la définition du convexe $K_{\mathcal{D}}$.

Échange total Dans cette primitive de communication, chaque processeur possède une donnée et veut la diffuser à tous les autres processeurs. Cette opération est appelé `MPI_AllGather` dans le standard de communication MPI. Dans sa version pipelinée, elle consiste à effectuer simultanément une opération de diffusion pipelinée depuis chaque processeur de la plate-forme.

Les schémas d'allocation associés à cette opération sont directement adaptés de ceux de la diffusion : un schéma d'allocation pour l'échange total contient un arbre couvrant enraciné en chaque processeur. En notant $\mathcal{A}_i$ l'ensemble des arbres couvrants enracinés en P_i, l'ensemble des schémas d'allocation est $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n$.

Comme précédemment, on peut définir le problème de séparation associé à cette opération.

Définition 3.9 (Échange-Total-SSEP(G,x))**.** *Étant donnés un graphe de plate-forme G et un vecteur $x \in \mathbb{Q}^{|E|+1}$, existe-t-il un vecteur $C \in \mathbb{Q}^{cardE+1}$ tel que ${}^tC \cdot x > \max\left\{{}^tC \cdot y,\ y \in K_{\mathcal{D}}\right\}$, où $\mathcal{D}$ est le programme linéaire dual associé à une opération d'échange total sur la plate-forme G.*

Comme pour la diffusion, nous pouvons construire un oracle de séparation pour le problème dual associé.

Lemme 3.3. *Échange-Total-SSEP(G,x) peut être résolu en temps polynomial.*

Démonstration. Les contraintes de type *I* ou *II* se traitent comme précédemment, puisqu'elles ne dépendent pas de la primitive de communication étudiée, mais uniquement du modèle de communication.

On rappelle que les contraintes de type *III* sont les suivantes :

$$\forall A_a \in \mathcal{A} \quad \sum_{\substack{e_k=(i,j) \\ e_k \in A_a}} c_{i,j}\ x_{k+1} \geqslant 1$$

Nous construisons de nouveau le graphe pondéré $G_A = (V, E, c_A$, où $c_A(e_k = (P_i, P_j)) = c_{i,j} \cdot x_{k+1}$. Le but est d'extraire un schéma d'allocation de poids minimal et de vérifier si son poids est supérieur à 1.

Pour cette opération, un schéma d'allocation est une collection d'arbres de diffusion. Nous faisons la même remarque que pour la distribution de données : deux arbres d'un même schéma contribuent indépendamment au poids du schéma, qui est simplement la somme des poids des arêtes utilisées par chaque arbre. En particulier, une même arête présente dans deux arbres d'un même schéma sera comptabilisée deux fois pour le calcul du poids du schéma. Afin de trouver un schéma de poids minimal, on peut dont rechercher indépendamment des arbres de diffusion enracinés en P_i, pour chaque processeur P_i, puis les assembler.

On peut calculer dans G_A un arbre $A^i_{\min}$ enraciné en P_i de poids minimal $w^i_{\min}$ en temps polynomial [43]. Nous considérons la collection d'arbres $(A^1_{\min}, \ldots, A^{|V|}_{\min})$, et son poids $w_{\min} = \sum_i w^i_{\min}$. On distingue deux cas :

- Si $w_{\min} \geqslant 1$, alors toutes les inégalités de type *III* sont vérifiées, puisque la contrainte est vérifiée pour le schéma poids minimal.
- Sinon, l'inégalité correspondant à $A_{\min}$ n'est pas satisfaite, i.e.

$$\sum_{e_k=(P_i,P_j)\in A_{\min}} c_{i,j} \cdot x_{k+1} < 1$$

  et $\{x, \sum_{e_k=(P_i,P_j)\in A_{\min}} c_{i,j} x_{k+1} = 1\}$ est un hyperplan séparant x de $K_{\mathcal{D}}$.

On peut donc, en temps polynomial, vérifier que x est dans le convexe $K_{\mathcal{D}}$ ou exhiber un hyperplan séparant x de $K_{\mathcal{D}}$ ■

Comme pour la diffusion, en utilisant les résultats précédents, on peut alors montrer le résultat suivant :

Théorème 3.6. *Étant donnée une plate-forme G, il est possible, en temps polynomial en la taille de la plate-forme G,*

- *de calculer le débit optimal d'une opération d'échange total dans G (Lemme 3.3),*
- *de trouver l'ensemble des arbres sur lesquels sont réalisées les diffusions et l'ensemble des couplages sur lesquels sont réalisés les échanges de messages, (Théorème 3.3),*
- *et de construire un ordonnancement périodique de débit optimal (Théorème 2.1).*

Échange personnalisé L'opération de communication collective d'«échange personnalisé», connue sous sa dénomination anglaise «all-to-all» ou «gossip» (commérage), est proche de la distribution de données : dans cette opération, chaque processeur souhaite envoyer une donnée différente à chacun des autres processeurs. La version pipelinée de cette opération consiste en une superposition d'opérations de distribution de données, une distribution étant initiée par chaque processeur.

Dans ce cas, les schémas d'allocation sont encore des ensembles de routes : en notant $\mathcal{R}_{a,b}$ l'ensemble des routes sans cycle de P_a à P_b, on a :

$$\mathcal{A} = \prod_{\substack{a,b \in V \\ a \neq b}} \mathcal{R}_{a,b}.$$

Nous définissons le problème de séparation associé à cette opérations :

Définition 3.10 (Échange-Personnalisé-SSEP(G,x))**.** *Étant donnés un graphe de plate-forme G et un vecteur $x \in \mathbb{Q}^{|E|+1}$, existe-t-il un vecteur $C \in \mathbb{Q}^{cardE+1}$ tel que ${}^tC \cdot x > \max\left\{{}^tC \cdot y,\ y \in K_{\mathcal{D}}\right\}$, où $\mathcal{D}$ est le programme linéaire dual associé à une opération d'échange personnalisé sur la plate-forme G.*

Comme pour la diffusion, nous pouvons construire un oracle de séparation pour le problème dual associé.

Lemme 3.4. *Échange-Personnalisé-SSEP(G,x) peut être résolu en temps polynomial.*

Démonstration. Les contraintes de type *I* ou *II* se traitent comme précédemment, puisqu'elles ne dépendent pas de la primitive de communication étudiée, mais uniquement du modèle de communication.

On rappelle que les contraintes de type *III* sont les suivantes :

$$\forall A_a \in \mathcal{A} \quad \sum_{\substack{e_k=(i,j) \\ e_k \in A_a}} c_{i,j}\ x_{k+1} \geqslant 1$$

Nous construisons de nouveau le graphe pondéré $G_A = (V, E, c_A,$ où $c_A(e_k = (P_i, P_j)) = c_{i,j} \cdot x_{k+1}$. Le but est d'extraire un schéma d'allocation de poids minimal et de vérifier si son poids est supérieur à 1.

Pour cette opération, un schéma d'allocation est une collection de routes de P_i à P_j pour toute paire de processeurs (P_i, P_j). De même que pour la diffusion, les différentes routes constituant un schéma contribuent indépendamment à son poids, qui est la somme des poids de chacune de ses routes. Pour trouver un schéma d'allocation de poids minimal, on construit en temps polynomial pour chaque paire de processeurs (P_i, P_j) une route $R_{i,j}$ dans G_A de poids minimal $w_{\min}^{i,j}$. On rassemble ensuite ces routes en un schéma $A_{\min}$, dont le poids vaut $w_{\min} = \sum_{i \neq j} w_{\min}^{i,j}$. On distingue deux cas :

- Si $w_{\min} \geqslant 1$, alors toutes les inégalités de type *III* sont vérifiées, puisque la contrainte est vérifiée pour le schéma poids minimal.
- Sinon, l'inégalité correspondant à $A_{\min}$ n'est pas satisfaite, i.e.

$$\sum_{e_k=(P_i,P_j) \in A_{\min}} c_{i,j} \cdot x_{k+1} < 1$$

et $\{x, \sum_{e_k=(P_i,P_j) \in A_{\min}} c_{i,j} x_{k+1} = 1\}$ est un hyperplan séparant x de $K_{\mathcal{D}}$.

On peut donc, en temps polynomial, vérifier que x est dans le convexe $K_{\mathcal{D}}$ ou exhiber un hyperplan séparant x de $K_{\mathcal{D}}$ ■

Comme pour la diffusion, en utilisant les résultats précédents, on peut alors montrer le résultat suivant :

Théorème 3.7. *Étant donnée une plate-forme G, il est possible, en temps polynomial en la taille de la plate-forme G,*

- *de calculer le débit optimal d'une opération d'échange personnalisé dans G (Lemme 3.4),*
- *de trouver l'ensemble des arbres sur lesquels sont réalisées les diffusions et l'ensemble des couplages sur lesquels sont réalisés les échanges de messages, (Théorème 3.3),*
- *et de construire un ordonnancement périodique de débit optimal (Théorème 2.1).*

Réduction Pour adapter la méthode précédente à l'étude de la réduction, il faut adapter le programme linéaire 3.1 afin de prendre en compte les spécificités de la réduction :

- On ne peut plus considérer que tous les messages ont la même taille : il est par exemple envisageable que le message contenant résultat partiel $v_{[u,w]}$ ait une taille proportionnelle à $w-u$. Il faut donc introduire une nouvelle pondération spécifique dans la contrainte (3.1b).
- Il faut prendre en compte les tâches de calcul et donc introduire de nouvelles contraintes pour borner la quantité de travail d'un processeur.

Nous rappelons les notations pour l'opération de réduction :

- La taille du mesage contenant le résultat partiel $v_{[u,w]}$ est $\mathsf{size}([u,w])$. Son transfert sur une arête (i,j) demande un temps $\mathsf{size}([u,w]) \times c_{i,j}$.
- La tâche de calcul $T_{u,v,w}$ correspondant à l'opération $v_{[u,w]} \leftarrow v_{[u,v]} \oplus v_{[v+1,w]}$ (pour $u \leqslant v < w$) est de taille $\mathsf{flops}(u,v,w)$, et demande un temps de traitement $\mathsf{flops}(u,v,w) \times z_k$ sur le processeur P_k.

Avec ces notations, le programme linéaire 3.1 devient :

$$
\begin{array}{l}
\textsc{Maximiser } \rho_{\text{opt}} = \sum_{A_a \in \mathcal{A}} x_a, \\
\textsc{sous les contraintes} \\
\left\{ \begin{array}{lll}
 & \displaystyle\sum_{C_c \in \mathcal{C}} y_c \leqslant 1 & \\
\forall (i,j) \in E, & \displaystyle\sum_{\substack{C_c \in \mathcal{C} \\ (i,j) \in C_c}} y_c \geqslant \sum_{A_a \in \mathcal{A}} \sum_{(v_{[u,w]},(i,j)) \in A_a} x_a \cdot \mathsf{size}([u,w]) \cdot c_{i,j} & \\
\forall P_k \in V, & \displaystyle\sum_{A_a \in \mathcal{A}} \sum_{(T_{u,v,w},P_k) \in A} x_a \cdot \mathsf{flops}(u,v,w) \cdot z_k \leqslant 1 & \\
\forall A_a \in \mathcal{A}, & x_a \geqslant 0 & \\
\forall C_c \in \mathcal{C}, & y_c \geqslant 0 &
\end{array} \right.
\end{array}
\tag{3.4}
$$

Ce qui est équivalent à

$$
(\mathcal{P}) \quad \begin{array}{l}
\textsc{Maximiser } {}^t c \cdot X \\
\textsc{Sous les contraintes} \\
\left\{ \begin{array}{l} A \cdot X \leqslant b, \\ X \geqslant 0 \end{array} \right.
\end{array}
$$

avec la matrice A suivante :

	allocation A_l pour $l = 1, \ldots, \lvert\mathcal{A}\rvert$	couplage C_l pour $l = 1, \ldots, \lvert\mathcal{C}\rvert$
	0,0,...,0	1,1,...,1
arête $e_k = (i,j) \in E$	$\sum_{(v_{[u,w]},(i,j))\in A_l} \mathsf{size}([u,w]) \cdot c_{i,j}$	(-1 si $e_k \in C_l$, 0 sinon)
processeur $P_k \in V$	$\sum_{(T_{u,v,w},P_k)\in A_l} \mathsf{flops}(u,v,w) \cdot z_k$	0

et b tel que :
- $b_1 = 1$
- pour $k = 1, \ldots, |E|$, $b_{k+1} = 0$.
- pour $k = 1, \ldots, |V|$, $b_{|E|+k+1} = 0$.

On peut déterminer le programme linéaire dual, comme précédemment :

$$\text{MINIMISER } x_1 + \sum_{k=1}^{|V|} x_{|E|+k+1},$$

$$\text{SOUS LES CONTRAINTES}$$

$$\begin{cases} (I) \quad x \geqslant 0 \\ (II) \quad \forall C_c \in \mathcal{C} \quad \sum_{e_k \in C_l} x_{k+1} \leqslant x_1 \\ (III) \quad \forall A_a \in \mathcal{A} \quad \begin{cases} \sum_{(v_{[u,w]}, e_k=(i,j))\in A_a} c_{i,j} \cdot \mathsf{size}([u,w]) \cdot x_{k+1} \\ \quad + \sum_{(T_{u,v,w},P_k)\in A_l} \mathsf{flops}(u,v,w) \cdot z_k \cdot x_{|E|+k+1} \geqslant 1 \end{cases} \end{cases}$$

Définition 3.11 (Réduction-SSEP(G,x))**.** *Étant donnés un graphe de plate-forme G et un vecteur $x \in \mathbb{Q}^{|E|+|V|+1}$, existe-t-il un vecteur $C \in \mathbb{Q}^{|E|+|V|+1}$ tel que ${}^tC \cdot x > \max\left\{{}^tC \cdot y,\ y \in K_{\mathcal{D}}\right\}$, où $\mathcal{D}$ est le programme linéaire dual ci-dessus, associé à une opération de réduction sur la plate-forme G.*

Nous pouvons construire un oracle de séparation pour le problème dual associé.

Lemme 3.5. *Réduction-SSEP(G,x) peut être résolu en temps polynomial.*

Démonstration. Soit $x \in \mathbb{Q}^{|E|+|V|+1}$. Nous cherchons à savoir si x respecte toutes les contraintes du programme linéaire dual, et si non, à exhiber une contrainte qui n'est pas satisfaite : elle consitue un hyperplan séparant x de $K_{\mathcal{D}}$.

Les contraintes de type *I* ou *II* se traitent comme précédemment, puisqu'elles ne dépendent pas de la primitive de communication étudiée, mais uniquement du modèle de communication.

Nous cherchons maintenant à vérifier si les contraintes de type *III* sont satisfaites. Ceci revient à vérifier si tous les arbres de réduction sur G, munis de certaines pondérations spécifiques dictées par le vecteur x, sont d'un poids total supérieur à 1. Pour résoudre ce problème, on cherche à construire un arbre de réduction $A_{\min}$ de poids $w_{\min}$ minimal, où le poids w_a d'un schéma A_a est défini par :

$$w_a = \sum_{(v_{[u,w]}, e_k=(i,j))\in A_a} c_{i,j} \cdot \mathsf{size}([u,w]) \cdot x_{k+1} + \sum_{(T_{u,v,w},P_k)\in A_l} \mathsf{flops}(u,v,w) \cdot z_k \cdot x_{|E|+k+1}$$

Une fois un arbre de réduction de poids $w_{\min}$ obtenu, nous pourrons conclure comme précédemment :

- Si $w_{\min} \geqslant 1$, alors toutes les inégalités de type *III* sont vérifiées, puisque la contrainte est vérifiée pour l'arbre de réduction poids minimal.
- Sinon, l'inégalité correspondant à $A_{\min}$ n'est pas satisfaite, ce qui nous donne un hyperplan séparant x du convexe $K_{\mathcal{D}}$.

La recherche d'un arbre de réduction de poids minimal est plus complexe que pour les autres primitives de communication collective, mais peut également être faite en temps polynomial.

Pour toute arête (i,j), on note $C(i,j) = x_{k+1} \times c_{i,j}$ et pour tout processeur P_k, on note $Z(k) = x_{|E|+k+1} \times z_k$. Le poids d'un arbre de réduction s'écrit alors :

$$w_a = \sum_{(v_{[u,w]}, e_k=(i,j))\in A_a} \mathsf{size}([u,w]) \cdot C(i,j) + \sum_{(T_{u,v,w},P_k)\in A_l} \mathsf{flops}(u,v,w) \cdot Z(k)$$

Il s'agit maintenant de trouver un arbre de réduction de poids minimal, où le poids de l'arbre est défini comme la somme des poids des opérations qui le composent, telles que :

- le poids du transfert d'un message $v_{[u,w]}$ sur l'arête (i,j) est $\mathsf{size}([u,w]) \times C(i,j)$,
- le poids du calcul d'une tâche $T_{u,v,w}$ sur le processeur P_k est $\mathsf{flops}(u,v,w) \times Z(k)$.

Nous allons maintenant utiliser des arbres de réduction partiels : ce sont des schémas de réduction dont la seule différence avec les arbres de réduction est qu'il ne permettent pas d'obtenir comme résultat la valeur $v_{[0,N]}$ en P_{cible}, mais n'importe quelle valeur $v_{[u,w]}$ en n'importe quel processeur P_k.

On appelle $X([u,w],P_k)$ un arbre de réduction partiel permettant d'obtenir le message $v_{[u,w]}$ en P_k, dont le poids $x([u,w],P_k)$ est minimal.

On note également $Y([u,w],P_k)$ un arbre de réduction partiel permettant d'obtenir le message $v_{[u,w]}$ par un calcul local en P_k, où la valeur dont le poids $y([u,w],P_k)$ est minimal. La seule différence avec $X([u,w],P_k)$ est que, dans $X([u,w],P_k)$ la valeur $v_{[u,w]}$ peut être transmise par un autre processeur, alors que dans $Y([u,w],P_k)$ elle doit nécessairement être calculée sur place par une tâche $T_{u,v,w}$ $(u \leqslant v < w)$.

Avec ces notations, l'arbre que nous cherchons est $X([0,N],P_{\mathsf{cible}})$, et a un poids $x([0,N],P_{\mathsf{cible}})$.

On appelle également $C(P_i \rightsquigarrow P_j)$ le poids d'un plus court chemin entre P_i et P_j dans le graphe pondéré (V,E,C), avec les coûts des arêtes $C(i,j)$ définis ci-dessus.

Initialisation Au départ, les processeurs P_{r_i} possèdent la valeur $v_{[i,i]}$. Les arbres $X([i,i], P_{r_i})$ ne comportent donc aucune tâche de calcul et aucun transfert, et sont de poids nul : $x([i,i], P_{r_i}) = 0$.

Récurrence On peut calculer les arbres X et Y avec une récurrence croisée :

- Le résultat de $Y([u,w], P_k)$ doit être calculé localement par une tâche $T_{u,v,w}$. Pour obtenir un arbre de poids minimum, les deux données de cette tâches doivent être formées par les arbres $X([u,v], P_k)$ et $X([v+1,w], P_k)$. On obtient la formule de récurrence suivante sur les poids :

$$y([u,w], P_k) = \min_{u \leqslant v < w} \left\{ \mathsf{flops}(u,v,w) \times Z(k) + x([u,v], P_k) + x([v+1,w], P_k) \right\}$$

  Connaissant l'indice v de ce minimum, on peut également construire l'arbre de réduction partiel $Y([u,w], P_k)$, en assemblant $X([u,v], P_k)$, $X([v+1,w], P_k)$ et la tâche de calcul $(T_{u,v,w}, P_k)$.
- Le résultat de $X([u,w], P_k)$ peut être soit celui de $Y([u,w], P_k)$, soit le résultat d'un transfert, le message $v_{[u,w]}$ ayant été calculé sur un processeur distant P_i, et transféré jusqu'à P_k en passant par un plus court chemin :

$$x([u,w], P_k) = \min \left\{ y([u,w], P_k), \min_{P_i \in V - \{P_k\}} \left\{ y([u,w], P_i) + \mathsf{size}([u,w]) \times C(P_i \rightsquigarrow P_k) \right\} \right\}$$

  En admettant que $C(P_k \rightsquigarrow P_k) = 0$, on a alors

$$x([u,w], P_k) = \min_{P_i \in V} \left\{ y([u,w], P_i) + \mathsf{size}([u,w]) \times C(P_i \rightsquigarrow P_k) \right\}$$

  En sachant quel processeur P_i est choisi pour ce minimum, on peut créer l'arbre $X([u,w], P_k)$ en rajoutant les transfert de la valeur $v_{[u,w]}$ le long d'un plus court chemin de P_i à P_k à l'arbre $Y([u,w], P_i)$.

Nous proposons l'algorithme suivant pour calculer l'arbre $X([0,N], P_{\mathsf{cible}})$, une fois les plus courts chemins $C(P_i \rightsquigarrow P_j)$ obtenus. Cet algorithme utilise des notions proches de la programmation dynamique, en calculant tous les $X([u,w], P_k)$ et les $Y([u,w], P_k)$ par $w - u$ croissants.

1: **Pour** $P_{r_i} \in \mathcal{R}$ **:**
2: $x([i,i], P_{r_i}) \leftarrow 0$
3: $X([i,i], P_{r_i}) \leftarrow \emptyset$
4: **Pour** $s = 1, \ldots, N$ **:**
{on calcule les arbres partiels de poids minimum formant des valeurs issues de la réduction d'intervalles de longueur s}
5: **Pour** $u = 0, \ldots, N - s$ **:**
6: $w = u + s$
7: $y([u,w], P_k) \leftarrow \min_{u \leqslant v < w} \{\mathsf{flops}(u,v,w) \times Z(k) + x([u,v], P_k) + x([v+1,w], P_k)\}$, on note v l'indice du minimum.
8: $Y([u,w], P_k) \leftarrow X([u,v], P_k) \cup X([v+1,w], P_k) \cup (T_{u,v,w}, P_k)$.
9: **Pour** $u = 0, \ldots, N - s$ **:**
10: $w = u + s$
11: $x([u,w], P_k) \leftarrow \min_{P_i \in V} \{y([u,w], P_i) + \mathsf{size}([u,w]) \times C(P_i \rightsquigarrow P_k)\}$, on note P_i le processeur réalisant le minimum
12: $X([u,w], P_k) \leftarrow Y([u,w], P_i)$
13: **Si** $i \neq k$ **Alors**
14: **Pour** toute arête e sur un plus court chemin de P_i à P_k **:**
15: $X([u,w], P_k) \leftarrow X([u,w], P_k) \cup (v_{[u,w]}, e)$

Cette algorithme permet de calculer un arbre de réduction $X([0,N], P_{\mathsf{cible}})$ de poids $x([0,m], P_{\mathsf{cible}})$ minimum.

Démonstration. Nous utilisons l'invariant de boucle suivant :

$$P(s) \begin{cases} \text{Après l'exécution de l'itération } s \text{ de la boucle de la ligne 4, les} \\ \text{arbres } X \text{ et } Y \text{ (et leurs poids) correspondant à des valeurs } v_{[u,w]} \\ \text{tels que } w - u \leqslant s \text{ sont correctement calculés.} \end{cases}$$

Après l'initialisation (donc après l'itération $s = 0$) cette assertion est vérifiée.

Supposons qu'elle soit juste après l'itération $s-1$. Pour calculer les valeurs de $Y([u,w], P_k)$, on utilise la formule de récurrence établie précédemment, et le calcul local ne fait intervenir que des valeurs provenant d'intervalles de tailles inférieures ou égales à $s-1$, qui sont correctes d'après l'invariant. A l'issue de la boucle des lignes 5-8, les Y et leurs poids y sont corrects, pour toutes les valeurs $v_{[u,w]}$ avec $w - u \leqslant s$.

Le calcul des $X([u,w], P_k)$ pour $w - u = s$ s'appuie quant à lui sur celui des $Y([u,w], P_k)$ que l'on vient de calculer, et sur la formule de récurrence établie précédemment. Donc les X et leurs poids x sont corrects pour toutes les valeurs $v_{[u,w]}$ avec $w - u \leqslant s$.

L'invariant est donc vérifié pour s. ■

De plus, cet algorithme s'exécute en $O(N^3 \times p)$.

Ainsi, on a construit un arbre de poids minimal, qui nous permet, en temps polynomial, de vérifier que x est dans le convexe $K_{\mathcal{D}}$ ou d'exhiber un hyperplan séparant x de $K_{\mathcal{D}}$ ■

Comme pour la diffusion, en utilisant les résultats précédents, on peut alors montrer le résultat suivant :

Théorème 3.8. *Étant donnés une plate-forme G, un ensemble des processeurs $\mathcal{R}$ et un processeur P_{cible}, il est possible, en temps polynomial en la taille de la plate-forme G,*
- *de calculer le débit optimal d'une opération de réduction dans G (Lemme 3.5),*
- *de trouver un ensemble pondéré d'arbres de réduction et un ensemble pondéré de couplages sur lesquels sont réalisés les échanges de messages, (Théorème 3.3),*
- *et de construire un ordonnancement périodique de débit optimal pour la réduction (Théorème 2.1).*

Dans cette partie, nous avons obtenu des résultats de complexité pour les opérations de diffusion et de distribution de données, ainsi que pour certaines de leur variante. Cependant, la technique proposée ne fournit pas d'algorithme utilisable en pratique pour calculer une solution optimale, puisqu'on utilise la méthode des ellipsoïdes pour résoudre le programme linéaire. Dans la partie suivante, nous développons une méthode efficace pour le cas particulier du modèle un-port bidirectionnel.

3.2 Découpler pour optimiser

Dans cette partie, nous concentrons notre étude sur le modèle de communication un-port bidirectionnel, dans lequel un processeur peut effectuer simultanément une émission et une réception de messages. Dans ce modèle, nous allons voir qu'il est possible de découpler la recherche de couplages de la recherche de schémas d'allocation.

3.2.1 Recherche de couplages sous le modèle bidirectionnel

Pour ceci, nous remplaçons la contrainte (3.1b) du programme linéaire 3.1, qui indiquait que toutes les communications nécessaires aux schémas d'allocation devaient être "recouvertes" par des couplages, par les contraintes suivantes :

$$\forall (i,j) \in E, \qquad \sum_{\substack{A_a \in \mathcal{A} \\ (i,j) \in A_a}} x_a \cdot c_{i,j} \leqslant T_{i,j} \tag{3.5}$$

$$\forall P_i \in V, \qquad \sum_{(i,j) \in E} T_{i,j} \leqslant 1 \tag{3.6}$$

$$\forall P_i \in V, \qquad \sum_{(j,i) \in E} T_{j,i} \leqslant 1 \tag{3.7}$$

Rappelons que x_a est le nombre moyen de messages empruntant le schéma d'allocation A_a, $c_{i,j}$ le temps nécessaire à la transmission d'un message sur l'arête (i,j). On note $T_{i,j}$ est le temps d'occupation d'une arête. La première contrainte (3.5) décrit l'occupation d'une arête : le temps nécessaire à transmettre tous les messages qui passent par cette arête doit être inférieur au temps d'occupation de l'arête. Les contraintes suivantes sont les contrainte un-port pour l'émission de messages et pour la réception : la contrainte (3.6) indique que le temps moyen que passe le processeur P_i à émettre des messages sur une de ses arêtes sortantes et inférieur à 1 unité de temps. Comme les émissions doivent être sérialisées, c'est la somme des temps d'occupation

des arêtes sortantes qui est considérée. La dernière contrainte (3.7) impose la même restriction, mais pour les communications entrantes en P_i.

Nous allons montrer que si les communications nécessaires aux schémas d'allocation respectent ces contraintes, alors nous pouvons construire un ensemble de couplages dans le graphe biparti G_B qui permettent d'effectuer ses communications.

Théorème 3.9. *Soit $\{(A_a, x_a), A_a \in \mathcal{A}\}$ une collection pondérée de schémas d'allocation, vérifiant les contraintes (3.5), (3.6) et (3.7). On peut alors construire un ensemble d'au plus $|E|$ couplages décrivant les communications pour le modèle un-port bidirectionnel qui vérifient les contraintes (3.1a), et (3.1b) du programme linéaire 3.1 en temps polynomial.*

Démonstration. Nous construisons le graphe biparti $G_B = (V_{\text{out}} \cup V_{\text{in}}, E_B, c_B)$ comme introduit au chapitre précédent :

- Chaque nœud P_i de G est transformé en un nœud de V_{out} et un nœud de V_{in}, l'un $P_i^{\text{out}} \in V_{\text{out}}$ représentant l'activité d'émission et l'autre $P_i^{\text{in}} \in V_{\text{in}}$ représentant l'activité de réception.
- Pour toute arête (P_i, P_j) dans G, on construit une arête $(P_i^{\text{out}}, P_j^{\text{in}})$ dans E_B.
- Les arêtes sont pondérées par les temps des communications nécessaires aux transferts des messages pour les schémas d'allocation :

$$c_B(i,j) = \sum_{\substack{A_a \in \mathcal{A} \\ (i,j) \in A_a}} x_a \cdot c_{i,j}$$

D'après le théorème de König pour les graphe bipartis [88, vol.A chapitre 20], il est possible de décomposer le graphe biparti G_B en une somme pondérée d'au plus $|E_B|$ couplages $(C_1, y_1), \ldots, (C_{|E|}, y_{|E|})$, où $|E_B|$ représente le nombre d'arêtes dans le graphe biparti ($|E_B| \leqslant |E|$). On a en particulier :

$$\forall (i,j) \in E, \quad \sum_{\substack{C_c \in \mathcal{C} \\ (i,j) \in C_c}} y_c = c_B(i,j) \tag{3.8}$$

De plus, le poids cumulé des couplages (donc le temps nécessaire pour réaliser l'ensemble des communications) est inférieur ou égal au degré maximal de chaque nœud du graphe biparti :

$$\sum_{i=1}^{|E_B|} y_i \leqslant \Delta_{\max}$$

Le degré d'un nœud P_i^{out} de V_{out} est $\sum_{(i,j)\in E} c_B(j,i)$, et le degré d'un nœud P_i^{in} de V_{in} est $\sum_{(j,i)\in E} c_B(i,j)$, donc on peut réécrire l'inégalité précédente en :

$$\sum_{i=1}^{|E_B|} y_i \leqslant \max_{P_i \in Procs} \left(\sum_{(j,i)\in E} c_B(i,j), \sum_{(i,j)\in E} c_B(j,i) \right) \tag{3.9}$$

Par construction de c_B et grâce à (3.8), on sait que l'ensemble pondéré de couplages $(C_1, y_1), \ldots, (C_{|E|}, y_{|E|})$ vérifie la contrainte (3.1b) sur les arêtes, du programme linéaire 3.1.

De plus, comme l'ensemble de schémas d'allocation vérifie (3.5), (3.6) et (3.7) , on a $c_B(i,j) \leqslant T_{i,j}$, puis

$$\max_{P_i \in Procs} \left(\sum_{(j,i)\in E} C_B(i,j), \sum_{(i,j)\in E} C_B(j,i) \right) \leqslant 1$$

donc avec (3.9), on sait que la contrainte (3.1a) est vérifiée. ■

On peut donc se contenter chercher les schémas d'allocation indépendamment des couplages pour le cas bidirectionnel. Ceci pourrait se faire en résolvant le programme linéaire suivant, où les contraintes (3.9) et (3.1a) du programme linéaire 3.1 ont été remplacées par les contraintes (3.5), (3.6) et (3.7) :

$$\begin{array}{l} \text{MAXIMISER } \rho_{\text{opt}} = \sum_{A_a \in \mathcal{A}} x_a, \\ \text{SOUS LES CONTRAINTES} \\ \begin{cases} \forall (i,j) \in E, & \displaystyle\sum_{\substack{A_a \in \mathcal{A} \\ (i,j)\in A_a}} x_a \cdot c_{i,j} \leqslant T_{i,j} \\ \forall P_i \in V, & \displaystyle\sum_{(i,j)\in E} T_{i,j} \leqslant 1 \\ \forall P_i \in V, & \displaystyle\sum_{(j,i)\in E} T_{j,i} \leqslant 1 \\ \forall A_a \in \mathcal{A}, & x_a \geqslant 0 \end{cases} \end{array} \tag{3.10}$$

Cependant, ce programme linéaire est encore de trop grande taille : même si on a supprimé les variables correspondant aux couplages, il reste une variable par schéma d'allocation, ce qui représente potentiellement un nombre de variables exponentiel en la taille de la plate-forme. Dans le chapitre suivant, nous présentons d'autres techniques qui permettent de réduire la taille du programme linéaire, en fonction des primitives de communications étudiées.

3.2.2 Cas du un-port unidirectionnel

Nous avons vu que dans le cas du modèle un-port bidirectionnel, on pouvait découpler la recherche de couplages et celles de schémas d'allocation. Voyons maintenant ce qu'il en est pour le cas du modèle un-port unidirectionnel.

En s'inspirant des contraintes que nous avons élaborées pour le modèles bidirectionnel (équations (3.5), (3.6) et (3.7)), nous pouvons écrire des contraintes pour le cas unidirectionnel :

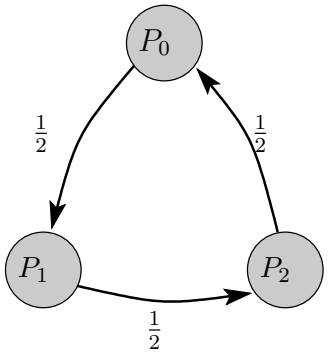
$$\forall (i,j) \in E, \qquad \sum_{\substack{A_a \in \mathcal{A} \\ (i,j)\in A_a}} x_a \cdot c_{i,j} \leqslant T_{i,j} \tag{3.11}$$

$$\forall P_i \in V, \qquad \sum_{(i,j)\in E} T_{i,j} + \sum_{(j,i)\in E} T_{j,i} \leqslant 1 \tag{3.12}$$

Ici encore, $T_{i,j}$ représente le temps d'occupation de l'arête (i,j). L'équation (3.11) indique que ce temps est suffisant pour effectuer les communications nécessaires et l'équation (3.12)

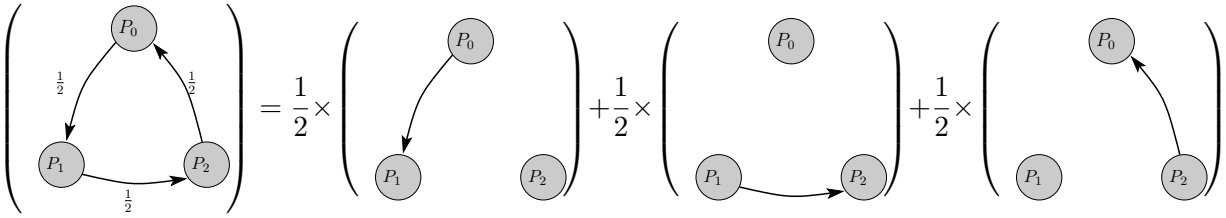
indique la congestion sur le processeur P_i : l'unique port de communication ne peut pas être occupé pendant plus d'une unité de temps, que ce soit pour envoyer des messages ou pour en recevoir.

Dans le cas bidirectionnel, nous avons vu qu'il était possible de reconstruire un ensemble de couplages pour des schémas d'allocation vérifiant ces contraintes. Malheureusement, ce n'est plus possible pour le modèle unidirectionnel. Considérons le plate-forme décrite par le graphe suivant.



Supposons que l'on veuille envoyer un message le long des trois arêtes, par exemple avec un schéma d'allocation $A_1 = \{(P_0, P_1), (P_1, P_2), (P_2, P_0)\}$ de poids $x_1 = 1$. Les contraintes ci-dessus sont bien vérifiées : chaque arête est utilisée pendant un temps $1/2$, donc chaque processeur reçoit un message pendant un temps $1/2$ et émet un message pendant un temps $1/2$.

Si on cherche maintenant à organiser les communications sous forme de couplages pour le modèle un-port unidirectionnel, on est obligé de former trois couplages de chacun une seule arête, ce qui conduit au découpage suivant :



Ainsi on ne peut organiser les communications en couplage en moins de $3/2$ unités de temps, au lieu de une unité de temps comme nécessaire.

Au vu de l'exemple précédent, nous ne pouvons pas utiliser la méthode précédente pour découpler la recherche de couplages et de schémas d'allocation pour le modèle un-port unidirectionnel. Le théorème de König permet, dans le modèle bidirectionnel, d'avoir égalité entre le temps nécessaire à organiser les communications en couplages, et le degré maximal d'un nœud, qu'on peut facilement contraindre dans le programme

Nous n'avons pas d'équivalent dans le cas unidirectionnel. Tout au plus pourrions-nous citer l'équivalent du théorème de Vizing pour les multigraphes, qui montre que $\chi(G) \leqslant \Delta(G) + \mu(G)$, où $\chi(G)$ est le nombre chromatique de G, $\Delta(G)$ son degré et $\mu(G)$ sa multiplicité. Le temps nécessaire pour effectuer toutes les communications, c'est-à-dire le nombre chromatique de G_A, est alors lié au degré du graphe et aux pondérations des arêtes. Mais la borne n'est pas nécessairement optimale, donc ce résultat n'est pas utilisable pour obtenir une organisation optimale des communications.

3.3 Conclusion

Dans ce chapitre, nous nous sommes intéressé à la complexité des opérations de communications collectives. Nous avons obtenu les résultats suivants :

- Nous avons décrit un programme linéaire général pour trouver une solution de débit optimal pour toute primitive de communications collectives, en l'écrivant comme ensembles pondérés de schémas d'allocation et de schémas de communication. Nous avons montré que même si le programme linéaire est de taille importante (non polynomial en la taille des données), il existe une solution optimale de taille polynomiale.
- Nous avons proposé une façon générale de résoudre ces problèmes, qui utilise la méthode de résolution d'un programme linéaire par les ellipsoïdes. Même si elle est en pratique peu efficace, elle permet de montrer que de nombreux problèmes de communications collectives, quelque soit le modèle de communication choisi, sont polynomiaux : la distribution de données, la diffusion et leurs variantes, ainsi que la réduction.
- Nous avons décrit un cadre général qui permet, dans le cas du modèle un-port bidirectionnel, de décoréler la recherche de schémas d'allocation de la recherche de couplages pour les communications. Cette méthode sera utilisée dans le chapitre suivant pour concevoir des algorithmes efficaces, de débit optimal, dans le cas du modèle un-port bidirectionnel.

Chapitre 4

Étude de cas pour le modèle un-port bidirectionnel

Dans le chapitre précédent, nous avons fourni les outils nécessaires à la recherche d'ordonnancements de débit optimal pour les communications collectives. Nous étudions maintenant leur application aux primitives qui nous intéressent.

En particulier, nous avons montré au chapitre précédent qu'en découplant la recherche de couplages pour le modèle un-port bidirectionnel de la recherche de schémas d'allocation, on pouvait découpler la recherche de schémas d'allocation et de couplages : on peut dans un premier temps rechercher les schémas d'allocation à l'aide d'un programme linéaire de taille plus petite, puis de cette décomposition en schémas d'allocation, trouver les couplages qui permettent d'organiser les communications. Cependant, le programme linéaire obtenu 3.10 ne peut être résolu de façon efficace, car il est possède encore trop de variables. Nous allons donc utiliser ici d'autres techniques, proches de celles introduites par Bertsimas et Gamarnik [26], pour optimiser cette résolution. Nous spécifions notre étude pour les différentes communications collectives étudiées, en commençant par le cas le plus simple, celui de la distribution de données. Nous étudierons ensuite le cas de la diffusion, puis celui de la réduction.

Ensuite, nous nous intéresserons aux communications collectives qui restent difficiles malgré cette simplification due au modèle un-port bidirectionnel. Nous montrerons ainsi que la diffusion restreinte est un problème NP-complet, tout comme une variante de la réduction, le calcul parallèle des préfixes.

4.1 Distribution de données

Rappelons que pour cette opération de communications collectives, le processeur source P_{source} souhaite envoyer des messages distincts à un ensemble de processeurs cibles V_{cibles}. Pour $P_k \in V_{\mathsf{cibles}}$ nous appelons messages de type k les messages que la source envoie à destination du processeur P_k.

Plutôt que de considérer les schémas d'allocation et leur débit, nous allons utiliser ici une approche proche de celle proposée par Bertsimas et Gamarnik, en considérant le flux de messages de type k depuis la source jusqu'à la destination P_k. Cependant, nous ne considérons pas le nombre total de messages envoyés à une destination, mais la valeur moyenne du nombre de

messages envoyés pendant une unité de temps : nous appelons $\mathsf{send}(P_i \to P_j, m_k)$ le nombre de messages de type k qui sont transmis par l'arête (i,j) pendant une unité de temps. Les $\mathsf{send}(P_i \to P_j, m_k)$ définissent un flux entre P_{source} et P_k :

$$\forall P_k \in V_{\mathsf{cibles}} \qquad \sum_{(P_{\mathsf{source}}, P_j) \in E} \mathsf{send}(P_{\mathsf{source}} \to P_j, m_k) = \rho \tag{4.1}$$

$$\forall P_k \in V_{\mathsf{cibles}} \qquad \sum_{(P_j, P_k) \in E} \mathsf{send}(P_j \to k, m_k) = \rho \tag{4.2}$$

$$\forall P_k \in V_{\mathsf{cibles}}, \forall P_i \neq P_k, P_{\mathsf{source}} \qquad \sum_{(P_j, P_i) \in E} \mathsf{send}(P_j \to P_i, m_k) = \sum_{(P_i, P_j) \in E} \mathsf{send}(P_i \to P_j, m_k) \tag{4.3}$$

L'équation 4.1 assure que tous les messages nécessaire à l'obtention d'un débit ρ sont émis par la source et l'équation 4.2 assure qu'ils sont reçus par la cible. Quant à l'équation 4.3, elle assure qu'en un processeur P_i il y a conservation des messages de type m_k (si P_i n'est ni la source ni la cible de ces messages) : il y autant de messages reçus que de messages émis en une unité de temps.

De plus, on peut facilement calculer le nombre total de messages traversant une arête et ainsi borner le temps d'occupation d'une arête. On rassemble ces contraintes dans le programme linéaire suivant, avec les contraintes du modèle un-port bidirectionnel introduites dans la partie 3.2.1.

$$\begin{aligned}
&\text{MAXIMISER } \rho_{\text{opt}}, \\
&\text{SOUS LES CONTRAINTES} \\
&\begin{cases}
\text{(4.4a)} & \forall P_k \in V_{\mathsf{cibles}} & \displaystyle\sum_{(P_{\mathsf{source}}, P_j) \in E} \mathsf{send}(P_{\mathsf{source}} \to P_j, m_k) = \rho_{\text{opt}} \\
\text{(4.4b)} & \forall P_k \in V_{\mathsf{cibles}} & \displaystyle\sum_{(P_j, P_k) \in E} \mathsf{send}(P_j \to P_k, m_k) = \rho_{\text{opt}} \\
\text{(4.4c)} & \begin{cases} \forall P_k \in V_{\mathsf{cibles}}, \forall P_i \\ P_i \neq P_k, P_{\mathsf{source}} \end{cases} & \displaystyle\sum_{(P_j, P_i) \in E} \mathsf{send}(P_j \to P_i, m_k) = \sum_{(P_i, P_j) \in E} \mathsf{send}(P_i \to P_j, m_k) \\
\text{(4.4d)} & \forall (i,j) \in E, & \displaystyle\sum_{P_k \in V_{\mathsf{cibles}}} \mathsf{send}(P_i \to P_j, m_k) \cdot c_{i,j} \leqslant T_{i,j} \\
\text{(4.4e)} & \forall P_i \in V, & \displaystyle\sum_{(i,j) \in E} T_{i,j} \leqslant 1 \\
\text{(4.4f)} & \forall P_i \in V, & \displaystyle\sum_{(j,i) \in E} T_{j,i} \leqslant 1
\end{cases}
\end{aligned} \tag{4.4}$$

Ce programme est de taille beaucoup plus petite que les programmes linéaires précédents : il comporte $|E| \cdot |V_{\mathsf{cibles}}|$ variables $\mathsf{send}(P_i \to P_j, m_k)$ et $|E|$ variables $T_{i,j}$, pour $|V| \cdot |V_{\mathsf{cibles}}| + 2|V| + |E|$ contraintes.

Théorème 4.1. *Le programme linéaire 4.4 donne une borne supérieure pour le débit d'un ordonnancement réalisant une série d'opérations de distribution de données.*

Démonstration. Considérons un ordonnancement pour une série de distribution de données depuis P_{source} vers les processeurs de V_{cibles}, sur la plate-forme G. On s'intéresse au débit moyen de messages distribués par cet ordonnancement. Le débit de l'ordonnancement est défini comme la limite de la suite N/D_N si celle-ci converge, mais dans tous les cas, le débit moyen est borné par $\limsup N/D_N$.

Considérons le trajet d'un message de cet ordonnancement, envoyé par la source à destination d'un processeur P_k. Il est possible que ce message soit envoyé plusieurs fois par la source, ou retransmis plusieurs fois par un autre processeur, puisque l'ordonnancement considéré est quelconque. Cependant, parmi les transferts de ce message, il existe au moins un ensemble de communications qui forment une route de P_{source} vers P_k : $P_{\mathsf{source}} = P_{r_0} \to P_{r_1} \to P_{r_2} \to \cdots \to P_{r_m} = P_k$, sans cycle ($P_{r_i} \neq P_{r_j}$ dès que $i \neq j$), et telle que le transfert du message sur l'arête $(P_{r_i}, P_{r_{i+1}})$ a lieu avant le transfert du message sur l'arête $(P_{r_{i+1}}, P_{r_{i+2}})$. Nous supprimons de l'ordonnancement considéré tous les autres transferts de ce message. L'ordonnancement obtenu est toujours valide (P_k reçoit ce message, les transferts des autres messages ne sont pas modifiés), et a une durée d'exécution inférieure ou égale à celle de l'ordonnancement initial. On procède de même pour tous les messages.

Soit $N \in \mathbb{N}$, on considère les opérations nécessaires aux N premières distributions de la série. Pour chaque arête (i, j), on appelle $N(P_i \to P_j, m_k)$ le nombre total de messages de type m_k envoyé sur l'arête (i, j) parmi les N premiers messages de type m_k. Les contraintes suivantes doivent être respectées par les $N(P_i \to P_j, m_k)$:

– Tout d'abord, après notre transformation de l'ordonnancement, les $N(P_i \to P_j, m_k)$ forment un flot de valeur N entre P_{source} et toute destination P_k. En particulier, les N messages à destination de P_k doivent être envoyés par la source :

$$\forall P_k \in V_{\mathsf{cibles}} \sum_{(P_{\mathsf{source}}, P_j) \in E} N(P_{\mathsf{source}} \to P_j, m_k) = N$$

– Ils doivent également être reçus par la cible :

$$\forall P_k \in V_{\mathsf{cibles}} \sum_{(P_j, P_k) \in E} N(P_j \to P_k, m_k) = N$$

– Il y a conservation des messages de type m_k sur P_i :

$$\forall P_k \in V_{\mathsf{cibles}}, \forall P_i P_i \neq P_k, P_{\mathsf{source}}, \quad \sum_{(P_j, P_i) \in E} N(P_j \to P_i, m_k) = \sum_{(P_i, P_j) \in E} N(P_i \to P_j, m_k)$$

– De plus, ces opérations ont lieu pendant un temps D_N, donc un processeur ne peut passer plus de D_N unités de temps à émettre ou à recevoir ces messages :

$$\forall P_i \in V, \sum_{(i,j) \in E} \sum_{P_k \in V_{\mathsf{cibles}}} N(P_i \to P_j, m_k) \cdot c_{i,j} \leqslant D_N$$

et

$$\forall P_i \in V, \sum_{(j,i) \in E} \sum_{P_k \in V_{\mathsf{cibles}}} N(P_j \to P_i, m_k) \cdot c_{j,i} \leqslant D_N$$

On pose

$$\mathsf{send}(P_i \to P_j, m_k) = \frac{N(P_i \to P_j, m_k)}{D_N}$$

et

$$T_{i,j} = \sum_{P_k \in V_{\text{cibles}}} \frac{N(P_i \to P_j, m_k)}{D_N} \cdot c_{i,j}.$$

Ces variables vérifient les équations du programme linéaire 4.4, donc on a $N/D_N \leqslant \rho_{\text{opt}}$. Comme on peut faire ce raisonnement pour toute valeur de N, on montre que

$$\limsup \frac{N}{D_N} \leqslant \rho_{\text{opt}}.$$

Donc ρ_{opt} est une borne supérieure au débit d'un ordonnancement pour une série de distributions de données. ■

Une fois obtenue la solution de ce programme linéaire, qui peut être résolu avec un solveur habituel, comme lp-solve [24], Maple [39] ou mupad [96], fondé sur la méthode du simplexe, on peut reconstruire un (petit) ensemble de schéma d'allocation décrivant la solution optimale.

Théorème 4.2. *Étant donné une solution optimale* $(\rho_{\text{opt}}, \mathsf{send}(P_i \to P_j, m_k))$ *du programme linéaire 4.4, on peut trouver en ensemble d'au plus* $|V_{\text{cibles}}| \times |E|$ *schémas d'allocation qui réalisent la solution optimale du problème de distribution de données sous le modèle un-port bidirectionnel.*

Démonstration. Pour une cible $P_k \in V_{\text{cibles}}$, nous construisons le graphe pondéré $G_k = (V, E, w_k)$ avec $w_k(i,j) = \mathsf{send}(P_i \to P_j, m_k)$. D'après les contraintes (4.1), (4.2), et (4.3), il existe un flot de valeurs ρ_{opt} entre P_{source} et P_k dans G_k. Nous pouvons donc décomposer ce graphe en une collection pondérée d'au plus $|E|$ routes $\{(R_1^k, \alpha_1^k), \dots (R_{|E|}^k, \alpha_{|E|}^k)\}$ de P_{source} à P_k, telles que :

$$\forall (i,j) \in E \sum_{\substack{u=1\dots E \\ (i,j) \in R_u^k}} \alpha_j^k = \mathsf{send}(P_i \to P_j, m_k)$$

et

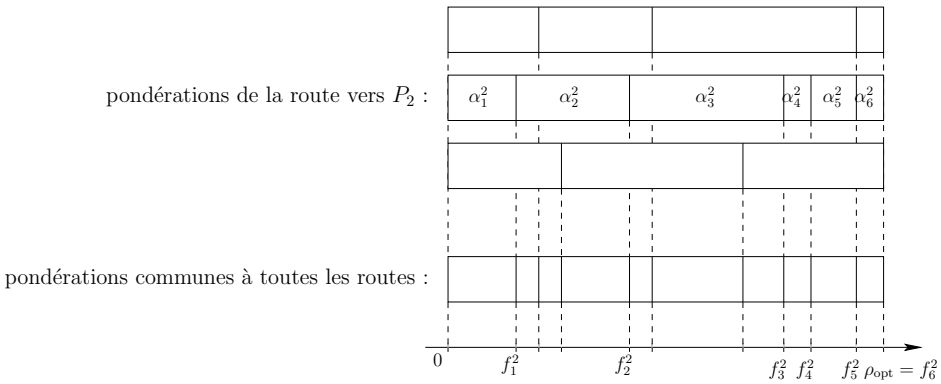
$$\sum_{u=1}^{E} \alpha_u^i = \rho_{\text{opt}}$$

On obtient donc un ensemble pondéré de routes pour chaque cible P_k. Il reste encore à rassembler ces routes dans un ensemble pondéré de schémas d'allocation. La seule difficulté est que les pondérations dont nous disposons sont différentes pour chaque route, et nous devons reconstruire un ensemble de pondérations commune à toutes les routes, afin de décrire l'ensemble des communications par une somme pondérée de schémas, où dans chaque schéma apparaît une route pour chaque cible.

Pour chaque ensemble de routes $\{R_u^k\}_{u=1\dots|E|}$, nous définissons les valeurs f_u^k :

$$f_u^k = \sum_{v=1}^{u} \alpha_v^i$$

Ces valeurs sont utilisées comme illustré la figure ci-dessous pour obtenir un ensemble pondéré de schémas d'allocation :



Pour ceci, on trie les f_u^k obtenus et on nomme f'_i la suite obtenue. Notons qu'il y a au plus $|V_{\mathsf{cibles}}| \times |E|$ éléments f'_i car chaque cible P_k de V_{cibles} contribue pour au plus $|E|$ valeurs f_u^k et il se peut que certaines valeurs se superposent. Pour chaque intervalle $[f'_i, f'_{i+1}]$ correspondant à deux valeurs consécutives, on crée un nouveau schéma d'allocation A_i de poids $f'_{i+1} - f'_i$ constitué des routes R_u^k telles que $f_u^k \leqslant f'_i$ et $f_{u+1}^k \geqslant f'_{i+1}$. ■

À l'aide des théorèmes 4.2, 3.9 et 2.1, on peut construire un ordonnancement périodique pour le problème de la distribution de données sous le modèle un-port bidirectionnel, s'appuyant sur la résolution d'un programme linéaire de taille $O(|V|^2) \times O(|V|^2)$. D'après le théorème 4.1, on sait que cette solution réalise le débit optimal.

4.2 Diffusion

4.2.1 Recherche efficaces d'arbres de diffusion

Pour le problème de la distribution de données, nous sommes parvenu à créer un programme linéaire «de petite taille» en nous appuyant sur la description d'un flot entre la source et les différents cibles, à la manière de Bertsimas et Gamarnik. Pour la diffusion, nous ne pouvons pas procéder de la même manière : comme des messages peuvent être dupliqués sur les nœuds de l'arbre qui possède plusieurs fils, et donc envoient une copie du message à chacun, il n'y a plus de flot entre la source et l'ensemble des cibles.

Cependant, si nous examinons une opération diffusion, nous pouvons a posteriori distinguer les communications qui sont utiles pour le processeur P_k, c'est-à-dire celles qui transportent un message dont le message que recevra P_k est une copie, ou encore, les communications $P_i \to P_j$ telles que P_j est un ancêtre de P_k dans l'arbre de diffusion. Si nous isolons les communications «utiles» à P_k, nous obtenons bien un flot de P_{source} à P_k. Nous pouvons faire de même pour tout processeur différent de la source, cependant les flots obtenus ne se superposent pas, mais ils se mélangent : une communication $P_i \to P_j$ peut être utile à plusieurs cibles, elle l'est potentiellement pour tout processeur P_k qui se trouve dans le sous-arbre enraciné en P_j, y compris P_j.

Nous allons donc écrire un programme linéaire semblable à celui de la distribution de données, en considérant le flot entre la source et chacune des cibles. En revanche, au lieu de considérer que les flots se superposent, comme le dicte la contrainte (4.4d), nous allons au contraire

que tous les flots se mélangent complètement, en écrivant :

$$\forall (i,j) \in E, \max_{P_k \in V} \{\mathsf{send}(P_i \to P_j, m_k)\} \cdot c_{i,j} \leqslant T_{i,j}$$

En nommant m_{k_0} le type de message qui réalise le maximum, $\mathsf{send}(P_i \to P_j, m_{k_0}) = \max_k \{\mathsf{send}(P_i \to P_j, m_k)\}$ ceci revient à considérer que pour tout $k \neq k_0$, les messages transitant sur l'arête (i,j) à destination de P_k forment un sous-ensemble des messages à destination de k_0. Cette hypothèse est forte et elle ne peut être justifiée a priori. Nous allons donc pour l'instant considérer que le programme linéaire nous fournit uniquement une borne supérieure sur le débit optimal et non le débit optimal lui même. Nous verrons ensuite qu'il est possible de construire une solution qui atteint cette borne supérieure, en utilisant un théorème de décomposition d'un graphe en une superposition d'arbres.

Pour plus de facilité, nous introduisons les variables $s(P_i \to P_j)$, désignant le nombre total de messages transitant sur l'arête (i,j) et définies par :

$$\forall (i,j) \in E, s(P_i \to P_j) = \max_{P_k \in V} \{\mathsf{send}(P_i \to P_j, m_k)\}.$$

Avec ces variables, la contrainte précédente se réécrit

$$\forall (i,j) \in E, s(P_i \to P_j) \cdot c_{i,j} \leqslant T_{i,j}.$$

Nous pouvons alors rassembler ces différentes contraintes en un programme linéaire.

MAXIMISER ρ_{opt},
SOUS LES CONTRAINTES

$$\begin{cases}
\text{(4.5a)} & \forall P_k \in V & \displaystyle\sum_{(P_{\mathsf{source}}, P_j) \in E} \mathsf{send}(P_{\mathsf{source}} \to P_j, m_k) = \rho_{\text{opt}} \\
\text{(4.5b)} & \forall P_k \in V & \displaystyle\sum_{(P_j, P_k) \in E} \mathsf{send}(P_j \to P_k, m_k) = \rho_{\text{opt}} \\
\text{(4.5c)} & \begin{cases} \forall P_k \in V, \forall P_i \\ P_i \neq P_k, P_{\mathsf{source}} \end{cases} & \displaystyle\sum_{(P_j, P_i) \in E} \mathsf{send}(P_j \to P_i, m_k) = \sum_{(P_i, P_j) \in E} \mathsf{send}(P_i \to P_j, m_k) \\
\text{(4.5d)} & \forall (i,j) \in E, & s(P_i \to P_j) = \max_{P_k \in V} \{\mathsf{send}(P_i \to P_j, m_k)\} \\
\text{(4.5e)} & \forall (i,j) \in E, & s(P_i \to P_j) \cdot c_{i,j} \leqslant T_{i,j} \\
\text{(4.5f)} & \forall P_i \in V, & \displaystyle\sum_{(i,j) \in E} T_{i,j} \leqslant 1 \\
\text{(4.5g)} & \forall P_i \in V, & \displaystyle\sum_{(j,i) \in E} T_{j,i} \leqslant 1
\end{cases} \tag{4.5}$$

Comme pour la distribution de données, le programme linéaire précédent donne une borne supérieur sur le débit atteignable.

Théorème 4.3. *Le programme linéaire 4.5 donne une borne supérieure pour le débit d'un ordonnancement réalisant une série d'opérations de diffusion.*

Démonstration. Considérons un ordonnancement pour une série de diffusions depuis P_{source}, sur la plate-forme G. On s'intéresse au débit moyen de messages diffusés par cet ordonnancement. Le débit de l'ordonnancement est défini comme la limite de la suite N/D_N si celle-ci converge, mais dans tous les cas, le débit moyen est borné par $\limsup N/D_N$.

Considérons la diffusion d'un message i de la série de messages à diffuser. Comme l'ordonnancement considéré est quelconque, il se peut qu'un processeur reçoive plusieurs copies de ce message. Dans ce cas, nous supprimons toutes les communications correspondants à ces transferts, sauf la réception de la première copie. L'ordonnancement obtenu est toujours valide, et son temps d'exécution est inférieur ou égal à celui de l'ordonnancement initial. L'ensemble des communications impliquées dans la diffusion d'un message forme alors un arbre dans le graphe de la plate-forme. Nous procédons de même pour chacun des messages diffusés.

Soit $N \in \mathbb{N}$, on considère les opérations nécessaires aux N premières diffusions de la série. Pour chaque arête (i,j), on appelle $N(P_i \to P_j, m_k)$ le nombre total de messages de type m_k envoyé sur l'arête (i,j) parmi les N premiers messages de type m_k. Les contraintes suivantes doivent être respectées par les $N(P_i \to P_j, m_k)$:

- Après la suppression des transferts inutiles de l'ordonnancement, les $N(P_i \to P_j, m_k)$ définissent un flot de valeur N depuis la source vers P_k. En particulier, les N messages à destination de P_k doivent être envoyés par la source :

$$\forall P_k \in V \sum_{(P_{\mathsf{source}},P_j)\in E} N(P_{\mathsf{source}} \to P_j, m_k) = N$$

- Ils doivent également être reçus par la cible :

$$\forall P_k \in V \sum_{(P_{\mathsf{source}},P_j)\in E} N(P_j \to P_{\mathsf{source}}, m_k) = N$$

- Il y a conservation des messages de type m_k :

$$\forall P_k \in V, \forall P_i P_i \neq P_k, P_{\mathsf{source}}, \quad \sum_{(P_j,P_i)\in E} N(P_j \to P_i, m_k) = \sum_{(P_i,P_j)\in E} N(P_i \to P_j, m_k)$$

- On s'intéresse maintenant à l'occupation des ports de communication. Sur chaque arête (i,j) doivent être transmis $N(P_j \to P_i, m_k)$ messages à destination de P_k. Nous ne savons pas si ces messages sont différents pour une autre destination $P_{k'}$, mais il y a au moins $\max_k N(P_j \to P_i, m_k)$ messages différents qui doivent être transmis en D_N unités de temps sur cette arête. En notant $N_{\max}(P_i \to P_j) = \max_k N(P_j \to P_i, m_k)$, on a

$$\forall P_i \in V, \sum_{(i,j)\in E} N_{\max}(P_i \to P_j) \cdot c_{i,j} \leqslant D_N$$

et

$$\forall P_i \in V, \sum_{(j,i)\in E} N_{\max}(P_j \to P_i) \cdot c_{j,i} \leqslant D_N$$

On pose

$$\mathsf{send}(P_i \to P_j, m_k) = \frac{N(P_i \to P_j, m_k)}{D_N} \qquad s(P_i \to P_j) = \frac{N_{\max}(P_i \to P_j)k}{D_N}$$

et

$$T_{i,j} = \frac{N_{\max}(P_i \to P_j)k}{D_N} \cdot c_{i,j}.$$

Ces variables vérifient les équations du programme linéaire 4.5, donc on a $N/D_N \leqslant \rho_{\text{opt}}$. Comme on peut faire ce raisonnement pour toute valeur de N, on montre que

$$\lim_{N\to\infty} \frac{N}{D_N} \leqslant \rho_{\text{opt}}.$$

Donc ρ_{opt} est une borne supérieure au débit d'un ordonnancement pour une série de diffusions. ■

Pour reconstruire un ensemble d'arbres de diffusion à partir d'une solution optimale (send, s, T) du programme linéaire, nous considérons le graphe de plate-forme dont les arêtes sont pondérées par les valeur de $s(P_i \to P_j)$, que l'on notera $s(i,j)$ pour simplifier les notations. On définit donc $G_s = (V, E, s)$.

Nous pouvons déjà établir la propriété suivante dans G_s.

Proposition 4.1. *Dans G_s, pour tout P_k, il existe un flot de valeur ρ_{opt} entre P_{source} et P_k.*

Démonstration. C'est une conséquence directe des contraintes (4.5a), (4.5b), et (4.5c), du programme linéaire dont (send, s, T) est solution. ■

Nous établissons maintenant le résultat qui permet de reconstruire une solution en schémas d'allocation en temps polynomial.

Théorème 4.4. *On peut extraire N arbres couvrants $A_1, \ldots, A_N$ de G_s et des pondérations $\lambda_1, \ldots \lambda_N$ tels que :*

(i) La somme pondérée des arbres est «incluse» dans G_s :

$$\forall (i,j), \quad \sum_{\substack{u=1\ldots N \\ (i,j)\in A_u}} \lambda_u \leqslant s(P_i \to P_j)$$

(ii) La somme des poids des arbres atteint le débit ρ_{opt} :

$$\sum_{u=1}^{N} \lambda_u = \rho_{\text{opt}}$$

(iii) Le nombre d'arbres est borné par

$$N \leqslant |V|^3 + |E|$$

(iv) L'ensemble pondéré d'arbres peut être trouvé en temps fortement polynomial.

Démonstration. Nous utilisons le résultat suivant, qui est la version pondérée du Théorème d'Edmonds pour les arboresences (branchings) et qui est prouvé dans [88, vol.B chapter 53] :

Théorème 4.5 (Théorème 53.9 dans [88, vol.B chapter 53]). *Soit $G = (V, E, w)$ un graphe dirigé avec des poids entiers sur les arêtes. Il existe N arbres $A_1, \dots, A_N$, avec des poids entiers $\gamma_1, \dots, \gamma_N$, tels que*

$$\forall i, j, \quad \sum_{\substack{u=1\dots N \\ (i,j)\in A_u}} \gamma_u(i,j) \leqslant w(i,j),$$

et tels que $\sum_u \gamma_u$ est maximal. De plus, ces arbres peuvent être trouvés en temps fortement polynomial et, par construction, $N \leqslant |V|^3 + |E|$.

Comme les pondérations des arêtes de G_s ne sont a priori pas entières, nous ne pouvons pas utiliser directement ce résultat. Mais comme précédemment, en notant $s(i,j) = \alpha(i,j)/\beta(i,j)$ sous forme irréductible, nous calculons le plus petit commun multiple $L = \text{ppcm}_{i,j}\{\beta(i,j)\}$ et nous appliquons le théorème précédent à $G'_s = (V, E, L \times s)$. Nous pouvons donc construire, en temps fortement polynomial, un ensemble pondéré d'arbres $(A_1, \gamma_1) \dots, (A_N, \gamma_N)$. Pour revenir au graphe G_s, nous considérons les arbres $(A_1, \lambda_1 = \gamma_1/L) \dots, (A_N, \lambda_N = \gamma_N/L)$. Par construction, ces arbres vérifient les propriétés (i) (iii) et (iv).

Il reste à prouver que $\sum \lambda_u = \rho_{\text{opt}}$. Le théorème 4.5 nous apprend que $\sum_u \lambda_u$ est maximal.

Nous considérons maintenant le multi-graphe graphe $G_s^{(m)} = (V, E^{(m)})$, tel que pour toute arête (i,j) de E, il existe une arête (i,j) dans $E^{(m)}$ de multiplicité $L \times s(i,j)$. L'ensemble d'arbres construit précédemment peut être transcrit dans ce multi-graphe : en considérant $L \times \lambda_u$ copie du graphe A_u, nous construisons un ensemble d'arbres couvrant arête-disjoints qui maximise le nombre d'arbres (égal à $L \times \sum \lambda_u$).

Dans ce multi-graphe, nous utilisons le théorème d'Edmonds sur les arboresences [99], qui montre le lien entre le nombre minimal d'arêtes dont la suppression rend un sommet inaccessible depuis la source P_{source} (on note ce nombre $\kappa(G_s^{(m)}, P_{\text{source}})$) et le nombre d'arbres couvrant arête-disjoints enracinés en P_{source} :

Théorème 4.6 (Théorème d'Edmonds sur les arboresences (branchings)). *Il existe exactement $\kappa(G_s^{(m)}, P_{\text{source}})$ d'arbres couvrants arête-disjoints enracinés en P_{source}.*

Nous avons donc ici :

$$L \times \sum_{u=1}^{N} \lambda_u = \kappa(G_s^{(m)}, P_{\text{source}}) \tag{4.6}$$

Nous avons montré dans la proposition 4.1 qu'il existe un flot de valeur ρ_{opt} dans G_s entre P_{source} et tout autre sommet P_k, donc il existe un flot de valeur $L \times \rho_{\text{opt}}$ dans $G_s^{(m)}$ entre P_{source} et P_k. D'après le théorème de Ford et Fulkerson [43], le cardinal d'une coupe minimale entre P_{source} et P_k est au moins $L \times \rho_{\text{opt}}$, donc il faut supprimer au minimum $L \times \rho_{\text{opt}}$ arêtes dans $G_s^{(m)}$ pour déconnecter P_{source} et P_k. Comme cette propriété est vraie pour tout k, nous avons $\kappa(G_s^{(m)}, P_{\text{source}}) \geqslant L \times \rho_{\text{opt}}$.

Supposons maintenant que $\kappa(G_s^{(m)}, P_{\text{source}}) = K > L \times \rho$. Alors, d'après le théorème de Ford et Fulkerson, pour tout P_k, il existe un flot de valeur K dans $G_s^{(m)}$ entre P_{source} et P_k. Appelons $x_k^{i,j}$ la valeur de ce flot sur l'arête (i,j). Alors $x_k^{i,j}/L$ décrit un flot de valeur $K/L > \rho_{\text{opt}}$ dans

le graphe G_s. On pose

$$\mathsf{send}(P_i \to P_j, m_k) = \frac{x_k^{i,j}}{L}$$

$$s(P_i \to P_j) = \max_k \{\frac{x_k^{i,j}}{L}\} \qquad\qquad T_{i,j} = c_{i,j} \cdot \max_k \{\frac{x_k^{i,j}}{L}\}$$

on construit une solution du programme linéaire 4.5, de débit $K/L > \rho_{\text{opt}}$, ce qui contredit l'optimalité de la solution initiale. Donc $\kappa(G_s^{(m)}, P_{\mathsf{source}}) = L \times \rho$. Avec l'équation 4.6, nous avons $\sum_{u=1}^{N} \lambda_u = \rho_{\text{opt}}$. ■

À l'aide des théorèmes 4.4, 3.9 et 2.1, on peut construire en temps fortement polynomial, un ordonnancement périodique pour le problème de la diffusion sous le modèle un-port bidirectionnel, s'appuyant sur la résolution d'un programme linéaire de taille $O(|V|^2) \times O(|V|^2)$. D'après le théorème 4.3, on sait que cette solution réalise le débit optimal.

Autres résultats Dans notre étude de la diffusion de données pipelinée, nous avons également obtenu deux autres résultats, que nous citons ici et qui sont présentés en détail dans [20, 23]. Le premier concerne la complexité de la diffusion en utilisant un seul arbre : nous montrons que trouver l'arbre de diffusion réalisant le meilleur débit est un problème NP-complet. On remarque donc que la relaxation en régime permanent ne permet de rendre le problème plus facile que lorsqu'on s'interdit toute contrainte artificielle à l'optimisation du débit : a priori, le débit optimal n'est pas atteint avec un seul arbre, mais avec plusieurs. Le problème général, à plusieurs arbres, peut être résolu, comme nous venons de le montrer ici. Au contraire, si l'on impose d'utiliser un seul arbre, le problème est NP-complet.

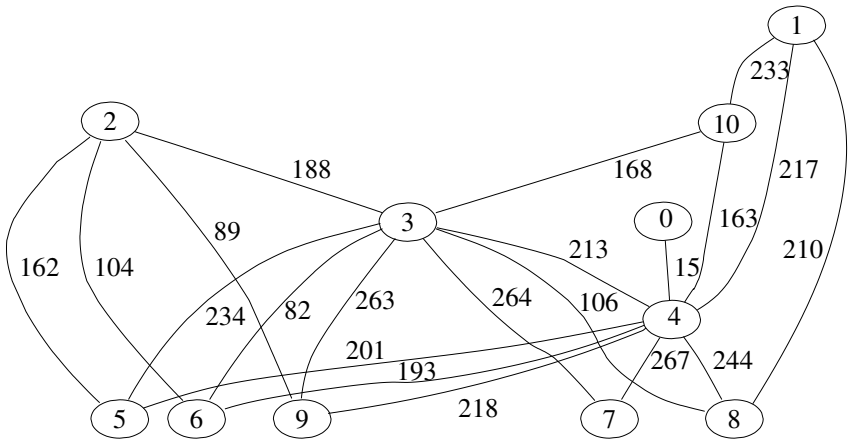
D'autre part, dans le cas particulier où la plate-forme est un graphe dirigé sans cycle, alors en utilisant le programme linéaire 4.5, on peut facilement reconstruire un ordonnancement de débit optimal, sans utiliser la décomposition en arbres complexe présentée ci-dessus.

4.2.2 Mise en œuvre sur un petit exemple

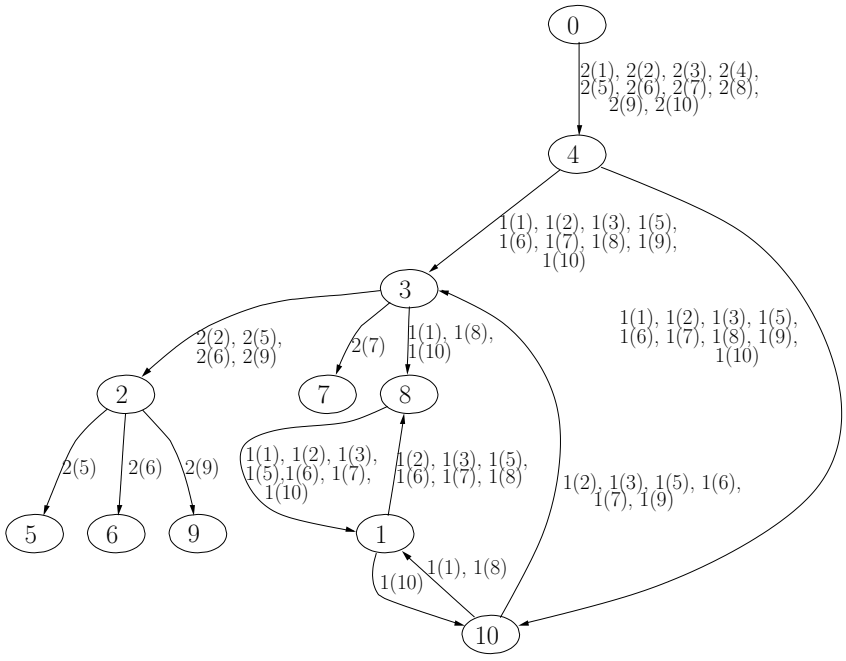
Nous présentons ici le traitement complet d'un cas pour la diffusion. Nous considérons une plate-forme élaborée par le générateur de topologie Tiers [33], dont les liens sont munis d'une bande passante aléatoire. Cette plate-forme est représentée sur la figure 4.1(a).

La figure 4.1(b) montre les résultats du programme linéaire pour la diffusion de messages depuis le processeur $P_{\mathsf{source}} = 0$. Les étiquettes des arêtes représentent les quantités de messages et leur destination : si l'arête (i, j) possède l'étiquette $y(k)$, ceci signifie que dans la solution du programme linéaire, $\mathsf{send}(P_i \to P_j, m_k) = y/T$. Toutes les quantités ont été multipliées par $T = 152$, afin d'obtenir des entiers. On voit ici que le débit obtenu est de 2 messages toutes les 152 unités de temps.

À partir de ces communications, on peut extraire deux arbres de diffusion, qui sont représentés sur la figure 4.2. Y sont représentés à la fois les arbres logiques et les communications correspondantes de la figure 4.1(b). On peut remarquer que toutes les communications associées à la solution du programme linéaire ne sont pas utilisées dans les arbres de diffusion. Par exemple, il existe un cycle de communications $P_1 \to P_8 \to P_1$, qui concerne des transferts ayant pour cible P_3, P_5, P_6 et P_7. Ces communications sont inutiles car elles sont déconnectées des



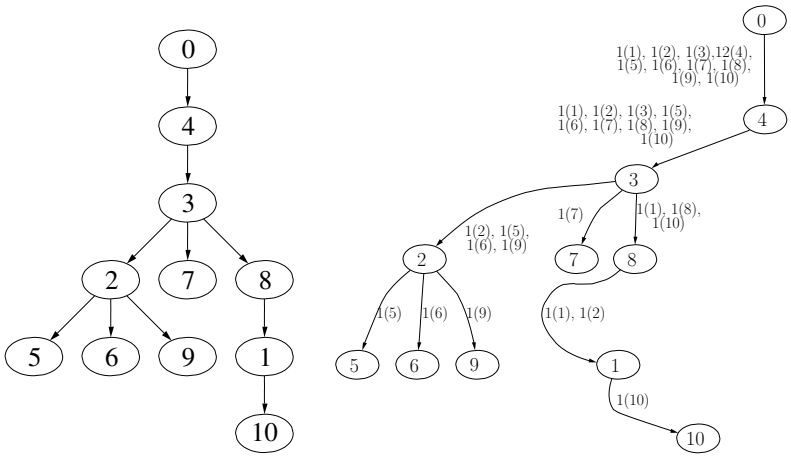
(a) Topologie. L'arête e est étiquetée par sa bande-passante $bw(e)$. Le coût de transfert d'un message est $c(e) = 1000/bw(e)$.



(b) Graphe de communication pour une solution optimale du programme linéaire. L'étiquette 2(1) sur l'arête (i, j) signifie que $\mathsf{send}(P_i \rightarrow P_j, m_1) = 2$.

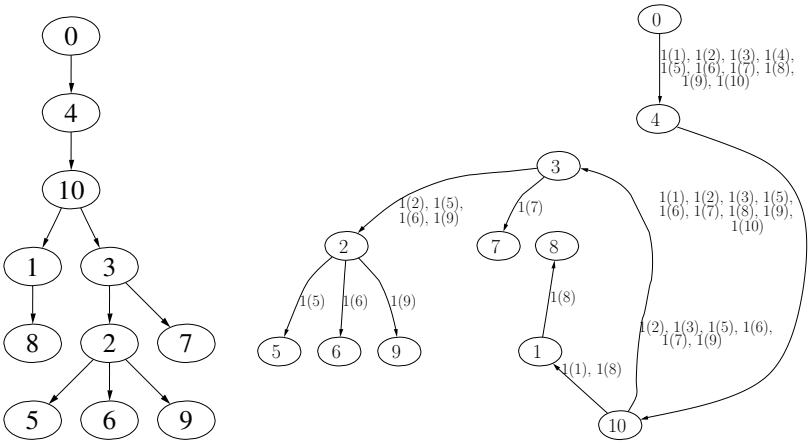
FIG. 4.1 – Exemple pour la diffusion : topologie et solution du programme linéaire.

autres communications visant ces processeurs, mais elles n'interfèrent pas avec les autres communications : le nombre maximum sur les arêtes du cycle est 1, qui correspond à des transferts «utiles» pour les processeurs P_1, P_8 et P_{10}. En décomposant la solution du programme linéaire en arbres, on néglige ces communications parasites.



arbre logique communications correspondantes

(a) Arbre de diffusion A_1



arbre logique communications correspondantes

(b) Arbre de diffusion A_2

FIG. 4.2 – Exemple pour la diffusion : arbres de diffusion.

Dans le chapitre 5, nous présenterons notre travail expérimental pour la diffusion de données sur plates-formes réelles.

4.3 Réduction

4.3.1 Programme linéaire

Nous nous intéressons maintenant au problème de la réduction pipelinée. Ce problème est plus complexe car il implique à la fois calculs et communications. Rappelons tout d'abord le

formalisme introduit pour la réduction au chapitre 2. L'ensemble des processeurs prenant part au calcul est noté $\mathcal{R}$ et chaque processeur $P_{r_i} \in Reds = \{P_{r_0}, \ldots, P_{r_N}\}$ possède initialement une valeur v_i. On cherche à calculer la valeur réduite $v = v_0 \oplus v_1 \oplus \cdots \oplus v_N$, où $\oplus$ est associatif et non commutatif. Le résultat doit au final se trouver sur le processeur P_{cible}. Pour $0 \leqslant k \leqslant m \leqslant N$, nous notons $v_{[k,m]}$ le résultat partiel issu de la réduction des valeurs $v_k, \ldots, v_m$:

$$v_{[k,m]} = v_k \oplus \cdots \oplus v_m$$

et $T_{k,l,m}$ la tâche de calcul associée à l'opération $v_{[k,l]} \oplus v_{[l+1,m]}$.

Nous avons vu que les schémas d'allocation pour la réduction sont des arbres de réduction, mêlant communications et calculs.

Comme pour les autres primitives de communications collectives, nous introduisons de nouvelles variables, caractérisant les quantités de messages circulant sur les liens : nous appelons $\mathsf{send}(P_i \to P_j, v_{[k,m]})$ le nombre de messages de types $v_{[k,m]}$ communiqués sur l'arête (i, j) en une unité de temps.

Nous devons également prendre en compte les quantités des différentes tâches de réduction effectuées par chaque processeur : on appelle $\mathsf{cons}(P_i, T_{k,l,m})$ le nombre de tâches de type $T_{k,l,m}$ effectuées par le processeur P_i en une unité de temps.

De plus, on appelle $\mathsf{size}([k, m])$ la taille d'un message $v_{[k,m]}$ et $\mathsf{flops}(k, l, m)$ la quantité de calcul nécessaire pour traiter une tâche $T_{k,l,m}$. Ainsi le processeur P_i, dont le temps de calcul d'une tâche unitaire est z_i, aura besoin de $z_i \times \mathsf{flops}(k, l, m)$ unités de calcul pour traiter une tâche $T_{k,l,m}$.

Comme précédemment, nous pouvons écrire les contraintes correspondant aux limites des ressources (liens de communication et processeurs). On note $T_{i,j}$ le temps d'occupation d'une arête (i, j) pendant une unité de temps. On a alors

$$\forall (i, j) \in E, \quad \sum_{[k,m]} c_{i,j} \times \mathsf{size}([k, m]) \times \mathsf{send}(P_i \to P_j, v_{[k,m]}) \leqslant T_{i,j}.$$

On peut également borner l'activité des processeurs. Avec les notations introduites ci-dessus, on a :

$$\forall P_i, \quad \sum_{T_{k,l,m}} z_i \times \mathsf{flops}(k, l, m) \times \mathsf{cons}(P_i, T_{k,l,m}) \leqslant 1.$$

On peut enfin écrire une sorte de «loi de conservation» des messages, bien que ce soit plus complexe que pour la distribution de données. On considère les messages de type $v_{[k,m]}$ qui se trouvent sur le processeur P_i. Si ce ne sont pas des messages de type $v_{[i,i]}$, ces messages ont été «produits» :

- soit parce qu'ils ont été reçu par P_i par un transfert $\mathsf{send}(P_j \to P_i, v_{[k,m]})$,
- soit parce qu'ils ont été créés en P_i par une tâche de type $\mathsf{cons}(P_i, T_{k,l,m})$.

De tels messages, si ce ne sont pas les messages finaux présent à la cible P_{cible}, doivent être «consommés» :

- soit par un transfert $\mathsf{send}(P_i \to P_j, v_{[k,m]})$,
- soit par une tâche de calcul $\mathsf{cons}(P_i, T_{k,m,n})$,
- soit par une tâche de calcul du type $\mathsf{cons}(P_i, T_{n,k-1,m})$.

On peut donc écrire la contrainte suivante :

$$\begin{aligned}
&\forall P_i, \forall v_{[k,m]} \text{ tels que } [k,m] \neq [i,i] \text{ ou } ([k,m], P_i) \neq ([0,N], P_{\mathsf{cible}}) \\
&\quad \sum_{P_j,(j,i)\in E} \mathsf{send}(P_j \to P_i, v_{[k,m]}) + \sum_{k \leqslant l < m} \mathsf{cons}(P_i, T_{k,l,m}) \\
&\qquad = \sum_{P_j,(i,j)\in E} \mathsf{send}(P_i \to P_j, v_{[k,m]}) + \sum_{n>m} \mathsf{cons}(P_i, T_{k,m,n}) + \sum_{n<k} \mathsf{cons}(P_i, T_{n,k-1,m})
\end{aligned}$$

De plus, le débit de messages $v_{[i,i]}$ sortant du nœud P_{r_i} doit être égal au débit total, de même que le débit de messages finaux $v_{[0,N]}$ atteignant ou créés au nœud P_{cible}.

$$\begin{aligned}
&\forall P_{r_i} \in \mathcal{R}, \sum_{(i,j)\in E} \mathsf{send}(P_{r_i} \to P_j, v_{[i,i]}) = \rho_{\text{opt}} \\
&\forall P_{r_i} \in \mathcal{R}, \sum_{(P_j,P_{\mathsf{cible}})\in E} \mathsf{send}(P_j \to P_{\mathsf{cible}}, v_{[0,N]}) + \sum_{0 \leqslant k < N} \mathsf{cons}(P_{\mathsf{cible}}, T_{0,k,N}) = \rho_{\text{opt}}
\end{aligned}$$

On impose également que pour tout nœud participant $P_{r_i} \in \mathcal{R}$, $\mathsf{send}(P_j \to P_i, v_{[i,i]}) = 0$, ainsi que $\mathsf{send}(P_{\mathsf{cible}} \to P_j, v_{[0,N]}) = 0$, pour toute arête $(P_j, P_{\mathsf{cible}})$.

Nous pouvons alors rassembler ces contraintes sous la forme d'un programme linéaire.

MAXIMISER ρ_{opt},
SOUS LES CONTRAINTES

$$\left\{\begin{array}{ll}
\text{(4.7a)} & \forall P_{r_i} \in \mathcal{R} \quad \displaystyle\sum_{(i,j)\in E} \mathsf{send}(P_{r_i} \to P_j, v_{[i,i]}) = \rho_{\text{opt}} \\
\text{(4.7b)} & \displaystyle\sum_{(P_j,P_{\mathsf{cible}})\in E} \mathsf{send}(P_j \to P_{\mathsf{cible}}, v_{[0,N]}) + \sum_{0 \leqslant k < N} \mathsf{cons}(P_{\mathsf{cible}}, T_{0,k,N}) = \rho_{\text{opt}} \\
\text{(4.7c)} & \left\{\begin{array}{l}
\forall P_i, \forall v_{[k,m]} \text{ tels que } [k,m] \neq [i,i] \text{ ou } ([k,m], P_i) \neq ([0,N], P_{\mathsf{cible}}) \\
\quad \displaystyle\sum_{P_j,(j,i)\in E} \mathsf{send}(P_j \to P_i, v_{[k,m]}) + \sum_{k \leqslant l < m} \mathsf{cons}(P_i, T_{k,l,m}) \\
\qquad = \displaystyle\sum_{P_j,(i,j)\in E} \mathsf{send}(P_i \to P_j, v_{[k,m]}) + \sum_{n>m} \mathsf{cons}(P_i, T_{k,m,n}) + \sum_{n<k} \mathsf{cons}(P_i, T_{n,k-1,m})
\end{array}\right. \\
\text{(4.7d)} & \forall (i,j) \in E, \quad \displaystyle\sum_{[k,m]} c_{i,j} \times \mathsf{size}([k,m]) \times \mathsf{send}(P_i \to P_j, v_{[k,m]}) \leqslant T_{i,j} \\
\text{(4.7e)} & \forall P_i \in V, \displaystyle\sum_{(i,j)\in E} T_{i,j} \leqslant 1 \\
\text{(4.7f)} & \forall P_i \in V, \displaystyle\sum_{(j,i)\in E} T_{j,i} \leqslant 1 \\
\text{(4.7g)} & \forall P_i, \quad \displaystyle\sum_{T_{k,l,m}} z_i \times \mathsf{flops}(k,l,m) \times \mathsf{cons}(P_i, T_{k,l,m}) \leqslant 1
\end{array}\right. \tag{4.7}$$

Comme pour les autres primitives de communications collectives, le programme linéaire précédent donne une borne supérieure sur le débit atteignable.

Théorème 4.7. *Le programme linéaire 4.7 donne une borne supérieure pour le débit d'un ordonnancement réalisant une série d'opérations de réduction.*

Démonstration. La preuve est en tous points semblable à celle de ce résultat pour la distribution de données ou pour la diffusion. ■

4.3.2 Extraction d'arbres de réduction

Une fois qu'on a obtenu une solution au programme linéaire ci-dessus, il nous faut reconstituer la structure de l'ordonnancement sous forme d'un ensemble pondéré d'arbres de réduction. Pour ceci, nous utilisons les algorithmes Recherche-Arbre et Extrait-Arbres, présentés sur la figure 4.3. Ces algorithmes sont fondés sur des principes assez simples. Partant des des différentes valeurs des $\mathsf{send}(P_i \to P_j, v_{[k,m]})$ et des $\mathsf{cons}(P_i, T_{k,l,m})$, on commence par rechercher un arbre de réduction avec l'algorithme Recherche-Arbre, puis on calcule le débit maximal qu'il peut avoir et on l'extrait de la solution. On continue tant que la somme des débits des arbres trouvés est inférieure à ρ_{opt}.

Pour trouver un arbre avec Recherche-Arbre, on maintient la liste des «entrées» qui doivent être construites, sous forme de messages localisés. Au début, la seule entrée est le message final, localisé sur le processeur cible : $(v_{[O,N]}, P_{\mathsf{cible}})$. Pour chaque entrée qui doit être construite, on cherche une tâche de calcul ou un transfert qui produit la donnée correspondante. On rajoute cette tâche ou ce transfert dans l'arbre courant, puis on met à jour les entrées à construire : celle que l'on vient de traiter est supprimée, et une ou deux autres entrées (dans le cas d'une tâche de calcul) peuvent être rajoutées. Lorsqu'il n'y a plus d'entrées à construire (toutes les entrées sont des messages $v_{[i,i]}, P_{r_i}$), on a trouvé un arbre.

On peut ainsi construire, à partir de la solution du programme linéaire, un ensemble pondéré d'arbres réalisant le débit ρ_{opt}.

Théorème 4.8. *L'algorithme Extrait-Arbres(send, cons) où (send, cons) est une solution optimale du programme linéaire, construit un ensemble d'arbres tel que :*

- $\forall (T_{k,l,m}, P_i) \quad \sum_{\substack{A \in \mathit{Arbres} \\ (T_{k,l,m}, P_i) \in A}} x(A) \leqslant \mathit{cons}(P_i, T_{k,l,m})$,
- $\forall (v_{[k,m]}, (i,j)) \quad \sum_{\substack{A \in \mathit{Arbres} \\ (v_{[k,m]}, (i,j)) \in A}} x(A) \leqslant \mathit{send}(P_i \to P_j, v_{[k,m]})$,
- $|\mathit{Arbres}|$ *est polynomial en la taille de la plate-forme.*
- *l'algorithme s'exécute en temps polynomial en la taille de la plate-forme.*

Démonstration. On note (send^0,cons^0) la solution du programme linéaire avant l'exécution de l'algorithme et (send, cons) les variables qui sont modifiées par l'algorithme. Nous allons montrer

RECHERCHE-ARBRE(send, cons)

1: $E := \{(v_{[0,N]}, P_{\text{cible}})\}$
2: Arbre $:= \emptyset$
3: **Tant que** il existe $e \in E$ avec $e \neq (v_{[i,i]}, P_{r_i})$ **:**
4: trouver un tel e, que l'on note $(v_{[k,m]}, P_i)$
5: **Si** il existe l tel que $\mathsf{cons}(P_i, T_{k,l,m}) > 0$ **Alors**
$\{$*le résultat* $v_{[k,m]}$ *est calculé sur place*$\}$
6: $E \leftarrow E - \{e\}$
7: $E \leftarrow E \cup \{(v_{[k,l]}, P_i), (v_{[l+1,m]}, P_i)\}$
8: Arbre $\leftarrow$ Arbre $\cup \{(T_{k,l,m}, P_i)\}$
9: **Aller en** 3
10: **Sinon Si** il existe un processeur P_j tel que $\mathcal{A}(\mathsf{send}(P_i \to P_j, v_{[k,m]}) > 0$ **Alors**
$\{$*le résultat* $v_{[k,m]}$ *est reçu du processeur* $P_j\}$
11: $E \leftarrow E - \{e\}$
12: $E \leftarrow E \cup \{(v_{[k,m]}, P_j)\}$
13: Arbre $\leftarrow$ Arbre $\cup \{(v_{[k,m]}, (j,i))\}$
14: **Aller en** 3
Renvoyer Arbre

EXTRAIT-ARBRES(send, cons)

1: Arbres $\leftarrow \emptyset$
2: **Tant que** $\sum_{T \in \text{Arbres}} x(T) < \rho_{\text{opt}}$ **:**
3: $A \leftarrow$ RECHERCHE-ARBRE(send, cons)
4: $x(A) \leftarrow \min\Big\{ \min_{(T_{k,l,m}, P_i) \in A} \mathsf{cons}(P_i, T_{k,l,m}), \min_{(v_{[k,m]}, (i,j)) \in A} \mathsf{send}(P_i \to P_j, v_{[k,m]}) \Big\}$
5: **Pour tout** $(T_{k,l,m}, P_i) \in A$ **:**
6: $\mathsf{cons}(P_i, T_{k,l,m}) \leftarrow \mathsf{cons}(P_i, T_{k,l,m}) - x(A)$
7: **Pour tout** $(v_{[k,m]}, (i,j)) \in A$ **:**
8: $\mathsf{send}(P_i \to P_j, v_{[k,m]}) \leftarrow \mathsf{send}(P_j \to P_i, v_{[k,m]}) - x(A)$
9: Arbres $\leftarrow$ Arbres $\cup \{(A, x(A))\}$
Renvoyer Arbres

FIG. 4.3 – Recherche et extraction d'arbres d'une solution du programme linéaire.

que l'algorithme conserve l'invariant suivant :

$$\begin{cases} \forall (T_{k,l,m}, P_i) \quad \mathsf{cons}(P_i, T_{k,l,m}) + \displaystyle\sum_{\substack{A \in \mathsf{Arbres} \\ (T_{k,l,m}, P_i) \in A}} x(A) = \mathsf{cons}^0(P_i, T_{k,l,m}) \\ \forall (v_{[k,m]}, (i,j)) \quad \mathsf{send}(P_i \to P_j, v_{[k,m]}) \displaystyle\sum_{\substack{A \in \mathsf{Arbres} \\ (v_{[k,m]}, (i,j)) \in A}} x(A) = \mathsf{send}^0(P_i \to P_j, v_{[k,m]}) \\ (\mathsf{send}, \mathsf{cons}) \text{ est une solution valide de débit } \rho_{\text{opt}} - \displaystyle\sum_{A \in \mathsf{Arbres}} x(A) \\ \forall A \in \mathsf{Arbres}, \quad A \text{ est un arbre de réduction valide} \end{cases}$$

Au début de l'exécution, nous avons (send^0,cons^0) = (send, cons), et $\mathsf{Arbres} = \emptyset$, donc l'invariant est vérifié.

À chaque étape de la boucle «tant que» dans Extrait-Arbres, l'algorithme Recherche-Arbre construit un ensemble de tâches (tâches de calcul ou transfert) tel que chaque entrée d'une tâche est soit la sortie d'une autre tâche, soit un message $(v_{[i,i]}, P_{r_i})$ et le résultat de ces tâches est $(v_{[0,N]}, P_{\mathsf{cible}})$. Donc l'ensemble de tâches construit A est un arbre de réduction valide. De plus, toutes les tâches dans A ont un débit non nul dans (send, cons) et supérieur à $x(A)$ par construction. Comme A est un arbre de réduction, la loi de conservation est vérifiée pour les tâches qui le composent munies de la pondération $x(A)$: en notant

$$\mathsf{cons}^A(P_i, T_{k,l,m}) = \begin{cases} x(A) & \text{si } (T_{k,l,m}, P_i) \in A \\ 0 & \text{sinon} \end{cases}$$
$$\mathsf{send}^A(P_i \to P_j, v_{[k,m]}) = \begin{cases} x(A) & \text{si } (v_{[k,m]}, (i,j)) \in A \\ 0 & \text{sinon} \end{cases}$$

On a :

$$\begin{aligned} &\forall P_i, \forall v_{[k,m]} \text{ tels que } [k,m] \neq [i,i] \text{ ou } ([k,m], P_i) \neq ([0,N], P_{\mathsf{cible}}) \\ &\quad \sum_{P_j, (j,i) \in E} \mathsf{send}^A(P_j \to P_i, v_{[k,m]}) + \sum_{k \leqslant l < m} \mathsf{cons}^A(P_i, T_{k,l,m}) \\ &\qquad = \sum_{P_j, (i,j) \in E} \mathsf{send}^A(P_i \to P_j, v_{[k,m]}) + \sum_{n > m} \mathsf{cons}^A(P_i, T_{k,m,n}) + \sum_{n < k} \mathsf{cons}^A(P_i, T_{n,k-1,m}) \end{aligned}$$

Comme (send, cons) est une solution valide, elle vérifie la loi de conservation, donc (send$-$send^A, cons$-$cons^A) vérifie la loi de conservation et est donc une solution valide. Or la modification de (send, cons) dans l'algorithme Extrait-Arbres (lignes 5 à 8), revient à calculer (send$-$send^A, cons$-$cons^A). Donc l'invariant est bien vérifié après une étape de la boucle «tant que».

Nous nous intéressons maintenant au nombre d'arbres créés. Une fois qu'un arbre A a été construit, on calcule sa pondération $x(A)$ comme le minimum des débits des tâches qui le compose. Donc il existe au moins une de ces tâches dont la nouvelle valeur dans (send, cons) est mise à zéro. Le nombre total de tâches est borné par :

– $N^3 \times |V|$ pour les tâches de calcul,
– $N^2 \times |E|$ pour les transferts.

Comme $N \leqslant |V|$ et $|E| \leqslant |V|^2$, on extrait au plus $|V|^4$ arbres de réduction. Enfin, comme une nouvelle tâche est ajoutée dans un arbre à chaque étape de Recherche-Arbre, l'algorithme s'exécute en temps polynomial. ■

Une fois construit un ensemble de schémas d'allocation pour la réduction, réalisant le débit du programme linéaire à l'aide du théorème 4.8, on peut reconstruire en temps fortement polynomial, grâce aux théorèmes 3.9 et 2.1, un ordonnancement périodique pour le problème de la réduction sous le modèle un-port bidirectionnel. D'après le théorème 4.7, on sait que cette solution réalise le débit optimal.

4.3.3 Expérimentations par simulation

4.3.3.1 Petit exemple réaliste

Comme pour le cas de la diffusion, nous présentons les résultats obtenus sur une topologie générée par Tiers [33], où les capacités de calcul des processeurs et les bandes-passantes des liens ont été choisis aléatoirement. La plate-forme est représentée sur la figure 4.4. Nous supposons ici que tout le messages $v_{[k,m]}$ ont la même taille (10 MB) et que toutes les tâches de réduction partielle nécessitent un calcul de 10 Mflops. Les processeurs pouvant prendre part aux calculs sont les nœuds des réseaux locaux (LAN) générés par Tiers, qui sont grisés sur la figure 4.4. Les autres nœuds sont des routeurs, qui ne possèdent pas de données et ne participent pas aux calculs.

La figure 4.5 présente une solution du programme linéaire pour la réduction, calculée par lp-solve et projetée sur la topologie. L'étiquette $[k, m] : x$ sur l'arête (i, j) signifie que dans la solution du programme linéaire, x messages $v_{[k,m]}$ sont transis sur l'arête (i, j) en une unité de temps. De même cons$[k, l, m] : x$ sur le processeur P_i signifie que ce processeur effectue x tâches $T_{k,l,m}$ en une unité de temps. Le débit obtenu est $\rho_{\text{opt}} = 2/9$. On peut extraire deux arbres de réduction de la solution, qui sont représentés sur les figures 4.6(a) et 4.6(b).

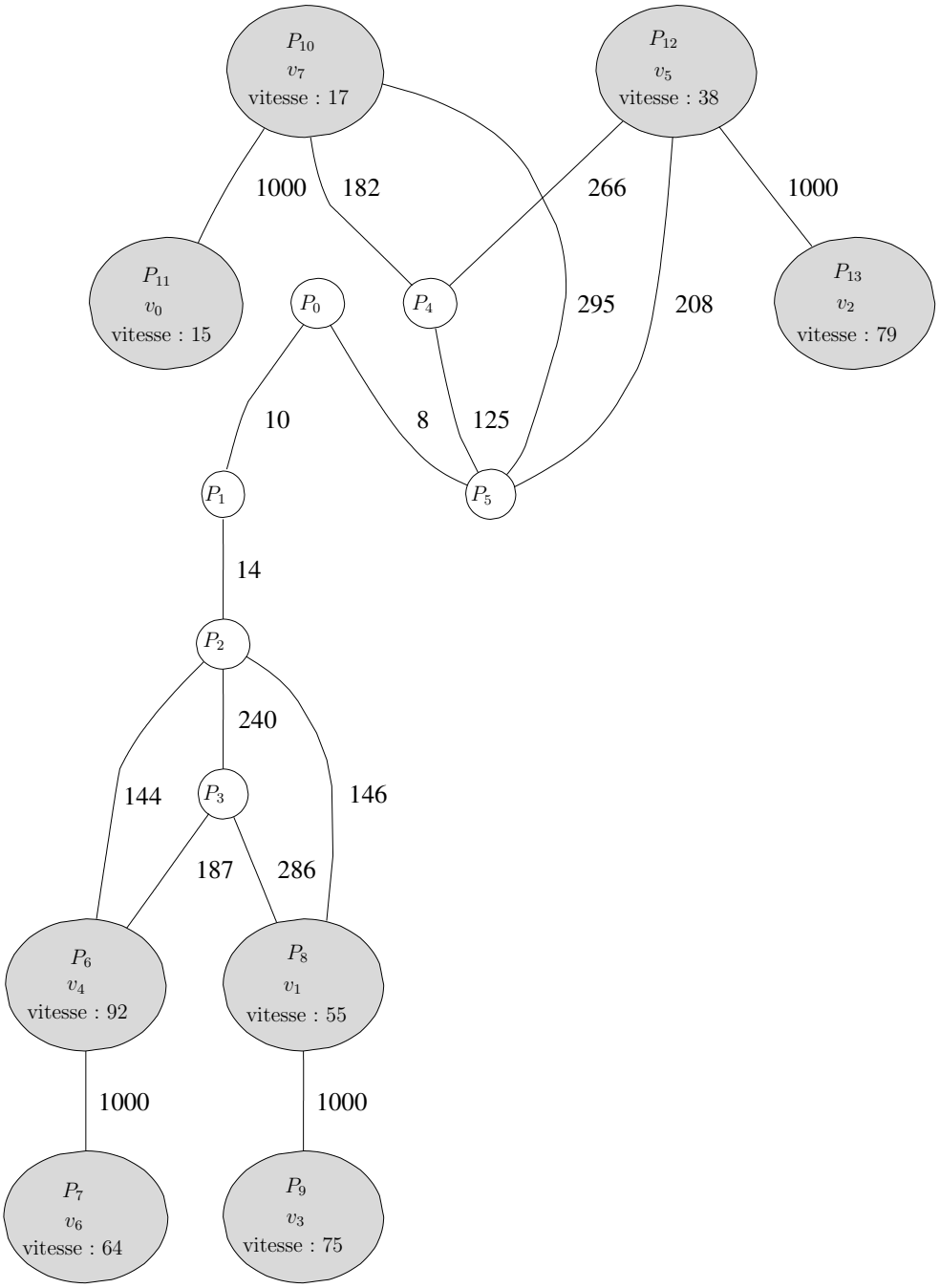


FIG. 4.4 – Une topologie hétérogène, générée par Tiers. Seuls les nœuds grisés sont des processeurs possédant des données et participant aux calculs. Les nœuds en blanc sont des routeurs. Le processeur cible est le nœud 6, possédant la valeur v_4 ($P_6 = P_{r_4}$). Les liens sont étiquetés par leur bande-passante (en MB/s) et les processeurs par leur vitesse de calcul (en Mflop/s)

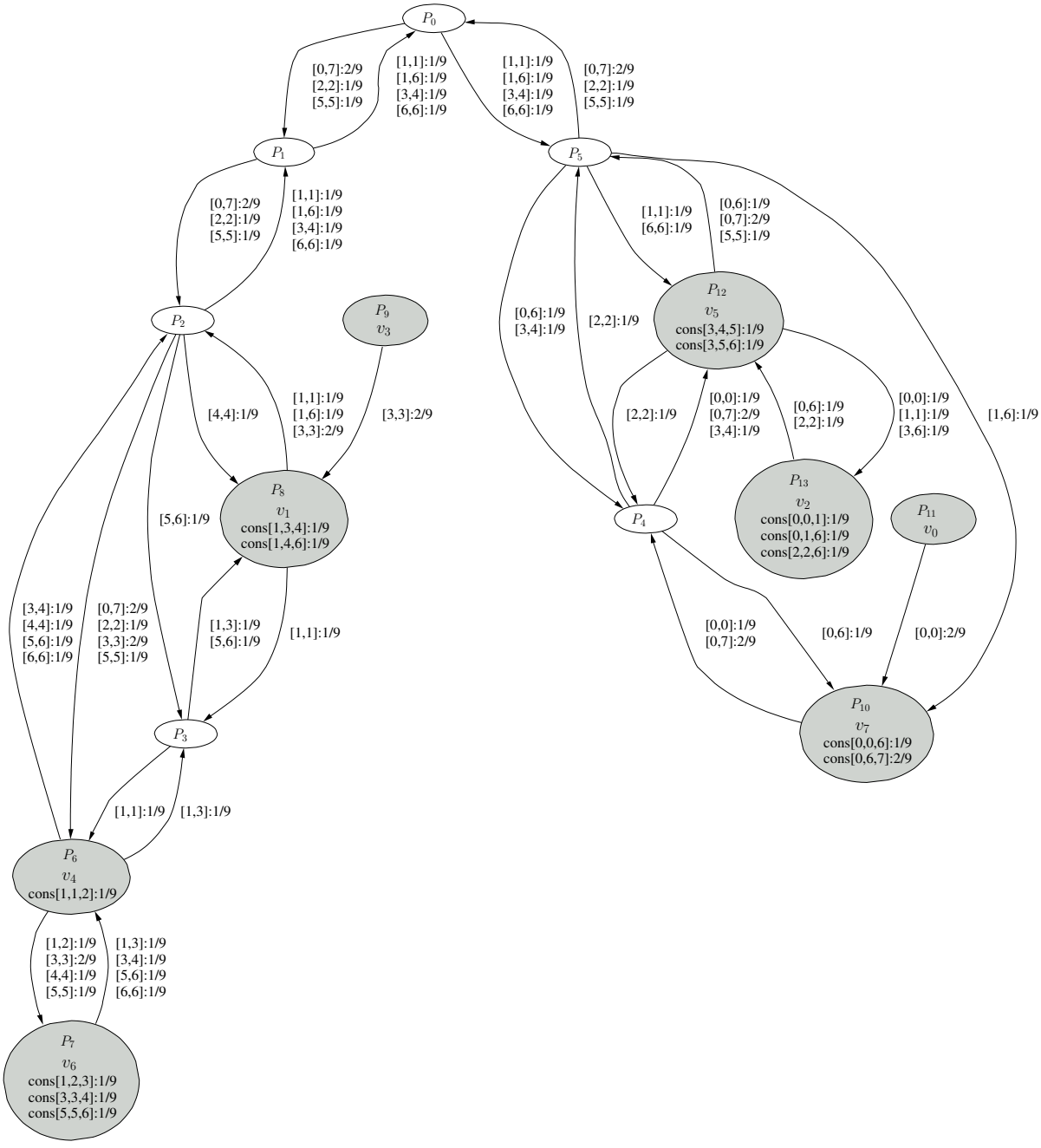
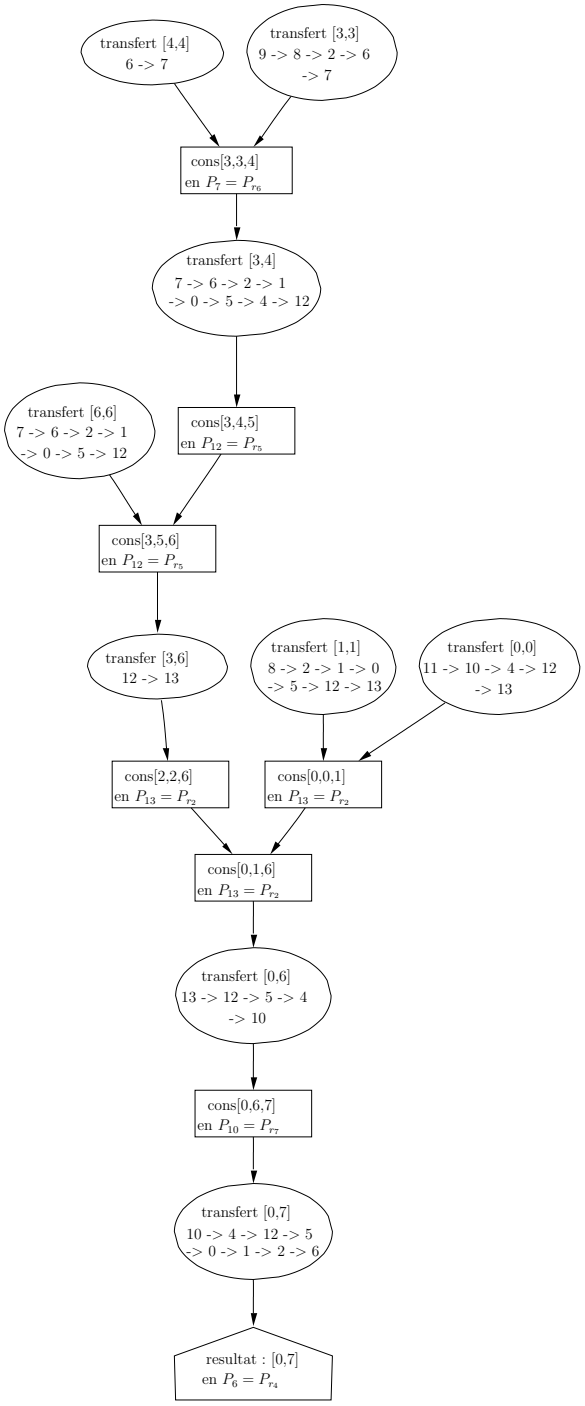
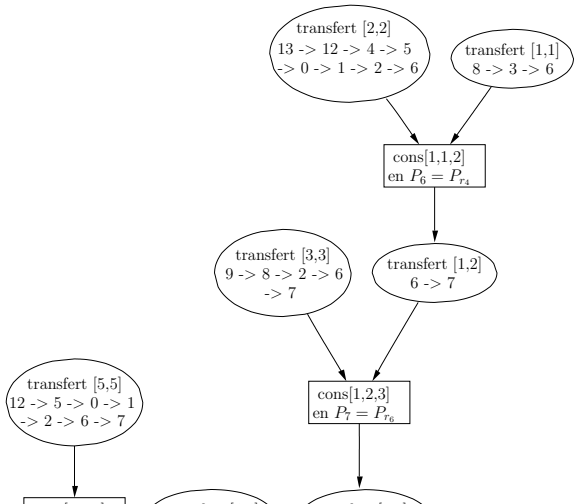


FIG. 4.5 – Une solution du programme linéaire, représentée sur le graphe de topologie.



(a) Arbre de réduction A_1

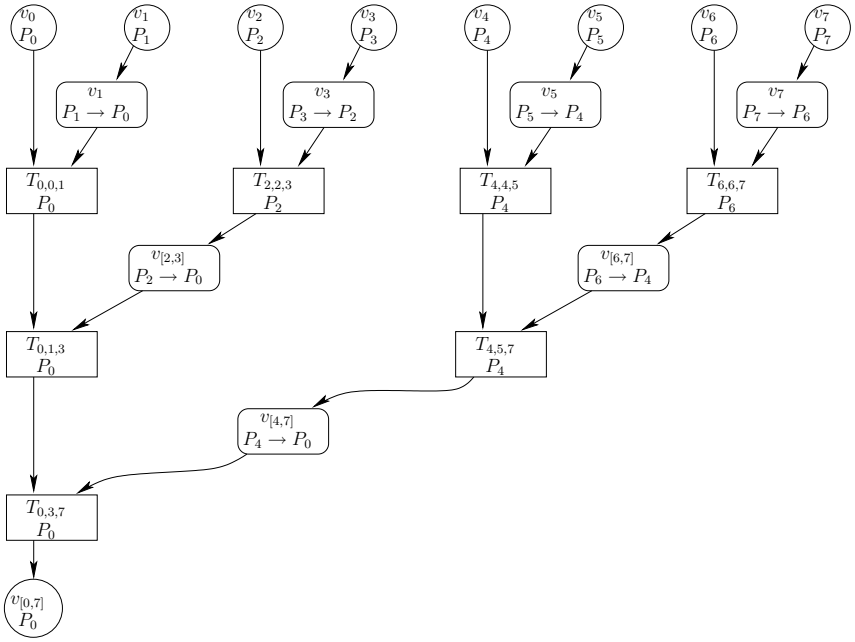


4.3.3.2 Comparaison avec une stratégie utilisant un seul arbre

Dans cette partie, nous voulons évaluer l'intérêt de notre approche pour le problème de la réduction pipelinée. Comme base de notre comparaison, nous choisissons d'imiter la stratégie choisie dans l'implémentation MPICH du standard MPI. Ce standard définit la primitive de communications collectives `MPI_Reduce` qui réalise la réduction d'un ensemble de données. Bien que le standard MPI permette de spécifier si l'opérateur de réduction est commutatif ou non, MPICH n'utilise pas cette fonctionnalité (pour éviter d'avoir des différences de résultats dues à des erreurs d'arrondi, ou à des erreurs de débordement), ce qui le rend comparable à notre opérateur de réduction. Dans l'implémentation MPICH[1], cette réduction est faite selon l'algorithme suivant, exécuté en chaque processeur P_i participant aux calculs :

on suppose que la cible de la réduction a l'indice logique 0
$m \leftarrow$ indice logique du processeur pour la réduction ($P_i = P_{r_m}$)
$v \leftarrow$ valeur du processeur ($v_m = v$)
on considère l'écriture binaire de $m = m_b \dots m_1 m_0$
Tant que $b > 0$ **:**
  Si $m_0 = 1$ **Alors**
    envoyer mon résultat à $m_b \dots m_1 0$
    terminer
  Sinon
    recevoir v' du processeur $m_b \dots m_1 1$
    $v \leftarrow v \oplus v'$
    on décale vers la droite les bits de m : $m \leftarrow m_b \dots m_1$

Pour une plate-forme à 8 nœuds, tels que $P_{r_i} = P_i$, cet algorithme produit l'arbre de réduction suivant :



Une telle construction est justifiée lorsque tous les processeurs sont équivalents et le réseau de communication rapide. Ceci est vrai sur une grappe de processeurs homogènes, qui est le

[1]Nous nous basons sur la version 1.2.6 de MPICH, la dernière disponible au moment de ces tests.

type de plates-formes pour lequel MPICH a été implémenté. Dans les expériences suivantes, nous comparons le débit qui peut être obtenu en utilisant l'arbre de MPICH, par rapport au débit obtenu par notre ensemble d'arbres. Notons que nous utilisons l'arbre de MPICH pour effectuer une succession pipelinée de réductions, alors que MPICH n'implémente pas une telle fonctionnalité.

En utilisant le générateur de topologie Tiers, nous générons plus de 200 plates-formes de 30 nœuds. Parmi les nœud des réseaux locaux (LAN), nous en choisissons certains qui participeront à la réduction. Pour chaque problème, consistant en un graphe de plate-forme et un ensemble de processeurs participants, nous calculons le débit optimal en résolvant le programme linéaire précédent, puis nous décomposons la solution obtenue en un ensemble pondéré d'arbres de réduction.

Nous avons implémenté un algorithme distribué qui permet d'effectuer la réduction en suivant un arbre donné par des règles locales, ou même selon un ensemble pondéré d'arbres. Nous traduisons l'ensemble d'arbres obtenus précédemment sous la forme de règles locales et nous faisons de même pour l'arbre de MPICH. Puis nous comparons le débit obtenu par ces deux méthodes (notre approche «multi-arbre» et l'arbre de MPICH) par simulation.

L'intérêt d'utiliser la simulation dans ce cas, outre le temps considérable gagné en mise au point et en expérimentations, est la capacité qu'offre un simulateur de comparer deux algorithmes sur une grande diversité de plates-formes et dans des conditions strictement similaires, ce que ne permettent pas les expériences en réel, difficiles à mettre en places à cause de l'instabilité naturelles des plates-formes distribuées. À cause des nombreux utilisateurs présents sur une grappe de calcul et des nombreuses connexions utilisant les liens longue distance, il est quasiment impossible de garantir qu'une plate-forme non dédiée à l'expérience aura des caractéristiques identiques pour plusieurs tests. C'est pourquoi on préfère souvent utiliser la simulation pour permettre la reproductibilité de nos tests. De plus, comme la simulation d'un algorithme est plus rapide que son exécution réelle, on peut tester nos algorithmes sur un grand nombre de plates-formes. Nous utilisons le simulateur SimGrid [36, 73], qui permet de simuler l'exécution d'algorithmes distribués sur des environnements réalistes.

Les résultats des simulations sont présentés à la figure 4.7, en fonction de la proportion de nœuds participants parmi tous les nœuds des réseaux locaux disponibles, appelée «densité». Ces résultats montrent que notre stratégie est plus efficace que celle utilisant l'arbre de MPICH, réalisant un débit deux à trois fois meilleur, en particulier quand la densité de nœuds participants est importante. La première raison est que nous prenons en compte des informations topologiques, qui sont ignorées lors de la construction de l'arbre de MPICH : sur une plate-forme hétérogène, nous savons quels processeurs sont les plus adaptés aux calculs et quels liens ont une grande bande-passante. Ces informations nous permettent de construire un ensemble d'arbres de réduction efficaces, alors que l'arbre de MPICH est construit en utilisant uniquement les indices logiques de nœuds, ce qui ne prend même pas en compte la proximité physique des processeurs.

Comme l'arbre de MPICH n'est pas adapté aux topologies hétérogènes, nous comparons également les deux stratégies dans le cas supposé le plus favorable à l'arbre de MPICH. Nous considérons une topologie homogène complètement connectée. Pour ces simulations, nous utilisons une clique de 10 processeurs de même puissance de calcul (100 Mflops/s), chaque paire de processeurs étant reliée par un lien de même bande-passante (100 MB/s). Comme dans les simulations précédentes, les messages ont tous la même taille, de même que les tâches de calcul. Dans ce contexte, l'arbre de MPICH est censé être aussi efficace que n'importe quel arbre de

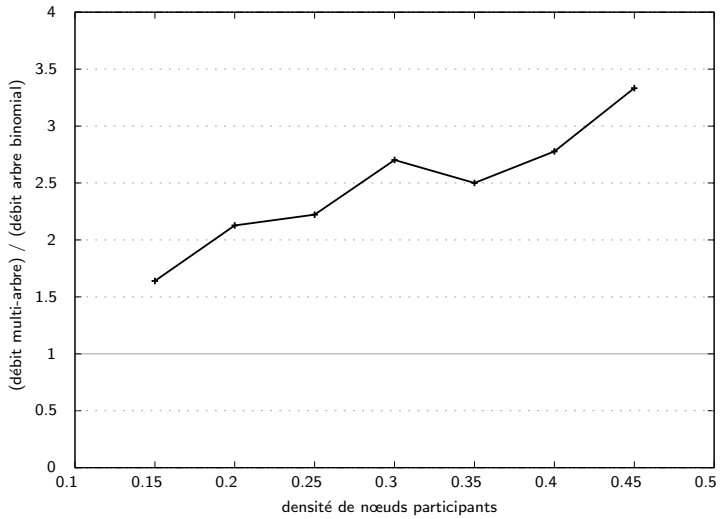


FIG. 4.7 – Comparaison du débit de l'approche multi-arbre par rapport à l'utilisation d'un seul arbre de MPICH, sur un ensemble de plates-formes générées par Tiers.

notre solution. Nous menons plusieurs simulations en faisant varier l'importance relative des calculs et de communications : pour une même taille de messages (100 MB), la taille des tâches de calcul varie de 0.1 Mflops à 10^6 Mflops. Les résultats de ces tests sont présentés à la figure 4.8.

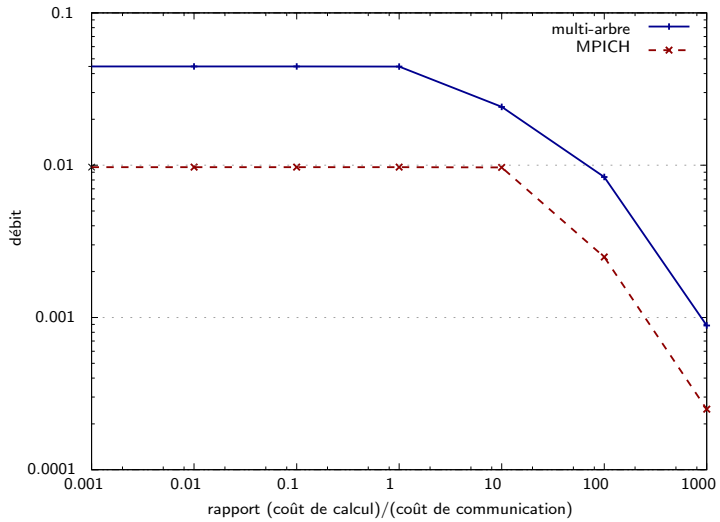


FIG. 4.8 – Débits obtenus par les deux approches sur une plate-forme en clique. L'échelle est logarithmique sur les deux axes.

On peut remarquer sur cette figure que lorsque le coût des calculs augmente, les deux approches voient leur débit régresser, cependant notre algorithme reste meilleur que l'arbre de MPICH, à un facteur à peu près constant (environ 4 fois meilleur). Ce résultat s'explique par la capacité de notre approche multi-arbre de distribuer le calcul sur tous les processeurs disponibles ; ceci peut entraîner de nombreuses communications, cependant pour un rapport (coût de calcul)/(coût de communication) élevé, le surcoût en communication est compensé par

le gain en temps de calcul.

Ces simulations soulignent les deux avantages principaux de notre approche : (i) nous prenons en compte des informations topologiques, ainsi nous sommes capables d'affecter les calculs aux processeurs les plus adaptés et (ii) en utilisant plusieurs arbres, nous sommes capables de distribuer le calcul d'une seule tâche à plusieurs processeurs. Tout ceci permet d'obtenir un meilleur débit de diffusion.

4.4 Diffusion restreinte

4.4.1 Position du problème

Nous nous intéressons ici au problème de la diffusion partielle d'une série messages d'un processeur P_{source} à un sous-ensemble V_{cibles} de processeurs de la plate-forme. Nous avons vu dans le chapitre précédent que nous n'étions pas capable de trouver un ordonnancement optimal pour le débit en utilisant la méthode de l'ellipsoïde : nous étions amenés à rechercher un arbre dirigé couvrant V_{cibles}, de poids minimal, ce qui est un problème difficile (problème de Steiner [65]).

Même si ce problème est très semblable à celui de la diffusion pour lequel nous avons pu trouver une méthode de résolution efficace pour le cas du modèle un-port bidirectionnel (voir partie 4.2), là encore nous allons voir que la diffusion restreinte est plus difficile, puisque nous montrerons que trouver le meilleur débit est un problème NP-complet.

Avant de présenter ce résultat, nous examinons sur un petit exemple pourquoi l'approche développé pour la diffusion est inapplicable ici. Nous pouvons en effet écrire un programme linéaire très semblable à ce que nous avons fait pour la diffusion. Comme pour la diffusion, nous considérons le nombre de message communiqués sur l'arête (i, j) pendant une unité de temps, à destination de $P_k \in V_{\mathsf{cibles}}$, quantité qu'on note $\mathsf{send}(P_i \to P_j, m_k)$. La seule différence est que cette quantité n'existe que pour $P_k \in V_{\mathsf{cibles}}$ et non pour tout nœud de la plate-forme. Les $\mathsf{send}(P_i \to P_j, m_k)$ définissent un flot entre P_{source} et P_k. Comme précédemment, nous posons

$$\forall (i,j) \in E, s(P_i \to P_j) = \max_{P_k \in V_{\mathsf{cibles}}} \{\mathsf{send}(P_i \to P_j, m_k)\}.$$

et nous supposons que tous les messages communiqués sur l'arête (i, j) sont un sous-ensemble des messages à destination de P_{k_0}, tel que $\mathsf{send}(P_i \to P_j, m_{k_0}) = \max_k \{\mathsf{send}(P_i \to P_j, m_k)\}$. Nous obtenons donc le programme linéaire suivant, qui, comme pour la diffusion, définit une

borne supérieure sur le débit d'une diffusion restreinte :

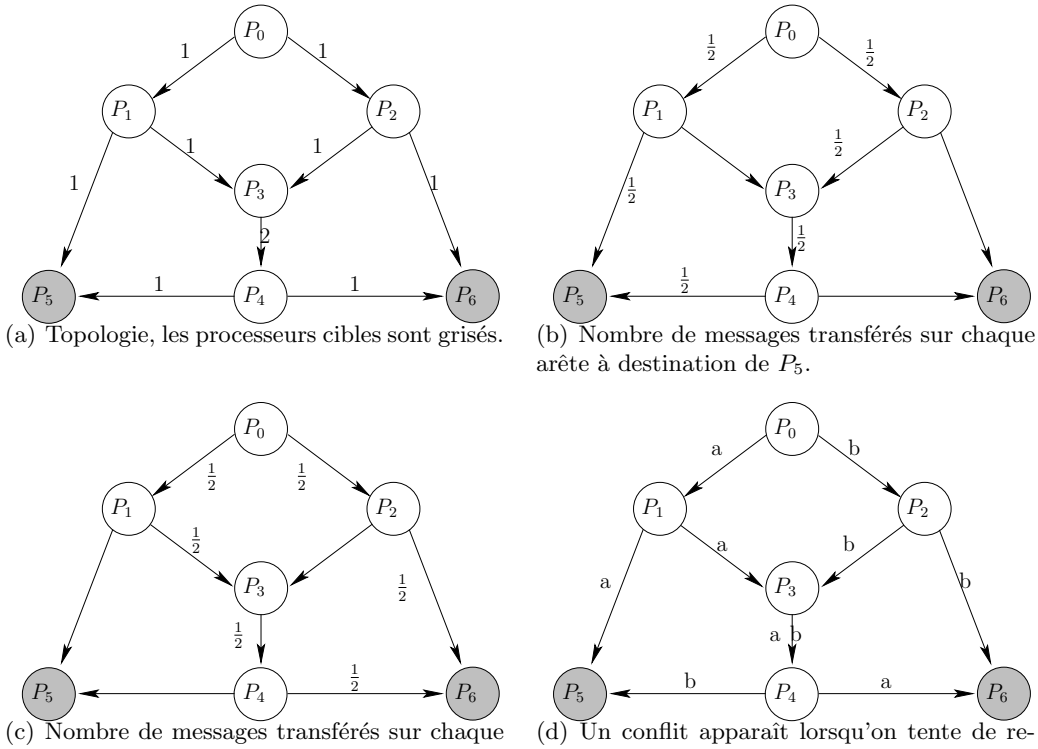
$$
\begin{array}{l}
\text{MAXIMISER } \rho_{\text{opt}}, \\
\text{SOUS LES CONTRAINTES} \\
\left\{
\begin{array}{lll}
\text{(4.8a)} & \forall P_k \in V_{\text{cibles}} & \displaystyle\sum_{(P_{\text{source}},P_j)\in E} \text{send}(P_{\text{source}} \to P_j, m_k) = \rho_{\text{opt}} \\
\text{(4.8b)} & \forall P_k \in V_{\text{cibles}} & \displaystyle\sum_{(P_{\text{source}},P_j)\in E} \text{send}(P_j \to P_{\text{source}}, m_k) = \rho_{\text{opt}} \\
\text{(4.8c)} & \left\{\begin{array}{l}\forall P_k \in V_{\text{cibles}}, \forall P_i \\ P_i \neq P_k, P_{\text{source}}\end{array}\right. & \displaystyle\sum_{(P_j,P_i)\in E} \text{send}(P_j \to P_i, m_k) = \sum_{(P_i,P_j)\in E} \text{send}(P_i \to P_j, m_k) \\
\text{(4.8d)} & \forall (i,j) \in E, & s(P_i \to P_j) = \displaystyle\max_{P_k \in V_{\text{cibles}}} \{\text{send}(P_i \to P_j, m_k)\} \\
\text{(4.8e)} & \forall (i,j) \in E, & s(P_i \to P_j) \cdot c_{i,j} \leqslant T_{i,j} \\
\text{(4.8f)} & \forall P_i \in V, & \displaystyle\sum_{(i,j)\in E} T_{i,j} \leqslant 1 \\
\text{(4.8g)} & \forall P_i \in V, & \displaystyle\sum_{(j,i)\in E} T_{j,i} \leqslant 1
\end{array}
\right.
\end{array}
\tag{4.8}
$$

Théorème 4.9. *Le programme linéaire 4.8 donne une borne supérieure pour le débit d'un ordonnancement réalisant une série d'opérations de diffusion restreinte depuis P_{source} vers V_{cibles}.*

La preuve de ce résultat est très semblable à celle fournie pour la diffusion de données.

Considérons la plate-forme illustrée sur la figure 4.9(a), où les arêtes sont étiquetées par le coût de transmission d'un message. Il y a deux destinations à la diffusion : $V_{\text{cibles}} = \{P_5, P_6\}$. Une solution optimale du programme linéaire précédent est illustrée dans les figures suivantes : la figure 4.9(b) représente les quantités de messages à destination du processeur P_5, donc la valeur de $\text{send}(P_i \to P_j, m_5)$ sur chaque arête (i,j) et la figure 4.9(c) représente les quantités de messages à destination de P_6. D'après ces figures, on peut atteindre un débit d'un message par unité de temps. Sur l'arête (P_3, P_4), au maximum 1/2 message est transmis, et comme $c_{3,4} = 2$, le port de communication en sortie de P_3 (comme le port en entrée de P_4) est utilisé à plein temps.

On tente maintenant de reconstruire une solution fondée sur ces quantités. Pour obtenir un débit de 1 message par unité de temps en direction de P_5, on voit sur la figure 4.9(b) que le message doit être découpé en deux parties, l'une passant par la route $P_0 \to P_1 \to P_5$, l'autre par la route $P_0 \to P_2 \to P_3 \to P_4 \to P_5$. On peut faire le même raisonnement pour la destination P_6. On appelle a la partie du message que la source envoie à P_1 et b la partie émise vers P_2. En utilisant les routes précédentes, il faudrait utiliser la structure illustrée par la figure 4.9(d) pour reconstruire une solution, ce qui implique que le lien (P_3, P_4) doit transporter les deux parties du messages. Ceci n'est pas possible en temps 1 à cause du coût de communication sur ce lien. On voit sur cet exemple qu'il n'est pas toujours possible de reconstruire un ensemble d'arbres de réduction convenables à partir d'une solution du programme linéaire. La différence avec le cas de la diffusion vient du fait qu'il n'existe pas d'équivalent du théorème de décomposition en arbres 4.5 pour des arbres couvrants uniquement un sous-ensemble de sommets dans le graphe.



(a) Topologie, les processeurs cibles sont grisés.

(b) Nombre de messages transférés sur chaque arête à destination de P_5.

(c) Nombre de messages transférés sur chaque arête à destination de P_6.

(d) Un conflit apparaît lorsqu'on tente de reconstruire la solution.

FIG. 4.9 – Problème pour la diffusion restreinte.

Remarque Un lecteur attentif et rompu aux techniques de «Network Coding» pourrait faire remarquer qu'une simple combinaison permet d'obtenir le débit optimal sur l'exemple précédent : si le processeur P_3, recevant les deux parties du messages a et b, calcule et envoie à P_4 la valeur (a XOR b), le processeur P_4 peut transmettre cette valeur à P_5 et P_6. Chacune de ces destinations, possédant alors soit a et (a XOR b), soit b et (a XOR b), peut reconstituer l'intégralité du message. On obtient bien le débit d'un message diffusé toutes les unités de temps, comme calculé par le programme linéaire.

Ahlswede [2], puis Koetter [61] ont montré que ce résultat était général : dans tout graphe, il existe un codage qui permet d'atteindre le flot maximal pour tout couple (source, destination). Cependant, ces considérations sortent du domaine de cette thèse : nous nous restreignons, pour l'étude de la diffusion restreinte, à ne pas utiliser de codage. En effet, les codages proposés par ces études et les algorithmes qui calculent ces codages pour un graphe de communication général sont d'une grande complexité. On pourrait aussi utiliser des combinaisons aléatoires, comme proposé dans le protocole de diffusion de fichiers en pair-à-pair Avalanche [54], cependant dans ce cas, tous les nœuds cibles doivent effectuer l'inversion d'une matrice de grande taille pour reconstituer les messages initiaux, qu'elle que soit leur vitesse de calcul. Il nous semble important, pour une primitive de communication collective a priori sans calcul, de ne pas proposer des algorithmes nécessitant des calculs de grande taille. Dans notre contexte, il faudrait en outre que les nœuds qui ne sont pas des destinations de la diffusion participent aux calculs en créant des combinaisons, ce qui nous semble encore plus délicat à justifier. Dans la suite, nous nous considérerons donc le problème de la diffusion restreinte sans calculs ou combinaisons de messages, et nous réservons l'utilisation des techniques de Network Coding à de futurs travaux.

4.4.2 Preuve de complexité

D'après le théorème 3.9, résoudre le problème de la diffusion sous le modèle un-port bidirectionnel revient à trouver un ensemble d'arbres A_a de diffusion restreinte pondérés par leur débit x_a, qui atteignent le débit optimal du programme linéaire 3.10 :

$$\begin{array}{l}\text{MAXIMISER } \rho_{\text{opt}} = \sum_{A_a \in \mathcal{A}} x_a, \\ \text{SOUS LES CONTRAINTES} \\ \begin{cases} \forall (i,j) \in E, & \displaystyle\sum_{\substack{A_a \in \mathcal{A} \\ (i,j) \in A_a}} x_a \cdot c_{i,j} \leqslant T_{i,j} \\ \forall P_i \in V, & \displaystyle\sum_{(i,j) \in E} T_{i,j} \leqslant 1 \\ \forall P_i \in V, & \displaystyle\sum_{(j,i) \in E} T_{j,i} \leqslant 1 \\ \forall A_a \in \mathcal{A}, & x_a \geqslant 0 \end{cases} \end{array} \tag{4.9}$$

De plus, d'après le théorème 3.1, on sait qu'on peut se restreindre à étudier les solutions composées uniquement d'un «petit» nombre d'arbres de diffusion. On peut donc définir le problème de la diffusion restreinte en se basant sur le problème DEC-PROG-LIN, comme suit. On rappelle qu'un notant $c_{i,j} = \frac{\alpha_{i,j}}{\beta_{i,j}}$ sous forme irréductible, on a noté $c_{\max} = \max_{i,j}\{\alpha_{i,j}, \beta_{i,j}\}$.

Définition 4.1 (DIFFUSION-RESTREINTE-COMPACTE)**.** *Étant donné une plate-forme $G = (V, E)$, un processeur source, un ensemble de destinations V_{cibles} et une borne K sur le débit, existe-t-il un ensemble pondéré d'arbres de diffusion (couvrant V_{cibles}) $(A_1, x_1), \ldots, (A_N, x_N)$ tels que :*

- *$N \leqslant |E| + 1$*
- *les $x_i = \frac{a_i}{b_i}$ vérifient $\log(a_i) + \log(b_i) \leqslant 2n(\log n + 2n \log c_{\max}) + n \log c_{\max}$,*
- *les arbres $(A_1, x_1), \ldots, (A_N, x_N)$ satisfont les contraintes du programme linéaire 4.9 et $\sum_i x_i \geqslant K$.*

Le théorème suivant, qui s'appuie sur l'étude théorique du chapitre précédent, montre que la formulation compacte ci-dessus est suffisante pour le problème d'optimisation du débit de la diffusion restreinte :

Théorème 4.10. *Étant donné une plate-forme $G = (V, E)$, un processeur source, un ensemble de destinations V_{cibles} et une borne K sur le débit, s'il existe un ordonnancement (quelconque) des communications réalisant la diffusion restreinte d'une série de N messages en temps $T(N)$ avec*

$$\limsup \frac{N}{T(N)} \geqslant K,$$

alors il existe une solution à DIFFUSION-RESTREINTE-COMPACTE(V, E, P_{source}, V_{cibles}, K).

Démonstration. On considère un ordonnancement quelconque pour qui réalise une série de N diffusions restreintes en temps $T(N)$. Tout d'abord, nous allons le décrire sous la forme de schémas d'allocation et de schémas de communication. On considère la diffusion du $k^{\text{ème}}$ message

de la série dans cet ordonnancement. Supposons que le processeur P_i reçoive deux fois ce message. Alors on peut supprimer la deuxième réception de ce message de l'ordonnancement, sans changer ni son temps d'exécution ni sa validité. Donc les arêtes sur lequel le $k^{\text{ème}}$ message de la série est diffusée forment un arbre enraciné en P_{source}. Comme il en est de même pour tous les messages de la série, et que certains de ces messages empruntent le même arbre, on note $n_a(N)$ le nombre de messages parmi les N premiers messages de la série, diffusés en utilisant l'arbre $A_a \in \mathcal{A}$. De plus, nous appelons $T_{i,j}(N)$ le temps total d'utilisation de l'arête (i,j) pour les N premiers messages de la série.

Les contraintes suivantes doivent être respectées par $n_i(N)$ et $T_{i,j}(N)$.

- Le temps d'occupation des arêtes doit être suffisant pour permettre l'envoi des messages pour tous les arbres de diffusion :

$$\forall (i,j) \in E, \quad \sum_{\substack{A_a \in \mathcal{A} \\ (i,j) \in A_a}} n_a(N) \cdot c_{i,j} \leqslant T_{i,j}(N)$$

- Le temps d'occupation d'un port de sortie d'un processeur doit être inférieur au temps total de l'ordonnancement :

$$\forall P_i \in V, \quad \sum_{(i,j) \in E} T_{i,j}(N) \leqslant T(N)$$

- Il en va de même pour le port d'entré d'un processeur :

$$\forall P_i \in V, \quad \sum_{(j,i) \in E} T_{j,i}(N) \leqslant T(N)$$

- De plus, ces quantités sont positives :

$$\forall A_a \in \mathcal{A}, \quad n_a(N) \geqslant 0$$

On pose

$$x_a = \frac{n_a(N)}{T(N)} \text{ et } T_{i,j} = \frac{T_{i,j}(N)}{T(N)}$$

Ces quantités définissent une solution valide (mais pas nécessairement optimale) du programme linéaire 4.9, qui atteint la valeur objective suivante :

$$\sum_{A_a \in \mathcal{A}} x_a = \sum_{A_a \in \mathcal{A}} \frac{n_a(N)}{T(N)} = \frac{N}{T(N)}.$$

Soit ρ_{opt} la valeur objective optimale du programme linéaire 4.9. On a donc $\frac{N}{T(N)} \leqslant \rho_{\text{opt}}$, et ce pour toute valeur de N. Donc $\limsup \frac{N}{T(N)} \leqslant \rho_{\text{opt}}$. Comme $\limsup \frac{N}{T(N)} \geqslant K$, on a $K \leqslant \rho_{\text{opt}}$.

Il existe une solution programme linéaire qui réalise un débit $\rho_{\text{opt}} \geqslant K$. D'après le théorème 3.1, on sait qu'il existe une solution compacte qui réalise un débit supérieur à K, c'est-à-dire un ensemble pondéré d'arbre $(A_1, x_1), \ldots, (A_N, x_N)$, avec

- $N \leqslant |E|$,
- $\log(a_i) + \log(b_i) \leqslant 2n(\log n + 2n \log c_{\max}) + n \log c_{\max}$ (en notant $x_i = \frac{a_i}{b_i}$),
- $\sum_i x_i \geqslant K$

Cet ensemble pondéré d'arbres $(A_1, x_1), \ldots, (A_N, x_N)$ est une solution à DIFFUSION-RESTREINTE-COMPACTE(V, E, P_{source}, V_{cibles}, K). ■

Maintenant que nous avons défini de façon compacte le problème de la diffusion restreinte et que nous l'avons lié au problème général, nous nous intéressons à sa complexité.

Théorème 4.11. *DIFFUSION-RESTREINTE-COMPACTE(V, E, P_{source}, V_{cibles}, K) est NP-complet et n'appartient pas à la classe APX.*

Démonstration. La preuve que DIFFUSION-RESTREINTE-COMPACTE est dans NP est une adaptation directe de la preuve que nous avons donné pour DEC-PROG-LIN-RED dans le théorème 3.1 au début du chapitre précédent.

Pour montrer que DIFFUSION-RESTREINTE-COMPACTE est NP-complet, nous utilisons une réduction depuis le problème de la couverture minimale de sommets, connu pour être NP-complet [51, problème MINIMUM-SET-COVER]. On considère une instance arbitraire $\mathcal{I}_1$ de ce problème :

$\mathcal{I}_1$: Soit C une collection de sous-ensembles d'un ensemble fini X de N éléments et soit B un entier telle que $1 \leqslant B \leqslant |C|$. existe-t-il dans C une couverture de X de taille au plus B ? En d'autre termes, existe-t-il un sous-ensemble C' de C de cardinal au plus B et tel que tout élément de X appartient à au moins un élément de C' ?

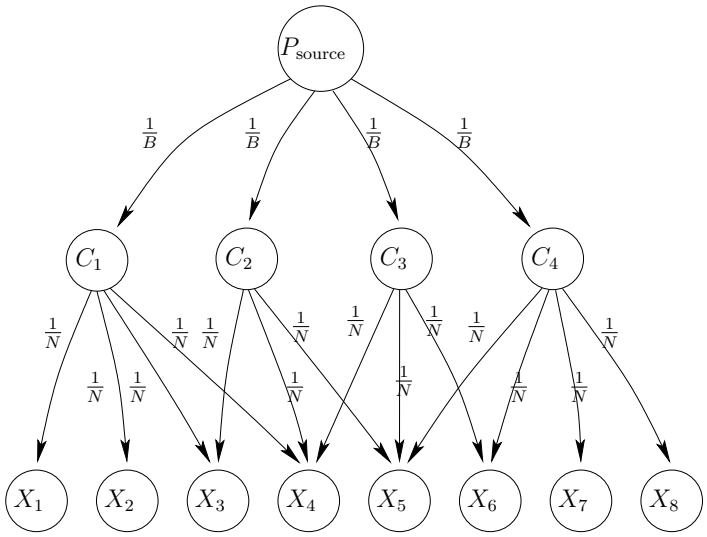


FIG. 4.10 – L'instance $\mathcal{I}_2$ de DIFFUSION-RESTREINTE-COMPACTE construite à partir de l'instance $\mathcal{I}_1$ suivante de MINIMUM-SET-COVER : $X = \{X_1, \ldots, X_8\}$, $C = \{\{X_1, X_2, X_3, X_4\}, \{X_3, X_4, X_5\}, \{X_4, X_5, X_6\}, \{X_5, X_6, X_6, X_8\}\}$.

En partant de $\mathcal{I}_1$, nous construisons l'instance suivante $\mathcal{I}_2$ de DIFFUSION-RESTREINTE-COMPACTE. Le graphe de la plate-forme (voir figure 4.10) est composé de :

- un processeur source P_{source},
- $|C|$ processeurs C_i $(1 \leqslant i \leqslant |C|)$, un pour chaque élément dans C,
- N processeurs X_i $(1 \leqslant i \leqslant N)$, un pour chaque élément dans X, ce sont les destinations de la diffusion $(X_i \in V_{\text{cibles}})$.

P_{source} est relié à chaque C_i par une arête de coût $1/B$. Un processeur C_i est relié à un processeur X_j par une arête de coût $1/N$, si et seulement si $X_j \in C_i$. Ce schéma risque de ne pas être un arbre, si un X_j appartient à plusieurs C_i de C'. On ajoute donc une arête (C_i, X_j) pour $C_i \in C'$ et X_j, seulement s'il n'existe pas $C_{i'} \in C'$ avec $i' < i$ et $X_j \in C_{i'}$. Enfin, on pose $K = 1$. La taille de $\mathcal{I}_2$ est clairement polynomiale en la taille de $\mathcal{I}_1$. Nous montrons que $\mathcal{I}_2$ a une solution si et seulement si $\mathcal{I}_2$ en admet une.

- Supposons que $\mathcal{I}_1$ ait une solution. Il existe donc une couverture C' de X, de cardinal au plus B. Nous construisons un unique arbre de diffusion A comme ceci : pour tout C_i dans la couverture C', nous ajoutons les arêtes $(P_{\mathsf{source}}, P_i)$ et toutes les arêtes (P_i, P_j) (pour $X_j \in C_i$) à l'arbre A. Pour utiliser cet arbre, P_{source} doit envoyer $|C'| \leqslant B$ messages sur des liens de communication de coût $1/B$, ce qui prend moins d'une unité de temps. Chaque processeur C_i dans C' reçoit un message (en temps $1/B$) et doit le transmettre à $|C_i| \leqslant N$ processeurs, en utilisant des liens de coût $1/N$. Toutes les temps d'occupation des ports d'entrée et de sortie des processeurs sont donc inférieurs à une unité de temps. De plus, comme C' est une couverture des X_j, tous les X_j reçoivent le message d'un $C_i \in C'$. On a donc construit un arbre de diffusion partielle de débit supérieur à $B = 1$; $\mathcal{I}_2$ admet une solution.
- Supposons maintenant que $\mathcal{I}_2$ ait une solution. Il existe donc un ensemble pondéré d'arbres $(A_1, x_1), \ldots, (A_N, x_N)$ réalisant un débit supérieur à 1, c'est-à-dire avec $\sum_i x_i \geqslant 1$. On note δ_i le degré sortant P_{source} dans l'arbre A_i, c'est-à-dire le nombre de processeurs C_i servant de relais dans l'arbre A_i. Pour effectuer toutes les communications sortant de P_{source} pour l'arbre A_i, il faut $x_i \cdot \delta_i / B$. Pour que toutes les communications de l'ensemble d'arbres puisse se faire en une unité de temps, on a :

$$\sum_i \frac{x_i \cdot \delta_i}{B} \leqslant 1 \tag{4.10}$$

  On note $\delta_{i_{min}} = \min_i \delta_i$. Par l'absurde, supposons que $\delta_{i_{min}} > B$, on a alors

$$\sum_i \frac{x_i \cdot \delta_i}{B} \geqslant \sum_i \frac{x_i \cdot \delta_{i_{\min}}}{B} > \sum_i x_i \geqslant 1 \tag{4.11}$$

  ce qui contredit l'équation précédente. Donc il existe un arbre de diffusion $A_{i_{\min}}$ qui utilise au plus B éléments C_i comme relais. Comme $A_{i_{\min}}$ est un arbre de diffusion restreinte, il couvre tous les éléments X_j. On note C' l'ensemble des éléments C_i utilisés par cet arbre. C' est une couverture de X de cardinal au plus B, donc $\mathcal{I}_1$ admet une solution.

Nous montrons maintenant que DIFFUSION-RESTREINTE-COMPACTE n'appartient pas à la classe APX. Supposons qu'il existe un algorithme fournissant en temps polynomial une λ-approximation de DIFFUSION-RESTREINTE-COMPACTE. Pour tout instance $\mathcal{I}_1$ du problème MINIMUM-SET-COVER, nous créons l'instance $\mathcal{I}_2$ présentée ci-dessus, et nous appliquons l'algorithme d'approximation de DIFFUSION-RESTREINTE-COMPACTE sur cette instance. Ceci peut être fait en temps polynomial, puisque la transformation est polynomiale. L'algorithme d'approximation de DIFFUSION-RESTREINTE-COMPACTE fournit un ensemble pondéré d'arbres $\{(A_1, x_1), \ldots, (A_N, x_N)\}$ vérifiant les deux première conditions de la définition 4.1 (nombre borné d'arbres et complexité bornée du codage des coefficients) et de plus,

$$\sum_{i=1}^{N} x_i \geqslant \frac{1}{\lambda}.$$

Pour chaque arbre A_i de cette approximation, nous calculons δ_i le degré du processeur source dans A_i, et nous choisissons l'arbre $A_{i_{\min}}$ qui a le degré minimum : $\delta_{i_{\min}} = \min \delta_i$. Ceci peut se faire en temps polynomial vu le nombre d'arbres et la complexité de codage des x_i. Comme précédemment, pour que toutes les communications émises par P_{source} puissent se faire en un temps unité, on a

$$\sum_i \frac{x_i \cdot \delta_i}{B} \leqslant 1$$

Par l'absurde, supposons que $\delta_{i_{\min}} > \lambda B$. Alors nous avons

$$\sum_i \frac{x_i \cdot \delta_i}{B} \geqslant \sum_i \frac{x_i \cdot \delta_{i_{\min}}}{B} = \frac{\delta_{i_{\min}}}{B} \sum_i x_i > \lambda \sum_i x_i$$

Or $\sum_i x_i \geqslant \frac{1}{\lambda}$ ce qui contredit l'inégalité précédente. Comme les messages transmis par l'arbre $A_{i_{\min}}$ doivent atteindre tous les processeurs X_i, les processeurs C_i utilisés comme relais dans cet arbre forment une couverture des X_i, de cardinal $\delta_{i_{\min}} \leqslant \lambda B$.

Nous avons donc un algorithme construisant en temps polynomial une λ-approximation de MINIMUM-SET-COVER, or on sait que MINIMUM-SET-COVER ne possède pas de tel algorithme (voir [5]). Donc il n'existe pas d'algorithme fournissant en temps polynomial une λ-approximation de DIFFUSION-RESTREINTE-COMPACTE. ■

Cas du modèle un-port unidirectionnel Nous avons établi la complexité du problème de la distribution restreinte pour le modèle un-port bidirectionnel. Ce résultat s'étend facilement au cas du modèle unidirectionnel : il suffit de dupliquer les processeurs qui ont besoin de faire à la fois des émissions et des réceptions en deux parties, comme on a fait pour la construction du graphe biparti G_B et en les reliant par un lien de coût nul. Dans l'instance $\mathcal{I}_2$, on transforme chaque processeur C_i en deux processeurs C_i^{in} et C_i^{out} et on rajoute une arête $(C_i^{\text{in}}, C_i^{\text{out}})$ de coût 0. L'arête $(P_{\mathsf{source}}, C_i)$ est transformée en une arête $(P_{\mathsf{source}}, C_i^{\text{in}})$ et les arêtes (C_i, X_j) sont transformées en (C_i^{out}, X_j). On peut alors utiliser la preuve précédente pour montrer la difficulté de ce problème sous le modèle unidirectionnel.

Cette construction est générale : si un problème de communications collectives est difficile sous le modèle un-port bidirectionnel, il l'est également sous le modèle unidirectionnel.

Nous venons de d'exhiber la première primitive de communications collectives dont le calcul du débit est NP-complet, même sous le modèle un-port bidirectionnel. Ce n'est pas un cas isolé : dans la prochaine partie, nous allons montrer le même résultat pour une autre primitive, le calcul des préfixes.

Autres résultats Comme le problème de la diffusion restreinte est NP-complet, nous avons élaborés des heuristiques pour résoudre ce problème. Ces heuristiques cherchent à construire des arbres couvrant les cibles de V_{cibles}, tout en maximisant le débit de diffusion obtenu. Certaines sont basées sur le résultat du programme linéaire présenté ci-dessus, qui fournit une borne supérieure sur le débit, d'autres s'inspirent d'heuristiques existantes pour le problème de Steiner. Ces heuristiques sont présentées et comparées à l'aide des simulations dans [21].

4.4.3 Extension au calcul des préfixes

Nous nous intéressons ici au calcul des préfixes, également connu sous le nom anglais «Parallel Prefix» et proche de la primitive `MPI_Scan` du standard MPI. Cette opération est très proche de la réduction, sauf que les résultats partiels $v_0 \oplus \cdot \oplus v_i$ doit également être calculé et mis à disposition du processeur possédant l'indice logique i.

Rappelons les notations de la réduction que nous utilisons ici. Parmi les processeurs de la plate-forme, certains processeurs $P_{r_i} \in \mathcal{R}$ possèdent une valeur v_i (pour $0 \leqslant i \leqslant N$). Le but est qu'à la fin du calcul le chaque processeur P_{r_i} possède la valeur $v_0 \oplus \cdot \oplus v_i$, où $\oplus$ est un opérateur associatif et non commutatif. Comme pour la réduction, nous notons $v_{[k,m]}$ le résultat partiel $v_k \oplus \cdot \oplus v_m$ et nous appelons $T_{k,l,m}$ la tâche de calcul associé à l'opération $v_{[k,m]} = v_{[k,l]} \oplus v_{[l+1,m]}$.

Contrairement au cas de la réduction au seule la valeur $v_0 \oplus \cdot \oplus v_N$ doit être calculée, une même tâche de calcul peut être effectué plusieurs fois et le résultat partiel $v_{[0,i]}$ peut être utilisé pour calculé $v_{[0,i+1]}$ ou bien ne pas être réutilisé : si par exemple le lien de communication (P_i, P_{i+1}) a une faible capacité de calcul, on préférera ne pas réutiliser le résultat $v_{[0,i]}$ en P_{i+1} et refaire une partie des calculs sur d'autres processeurs.

Rappelons également qu'on note $\mathsf{flops}(k,l,m)$ la quantité de calcul nécessaire à la tâche $T_{k,l,m}$ et z_{P_i} le temps de calcul d'une tâche unitaire sur le processeur P_i. Ainsi le processeur P_i aura besoin de $z_{P_i} \times \mathsf{flops}(k,l,m)$ unités de calcul pour traiter une tâche $T_{k,l,m}$. De plus, on appelle $\mathsf{size}([k,m])$ la taille d'un message $v_{[k,m]}$, de sorte que la communication d'un tel message sur l'arête (i,j) occupera cette arête pendant $c_{i,j} \times \mathsf{size}([k,m])$ unités de temps.

Les schémas d'allocation associés à cette opération sont très proches des arbres de réduction. Ce sont des ensembles de messages localisés, de tâches de calcul et de transfert organisés comme expliqué au chapitre 2 pour la réduction ; simplement, au lieu de n'avoir qu'un message de sortie $(v_{[0,N]}, P_{\mathsf{cible}})$, ils doivent posséder tous les messages $(v_{[0,i]}, P_{r_i})$, pour $P_{r_i} \in \mathcal{R}$.

On définit le problème de décision associé à l'optimisation du débit d'une opération de calcul des préfixes.

Définition 4.2 (PARALLEL-PREFIX-COMPACT)**.** *Étant donné une plate-forme $G = (V,E)$, un ensemble de processeurs participants $\mathcal{R}$ et une borne K sur le débit, existe-t-il un ensemble pondéré de schémas d'allocation pour le calcul des préfixes $(A_1,x_1),\dots,(A_N,x_N)$ tels que :*

- $N \leqslant |E| + 1$
- *les $x_i = \frac{a_i}{b_i}$ vérifient $\log(a_i) + \log(b_i) \leqslant 2n(\log n + 2n \log c_{\max}) + n \log c_{\max}$,*
- *les arbres $(A_1,x_1),\dots,(A_N,x_N)$ satisfont les contraintes du programme linéaire 4.9 et $\sum_i x_i \geqslant K$.*

Comme pour le problème de la diffusion restreinte, on peut montrer que formulation compacte du problème de calcul des préfixes est suffisamment générale. La preuve de ce résultat est similaire à celle du théorème 4.10.

Théorème 4.12. *Étant donné une plate-forme $G = (V,E)$, un ensemble de processeurs participants $\mathcal{R}$ et une borne K sur le débit, s'il existe un ordonnancement (quelconque) réalisant une série de N calculs des préfixes en temps $T(N)$ avec*

$$\limsup \frac{N}{T(N)} \geqslant K,$$

alors il existe une solution à PARALLEL-PREFIX-COMPACT(V, E, $\mathcal{R}$, K).

Nous montrons que l'optimisation du débit d'une série de calculs des préfixes est un problème NP-complet.

Théorème 4.13. *PARALLEL-PREFIX-COMPACT(V, E, $\mathcal{R}$, K) est NP-complet.*

Démonstration. L'appartenance de ce problème à la classe NP découle directement, comme pour la diffusion restreinte, du théorème 3.1 du chapitre précédent.

Pour montrer que ce problème est NP-complet, nous utilisons encore une fois une réduction depuis MINIMUM-SET-COVER. À partir d'une instance $\mathcal{I}_1$ composée d'un ensemble X à N éléments, d'une collection C de sous-ensembles de C et d'une borne B, nous créons une instance $\mathcal{I}_2$ (illustrée sur la figure 4.11) composée de $2N + |C| + 1$ processeurs, répartis comme suit :

- un processeur P_S,
- $|C|$ processeurs $C_1, \ldots, C_{|C|}$,
- N processeurs X_i, pour $X_i \in X$,
- N processeurs X'_i, pour $X_i \in X$.

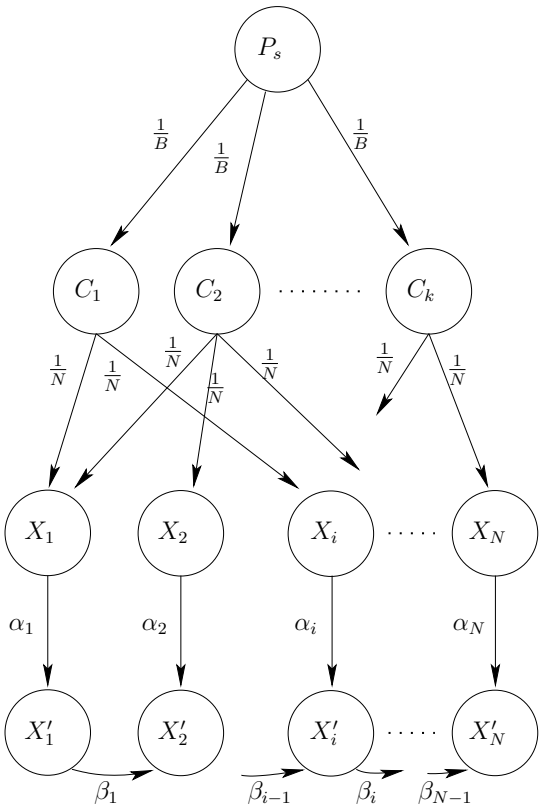


FIG. 4.11 – Topologie de l'instance $\mathcal{I}_2$.

Les liens de communication sont les suivants :

- pour chaque C_i, une arête (P_s, C_i) de coût $1/B$,
- une arête (C_i, X_j), de coût $1/N$, si et seulement si $X_j \in C_i$,
- pour chaque X_i, une arête (X_i, X'_i), de coût $\alpha_i = \dfrac{1}{i} - \dfrac{1}{N+1}$
- pour $i < N$, une arête (X'_i, X'_{i+1}), de coût $\beta_i = \dfrac{1}{i+1} + \dfrac{1}{i(N+1)}$.

L'ensemble des processeurs participants au calcul et $(P_{r_0}, \ldots, P_{r_N}) = (P_s, X'_1, \ldots, X'_N)$. Toutes les tâches de calcul ont un coût unitaire : $\mathsf{flops}(k,l,m) = 1$ pour tout $T_{k,l,m}$ et chaque processeur P de $\mathcal{R}$ a la même puissance de calcul : $z_{P_s} = z_{X'_i} = 1/N$, alors que les autres processeurs ne calculent pas : $z_{C_i} = z_{X_j} = +\infty$. Nous supposons que la taille d'un message $v_{[k,m]}$ est proportionnel à la longueur de l'intervalle : $\mathsf{size}([k,m]) = k - m + 1$. Nous posons enfin $B = 1$. La taille de $\mathcal{I}_2$ est clairement polynomiale en celle de $\mathcal{I}_1$. Nous montrons que $\mathcal{I}_2$ a une solution si et seulement si $\mathcal{I}_2$ en a une :

- Supposons d'abord que $\mathcal{I}_1$ ait une solution : il existe une couverture C' des X_i, de taille au plus B. Nous construisons un seul schéma d'allocation pour le calcul des préfixes comme suit :
  - Le processeur P_s envoie sa valeur v_0 aux processeurs C_i tels que $C_i \in C'$, comme $|C'| \leqslant B$, que $\mathsf{size}([0,0]) = 1$ et que les liens utilisés ont un coût $1/B$, cela nécessite moins d'une unité de temps.
  - Les processeurs C_i qui ont reçu v_0 le retransmette aux X_j qu'ils couvrent. On utilise la même technique que précédemment pour éviter qu'un X_j reçoivent deux fois cette valeur : C_i envoie v_0 a X_j si et seulement si $X_j \in C_i$ et il n'existe pas $i' < i$ avec $X_j \in C_{i'}$. Un processeur C_i envoie au plus N messages de taille 1 et un processeur X_j en reçoit au plus $B \leqslant N$, donc comme les liens impliqués sont de coût $1/N$, ceci peut se faire en une unité de temps.
  - X_j envoie v_0 à X'_j en temps $\alpha_j \leqslant 1$.
  - Pour $i < N$, X'_i envoie à X'_{i+1} les messages $v_1, \ldots v_i$, chacun de taille 1. Le temps d'émission de X'_i est donc $i \times \beta_i = \frac{i}{i+1} + \frac{1}{N+1} \leqslant 1$. Le temps de réception d'un X'_i est $1 \times \alpha_i + (i-1)\beta_{i-1} = 1$. Ces communications peuvent donc se faire en temps 1.
  - Chaque processeur X_i effectue tous les calculs nécessaires au calcul de son résultat $v_{[0,i]} = (\cdots((v_0 \oplus v_1) \oplus v_2)\cdots) \oplus x_i$. Son temps de calcul est donc

$$\sum_{j=1}^{i} z_{X'_i} \times \mathsf{flops}(0,j,j) = \sum_{j=1}^{i} 1/N \times 1 = i/N \leqslant 1$$

  Chacune de ces opérations peut être effectuée en temps 1, on atteint donc bien un débit d'une opération par unité de temps, donc $\mathcal{I}_2$ a une solution.
- Supposons maintenant que $\mathcal{I}_2$ ait une solution. Il existe donc un ensemble pondéré de schémas d'allocation $(A_1, x_1), \ldots, (A_N, x_N)$ réalisant un débit supérieur à 1, c'est-à-dire avec $x = \sum_i x_i \geqslant 1$. Dans un premier temps, on montre que x messages contenant des valeurs v_0 traversent chaque arête (X_i, X'_i) en une unité de temps. Procédons par l'absurde et supposons qu'il existe une arête (X_i, X'_i) qui transmet une quantité $y < x$ de tels messages. Pour pouvoir effectuer obtenir x valeurs $v_{[0,i]}$ en une unité de temps, le processeur X'_i doit obtenir, d'une manière ou d'une autre, x valeur v_0, soit sous la forme d'un message dédié $v_{[0,0]}$, soit sous la forme d'un message $v_{[0,j]}$. Si $i = 1$, X'_i n'a d'autres moyens de recevoir ces valeurs que par le lien (X_1, X'_1), donc ce lien communique au moins x messages $v_{[0,0]}$ et $i \neq 1$. Le processeur X'_i reçoit donc les $z \geqslant x - y$ valeurs «manquantes» par le lien (X'_{i-1}, X'_i). On peut remarquer qu'avec notre modélisation des tailles des messages, transmettre une valeur $v_{[k,m]}$ est équivalent (en taille de messages) à transmettre chacune des valeur $v_k, \ldots, v_m$ séparément, ou bien toutes combinaisons de ces valeurs. Pour simplifier, nous considérons que seules des messages de type $v_{[k,k]}$ sont utilisés. Ceci n'induit pas de perte de généralité, puisque chaque processeur X'_i a la capacité de calcul nécessaire pour effectuer tous les calculs dont il a besoin. En plus des

z valeurs v_0, X'_i reçoit de X'_{i-1} les messages contenant des valeurs $v_1,\ldots,v_{i-1}$ et doit en recevoir x de chaque type en une unité de temps, pour atteindre le débit voulu. Le temps nécessaire à la réception de tous ces messages en X'_i est

$$\begin{aligned}
T^{\text{recp}}_{X'_i} &= \big(x(i-1)+z\big)\beta_{i-1} + y \times \alpha_i \\
&= \big(x(i-1)+z\big) \times \left(\frac{1}{i} + \frac{1}{(N+1)(i-1)}\right) + y \times \left(\frac{1}{i} - \frac{1}{N+1}\right) \\
&\geqslant \big(x(i-1)+z\big) \times \left(\frac{1}{i} + \frac{1}{(N+1)(i-1)}\right) + (x-z) \times \left(\frac{1}{i} - \frac{1}{N+1}\right) \\
&= x + z\frac{i}{(N+1)(i-1)}
\end{aligned}$$

Comme $\sum_i x_i \geqslant 1$ et $z > 0$, X'_i doit recevoir des messages pendant plus d'une unité de temps, ce qui est en contradiction contradiction avec les contraintes sur les schémas d'allocation. Donc chaque arête (X_i, X'_i) est utilisée pour transmettre au moins x messages v_0 pendant chaque unité de temps. Ceci signifie que dans chaque schéma d'allocation, tous les processeurs X_i reçoivent le message v_0 de P_S.
Nous procédons maintenant comme pour la diffusion restreinte de P_s aux X_i. On note δ_i le nombre de messages contenant v_0 envoyés par P_s aux processeurs C_j dans le schéma A_i (c'est le degré sortant de P_s dans le schéma d'allocation A_i) et $\delta_{i_{\min}}$ la plus petite de ces valeurs. D'après la contrainte un-port en sortie de P_s, on a :

$$\sum_i \frac{x_i \cdot \delta_i}{B} \leqslant 1, \quad \text{d'où } \sum_i x_i \leqslant \frac{B}{\delta_{i_{\min}}}$$

Si on suppose $\delta_{i_{\min}} > B$, on obtient $\sum_i x_i < 1$, ce qui est en contradiction avec la contrainte sur le débit réalisé par la solution. Donc il existe un schéma $A_{i_{\min}}$ utilisant au plus B processeurs C_i et couvrant tous les X_i. On en déduit une couverture de X de cardinal au plus B. Donc $\mathcal{I}_1$ a une solution.

■

En revanche, la réduction ci-dessus ne nous permet pas d'affirmer que le calcul des préfixes n'appartient pas à la classe APX, comme c'était le cas pour la diffusion restreinte.

4.5 Conclusion

Dans ce chapitre, nous avons étudié les communications sous le modèle un-port bidirectionnel. Nous avons proposé des algorithmes efficaces pour la recherche schémas d'allocation pour les primitives de distribution de données, de réduction et de diffusion. Nous avons également montré que certaines primitives étaient difficiles avec ce modèle : la diffusion restreinte ou le calcul des préfixes. La difficulté de ces problèmes réside intuitivement dans la capacité de dupliquer des messages (pour la diffusion restreinte), ou des tâches de calcul (pour le calcul des préfixes). Cette latitude accrue par rapport aux autres problèmes comme la distribution de données ou la réduction, pour lesquels il existe de strictes lois de conservation des messages, rend ces problèmes notablement plus difficile. Le cas de la diffusion, pour lequel nous sommes capables de construire un ensemble d'arbres réalisant le débit optimal, apparaît donc plutôt

comme une exception, et nécessite l'utilisation d'un puissant théorème d'extraction d'arbres. On peut résumer les résultats de complexité obtenus dans ce chapitre et le précédent dans le tableau suivant :

primitive \ modèle	un-port unidirectionnel	un-port bidirectionnel
distribution de données	polynomial (ellipsoïde)	polynomial (efficace)
diffusion	polynomial (ellipsoïde)	polynomial (efficace)
réduction	polynomial (ellipsoïde)	polynomial (efficace)
diffusion restreinte	NP-complet, non APX	NP-complet, non APX
calcul des préfixes	NP-complet	NP-complet

Extension aux graphes de tâches. Nous avons évoqué au chapitre 2 le cas des graphes de tâches, en remarquant que l'allocation d'un graphe de tâches sur une plate-forme n'était qu'un cas particulier de schéma d'allocation. Nous avons donc également étudié l'ordonnancement d'une série de graphes de tâches en utilisant une approche assez similaire à celle du cas de la réduction, c'est-à-dire en écrivant des contraintes sur les messages correspondants aux arêtes du graphe de tâches (les fichiers d'entrée/sortie de l'application). En particulier, on peut écrire, comme pour la réduction, une loi de conservation de ces messages. Cependant, alors que la structure en arbre du schéma d'allocation de la réduction permettait facilement d'extraire des schéma à partir d'une solution du programme linéaire, cela n'est plus le cas pour des graphes de tâches quelconques. Il faut alors prendre en compte une sorte d'historique des tâches, pour ne pas mélanger des fichiers provenant d'instances différentes. Dans le cas général, nous avons montré que le problème de calcul du débit optimal est NP-complet [22]. Cependant pour les graphes de tâches de profondeur de dépendance bornée, nous sommes en mesure de calculer le débit optimal et de construire un ensemble d'allocations qui le réalise, la profondeur de dépendance étant en quelque sorte le nombre maximal de chemins distincts (ne couvrant aucun sommet commun autre que les extrémités) dans le graphe de tâches entre deux sommets. Comme ces résultats pour les graphes de tâches ont déjà été présentés dans la thèse d'Arnaud Legrand [72] et l'habilitation à diriger des recherches d'Olivier Beaumont [15], nous avons choisi de ne pas les exposer en détail ici.

Tout au long de ce chapitre, nous avons présenté quelques illustrations expérimentales de nos résultats. Pour la primitive de réduction, nous avons également comparé les performances obtenues par notre approche et celle d'une approche plus simple, constituée d'un seule arbre similaire à celui que construit MPICH pour effectuer une réduction. Cette comparaison a été faite par simulation sur un grand nombre de plates-formes produites par un générateur de topologies et sur quelques autres plates-formes particulières. Sans surprise, notre approche réalise un débit bien meilleur qu'une approche à la MPICH. Toutefois, pour démontrer l'intérêt pratique de ces travaux, nous présentons dans le chapitre suivant les résultats d'une expérimentation réelle pour la primitive la plus utilisée, la diffusion de données.

Chapitre 5

Expérimentations : cas de la diffusion

Dans ce chapitre, nous présentons les expérimentations que nous avons effectuées pour l'opération de diffusion sur une plate-forme réelle. Nous avons justifié le modèle un-port en nous appuyant sur [86], qui montrent que des implantations de MPI se conforme à ce modèle. Cependant, dans notre implantation des mécanismes de diffusion, nous allons tenter d'utiliser des transferts concurrents pour qu'un processeur puisse diffuser un message reçu à chacun de ses fils, car il ne nous parait pas profitable d'imposer une contrainte un-port au niveau logiciel, en sérialisant volontairement les envois. Dans ce contexte, un modèle multi-port borné, comme celui introduit par Hong et Prasanna [63] parait plus justifié, et suffisamment simple pour être utilisé ici. Nous commençons donc par présenter ce modèle, ainsi que l'adaptation de notre travail précédent sur ce modèle. Nous présentons ensuite quelques heuristiques pour la diffusion n'utilisant qu'un seul arbre de diffusion, afin de quantifier le gain qu'apporte l'utilisation de plusieurs arbres, en contrepartie d'une implantation plus délicate. Engin nous détaillons l'implantation que nous proposons ainsi que les résultats obtenus.

5.1 Adaptation pour le modèle multi-port

5.1.1 Modèles un-port et multi-port borné

Comme nous l'avons dit en introduction (partie 1.2), d'autres modèles ont été proposés pour tenir compte de la congestion sur les interfaces de communications des processeurs et des routeurs. Le modèle qui semble le plus intéressant de ce point de vue est le modèle multi-port borné. Celui-ci suppose que plusieurs transferts entrants ou sortant d'un nœuds peuvent coexister, mais que la capacité entrante ou sortante du nœud est limitée. On appelle $B_{\text{in}}(P_i)$ la quantité maximale de données qui peut être reçue par P_i en une unité de temps (i.e. la bande-passante en entrée), et $B_{\text{out}}(P_i)$ la quantité de données qui peut être émise par P_i en une unité de temps (i.e. la bande-passante en sortie de P_i). On note $\mathsf{data}(P_i \to P_j)$ la quantité de données envoyées en temps unité de P_i à P_j. Le modèle multi-port borné impose que :

$$\forall P_i \in V \quad \sum_{P_j,(i,j)\in E} \mathsf{data}(P_i \to P_j) \leqslant B_{\text{out}}(P_i)$$

$$\forall P_i \in V \quad \sum_{P_j,(j,i)\in E} \mathsf{data}(P_j \to P_i) \leqslant B_{\text{in}}(P_i)$$

Dans ce modèle, il faut également rajouter des contraintes de bande-passante sur chaque lien. Si un nœud P_i ne transfère des données qu'à un seul autre nœud P_j, il se peut que la bande-passante de ce transfert ne soit ni $B_{\text{out}}(P_i)$, ni $B_{\text{in}}(P_j)$, mais qu'elle soit limitée par la bande-passante maximale $b_{i,j}$ du lien de communication (P_i, P_j) :

$$\forall (P_i, P_j) \in E \quad \mathsf{data}(P_i \to P_j) \leqslant b_{i,j}$$

Notons que cette contrainte de limitation de la capacité des liens était déjà prise en compte dans le modèle un-port. En effet, dans un modèle sans latence, on peut assimiler la bande-passante $b_{i,j}$ du modèle multi-port à l'inverse du coût $c_{i,j}$ de transmission d'un message de taille unité du modèle un-port. La contrainte précédente devient ainsi :

$$\forall (P_i, P_j) \in E \quad \mathsf{data}(P_i \to P_j) \times c_{i,j} \leqslant 1$$

Cette inéquation est une conséquence directe de la contrainte un-port en sortie de P_i (ou en entrée de P_j), que l'on rappelle ici :

$$\forall P_i \in V \quad \sum_j \mathsf{data}(P_i \to P_j) \times c_{i,j} \leqslant 1$$

Un des avantages de ce modèle sur le modèle un-port est qu'il permet d'exprimer des problèmes d'optimisation de débit comme des problèmes de multi-flot, et ainsi profiter de nombreux travaux qui étudient ces problèmes. On peut par exemple citer le travail de Hong et Prasanna [63] qui utilisent des techniques de calcul distribué du flot maximum pour déterminer une allocation pour un problème de tâches indépendantes.

5.1.2 Résolution pour le modèle multi-port

Nous montrons ici que ce modèle peut être pris en compte avec notre formulation utilisant la programmation linéaire. Plus précisément pour le cas de la diffusion, nous montrons que la technique de résolution efficace développée dans la partie 4.2 peut être étendue au modèle multi-port.

Adaptation de la méthode générale Dans le modèle multi-port, il n'y a plus lieu d'utiliser des couplages pour les communications. En effet, nous avons introduit les schémas de communication comme étant des ensembles de communications qui peuvent avoir lieu simultanément, dans le modèle de communication choisi. Dans le modèle multi-port, tout ensemble de communications peut avoir lieu simultanément, pourvu que les débits qui sont alloués aux différentes communications respectent les contraintes des ressources. Nous n'avons donc plus besoin de découper l'ensemble de communications en couplages.

Pour adapter le programme linéaire général 3.1 de la partie 3.1.1 au cas multi-port, rappelons que le x_a désigne la quantité de messages utilisant le schéma d'allocation A_a par unité de temps, c'est-à-dire le débit d'utilisation de ce schéma. Comme dans le chapitre 3, nous considérons que tous les messages sont de taille unité. La quantité $\mathsf{data}(P_i \to P_j)$ de messages utilisant l'arête (i, j) en une unité de temps est donc :

$$\mathsf{data}(P_i \to P_j) = \sum_{\substack{A_a \in \mathcal{A} \\ (i,j) \in A_a}} x_a$$

Avec les contraintes développées ci-dessus, on obtient le programme linéaire suivant :

$$
\begin{aligned}
&\text{MAXIMISER } \rho_{\text{opt}} = \sum_{A_a \in \mathcal{A}} x_a, \\
&\text{SOUS LES CONTRAINTES} \\
&\begin{cases}
\text{(5.1a)} \quad \forall P_i \in V & \displaystyle\sum_{P_j,(i,j)\in E} \sum_{\substack{A_a \in \mathcal{A} \\ (i,j)\in A_a}} x_a \leqslant B_{\text{out}}(P_i) \\
\text{(5.1b)} \quad \forall P_i \in V & \displaystyle\sum_{P_j,(j,i)\in E} \sum_{\substack{A_a \in \mathcal{A} \\ (j,i)\in A_a}} x_a \leqslant B_{\text{in}}(P_i) \\
\text{(5.1c)} \quad \forall (P_i,P_j) \in E & \displaystyle\sum_{\substack{A_a \in \mathcal{A} \\ (i,j)\in A_a}} x_a \leqslant b_{i,j} \\
\text{(5.1d)} \quad \forall A_a \in \mathcal{A}, & x_a \geqslant 0
\end{cases}
\end{aligned}
\tag{5.1}
$$

Cette formulation a l'avantage d'être générale pour tout problème de communications collectives. Cependant, elle n'est pas très utile pour concevoir des algorithmes efficaces. On peut la rapprocher de l'approche générale développée à la partie 3.2.1 pour le modèle un-port bidirectionnel. Comme nous l'avons fait pour ce modèle, maintenant que nous avons formulé les contraintes du modèle de communication sans utiliser de couplages, nous allons spécifier le programme linéaire en fonction des primitives de communications collectives étudiées. Nous nous intéressons ici uniquement au cas de la diffusion.

Dans ce modèle multi-port, on peut de contenter de décrire un ordonnancement sous la forme d'un ensemble pondéré de schémas d'allocation : chacun de ces schémas d'allocation sera utilisé avec le débit prescrit par sa pondération et nous savons qu'il est possible d'effectuer simultanément toutes les communications nécessaires avec ces débits. D'autre part, comme nous considérons ici des primitives de communications collectives sans calcul, il n'y a pas lieu de fournir un ordonnancement pour les tâches de calcul.

Résolution efficace Nous reprenons la méthode développée pour la diffusion de données sous le modèle un-port bidirectionnel (partie 4.2). Nous rappelons les notations utilisées :

- Nous appelons $\mathsf{send}(P_i \to P_j, m_k)$ le nombre de messages à destination de P_k transmis en une unité de temps sur l'arête (i,j).
- On note $s(P_i \to P_j)$ le nombre total de messages transmis par l'arête (i,j) en une unité de temps. Comme précédemment, on fait l'hypothèse optimiste que les messages transmis sur une arête à destination de différents processeurs se recouvrent complètement :

$$\forall (i,j) \in E \; s(P_i \to P_j) = \sum_{P_k \in V} \mathsf{send}(P_i \to P_j, m_k)$$

Avec cette hypothèse optimiste, nous obtenons le programme linéaire suivant qui définit une borne supérieure du débit optimal. C'est l'adaptation du programme linéaire 4.5 au modèle

multi-port.

$$
\begin{array}{l}
\text{MAXIMISER } \rho_{\text{opt}}, \\
\text{SOUS LES CONTRAINTES}
\end{array}
\left\{
\begin{array}{lll}
\text{(5.2a)} & \forall P_k \in V & \displaystyle\sum_{(P_{\mathsf{source}},P_j)\in E} \mathsf{send}(P_{\mathsf{source}} \to P_j, m_k) = \rho_{\text{opt}} \\
\text{(5.2b)} & \forall P_k \in V & \displaystyle\sum_{(P_j,P_k)\in E} \mathsf{send}(P_j \to P_k, m_k) = \rho_{\text{opt}} \\
\text{(5.2c)} & \left\{ \begin{array}{l} \forall P_k \in V, \forall P_i \\ P_i \neq P_k, P_{\mathsf{source}} \end{array} \right. & \displaystyle\sum_{(P_j,P_i)\in E} \mathsf{send}(P_j \to P_i, m_k) = \sum_{(P_i,P_j)\in E} \mathsf{send}(P_i \to P_j, m_k) \\
\text{(5.2d)} & \forall (i,j) \in E, & s(P_i \to P_j) = \max\limits_{P_k \in V} \{\mathsf{send}(P_i \to P_j, m_k)\} \\
\text{(5.2e)} & \forall P_i \in V & \displaystyle\sum_{P_j,(i,j)\in E} s(P_i \to P_j) \leqslant B_{\text{out}}(P_i) \\
\text{(5.2f)} & \forall P_i \in V & \displaystyle\sum_{P_j,(j,i)\in E} s(P_i \to P_j) \leqslant B_{\text{in}}(P_i) \\
\text{(5.2g)} & \forall (P_i,P_j) \in E & s(P_i \to P_j) \leqslant b_{i,j}
\end{array}
\right. \tag{5.2}
$$

Comme pour le modèle un-port bidirectionnel, le programme linéaire précédent donne une borne supérieur sur le débit atteignable.

Théorème 5.1. *Le programme linéaire 5.2 donne une borne supérieure pour le débit d'un ordonnancement réalisant une série d'opérations de diffusion sous le modèle multi-port borné.*

La preuve de ce résultat est similaire à celle du théorème 4.3.

Pour reconstruire un ordonnancement à partir d'une solution de ce programme linéaire, nous utilisons exactement la même méthode que celle présentée pour le modèle un-port bidirectionnel. Dans la partie 4.2, nous avons présenté une méthode pour construire, à partir d'une solution (send, s, T), reconstruire un ensemble d'arbre de diffusion réalisant le débit ρ_{opt}. On considère le graphe de plate-forme dont les arêtes (i,j) sont étiquetées par les valeurs de $s(P_i \to P_j)$: $G_s = (V, E, s)$. Nous avons montré dans le théorème 4.4, qu'on pouvait en temps polynomial, construire un ensemble d'arbres de diffusion $A_1, \ldots, A_N$ munis de pondérations $\lambda_1, \ldots \lambda_N$ tels que :

– La somme pondérée des arbres est «inclue» dans G_s :

$$\forall (i,j), \quad \sum_{\substack{u=1\ldots N \\ (i,j)\in A_u}} \lambda_u \leqslant s(P_i \to P_j)$$

– La somme des poids des arbres atteint le débit ρ_{opt} :

$$\sum_{u=1}^{N} \lambda_u = \rho_{\text{opt}}$$

– Le nombre d'arbres est borné par

$$N \leqslant |V|^3 + |E|$$

Ce résultat permet ici encore de reconstruire un ensemble d'arbres de diffusion réalisant le débit optimal ρ_{opt}.

5.2 Heuristiques pour un arbre de diffusion

Nous sommes donc en mesure de construire un ensemble pondérés d'arbres de diffusion qui réalise le débit optimal, à la fois pour le modèle un-port bidirectionnel et pour le modèle multi-port borné. Cependant, utiliser plusieurs arbres de diffusion concurrents demande un mécanisme de contrôle assez élaboré : comme les différents arbres ne sont pas utilisés avec le même débit, on doit créer pour chaque arbre, sur chaque nœud, un processus en charge des messages transmis par cet arbre, et il faut en même temps rassembler et réordonner les messages provenant de tous ces processus. Comme cette complexité peut être un frein important à l'implantation de cette stratégie, nous proposons ici des méthodes heuristiques pour construire un seul arbre de diffusion. Nous utilisons les deux modèles de communication précédents (un-port bidirectionnel et multi-port borné). Trouver l'arbre de diffusion réalisant le meilleur débit est un problème NP-complet pour les deux modèles considérés ; ce résultat est une conséquence directe de la NP-complétude du problème consistant à trouver un arbre couvrant de degré borné dans un graphe quelconque [51, problème DEGREE-CONSTRAINT-SPANNING-TREE].

Nous proposons des heuristiques fondées sur des méthodes classiques pour construire un arbre de poids minimum et également des méthodes fondée sur le résultat du programme linéaire.

5.2.1 Heuristiques fondées sur des techniques de graphe

Nous présentons ici des heuristiques assez simples, utilisant des techniques classiques sur les graphes. Ce sont des heuristiques pour le modèle de communication un-port bidirectionnel.

Élagage simple de la plate-forme (SIMPLE-PRUNE)

Cette première heuristique de construction d'arbre est la plus simple : on considère le graphe de la plate-forme $G = (V, E, c)$ dont les arêtes sont pondérées par leur coût de communication et on supprime des arêtes jusqu'à obtenir un arbre, (en prenant naturellement garde à ne pas déconnecter des sommets). Les arêtes sont supprimées en commençant par celles de poids c maximum, comme présenté dans l'algorithme ci-dessous.

```
Arbre ← toutes les arêtes de E
Tant que |Arbre| > |V| − 1 :
  L ← arêtes (i, j) de Arbre triées par valeurs de c_{i,j} décroissantes
  Pour toute arête e ∈ L :
    Si le graphe (V, Arbre\{e}) est connexe Alors
      Arbre ← Arbre\{e}
```

Élagage sophistiqué de la plate-forme (REFINED-PRUNE)

L'heuristique précédente tente de supprimer de l'arbre de diffusion toutes les arêtes qui ont un poids important. Pourtant, ceci n'est pas forcément la bonne stratégie : si un processeur a beaucoup de fils dans l'arbre (par exemple 10) et que toutes ses arêtes sortantes sont de poids moyen (par exemple 2), il va passer un temps important (dans l'exemple, 20 unités de temps) à transmettre un message de la série de diffusions à chacun de ses fils ; un processeur ayant un seul fils dans l'arbre connecté par une arête de poids important (par exemple 15) aura besoin d'un temps moins important (15 unités de temps) pour chaque message. Le débit d'un processeur dans l'arbre est donc inversement proportionnel à son degré sortant pondéré dans l'arbre, c'est-à-dire la somme des poids des arêtes sortantes utilisées. On va donc chercher à minimiser cette quantité, plutôt qu'à éliminer les arêtes de poids important. C'est ce que fait l'algorithme suivant, en calculant le degré sortant pondéré $\delta_{\text{out}}(P_i)$ d'un processeur P_i et en le maintenant au cours de l'exécution. Il s'agit maintenant de supprimer l'arête la plus coûteuse du processeur qui a le degré sortant maximum, sous réserve de garder le graphe connexe.

1: Arbre $\leftarrow$ toutes les arêtes de E
2: **Pour** tout sommet $P_i \in V$ **:**
3: $\quad \delta_{\text{out}}(P_i) \leftarrow \sum_{j,\ (i,j)\in E} c_{i,j}$
4: **Tant que** |Arbre| $> |V| - 1$ **:**
5: $\quad V' \leftarrow$ sommets de V triés par valeurs de $\delta_{\text{out}}(u)$ décroissantes
6: $\quad$ **Pour** $P_i \in V'$ **:**
7: $\quad\quad L \leftarrow$ arêtes sortantes de P_i triées par valeurs de $c_{i,j}$ décroissantes
8: $\quad\quad$ **Pour** toute arête $e = (i,j) \in L$ **:**
9: $\quad\quad\quad$ **Si** le graphe $(V, \text{Arbre}\backslash\{e\})$ est connexe **Alors**
10: $\quad\quad\quad\quad$ Arbre $\leftarrow$ Arbre$\backslash\{e\}$
11: $\quad\quad\quad\quad \delta_{\text{out}}(P_i) \leftarrow \delta_{\text{out}}(P_i) - c_{i,j}$
12: $\quad\quad\quad\quad$ Aller à la ligne 4

Arbre couvrant de degré sortant minimal (GROW-MIN-OUTDEG)

Cette heuristique s'inspire de l'algorithme de Prim [43] pour construire un arbre couvrant de poids minimal. La métrique habituelle pour le poids d'un arbre est la somme des poids des arêtes. Mais, comme nous l'avons remarqué ci-dessus, la somme des coûts de arêtes n'est pas une métrique très intéressante ici. Au contraire, nous cherchons à minimiser le maximum des degrés pondérés des nœuds. L'algorithme présente une adaptation de l'algorithme de Prim pour cette nouvelle métrique, dans lequel V' désigne l'ensemble des sommets de G couverts par l'arbre de diffusion courant. Lorsque nous prenons la décision d'ajouter une arête (i,j) à l'arbre, nous mettons à jour les coûts des autres arêtes (i,k) sortantes de P_i, de sorte que le poids $w(i,j)$ d'une arête (i,j) représente toujours le degré sortant du nœud P_i après insertion de (i,j) dans l'arbre. En sélectionnant l'arête de poids minimal $w(i,j)$, nous augmentons aussi peu que possible le maximum des degrés pondérés des nœuds de l'arbre.

```
Arbre ← ∅
V' ← {P_source}
Pour toute arête e = (i, j) :
  w(i, j) ← T_{i,j}
Tant que V' ≠ V :
  choisir l'arête (i, j) telle que P_i ∈ V', P_j ∉ V' et w(i, j) est minimal
  V' ← V' ∪ {j}
  Arbre ← Arbre ∪ {(i, j)}
  Pour toute arête (i, k) ∉ Arbre :
    w(i, k ← w(i, k) + w(i, j)
```

5.2.2 Heuristiques s'inspirant d'une solution du programme linéaire

Nous développons maintenant deux heuristiques qui se fondent sur une solution du programme linéaire. Ces heuristiques peuvent être construites aussi bien à partir d'une solution du programme linéaire 4.5 sous le modèle un-port bidirectionnel ou pour une solution du programme linéaire 5.2 sous le modèle multi-port borné.

Nous supposons donc que nous disposons d'une solution (send, s) d'un des programmes linéaires. Nous utilisons le graphe des communications de la solution du programme linéaire, c'est-à-dire le graphe de la plate-forme $G = (V, E)$ dont les arêtes (i, j) sont pondérées par le nombre de messages $s(P_i \to P_j)$ transmis dans la solution programme linéaire.

Élagage du graphe de communication (LP-PRUNE)

Dans cette heuristique, nous partons du graphe total des communications, puis nous supprimons les arêtes le moins utilisées par la solution du programme linéaire, c'est-à-dire les arêtes avec la plus petite quantité de messages $s(P_i \to P_j)$, sous réserve de conserver la connexité du graphe, jusqu'à obtenir un arbre. Cette heuristique est présentée dans l'algorithme ci-dessous.

```
Arbre ← toutes les arêtes de E
Tant que |Arbre| > |V| − 1 :
  L ← arêtes (i, j) de Arbre triées par valeurs de s(P_i → P_j) décroissantes
  Pour toute arête e ∈ L :
    Si le graphe (V, Arbre\{e}) est connexe Alors
      Arbre ← Arbre\{e}
```

Construction d'un arbre couvrant (LP-GROW)

Cette heuristique consiste à faire croître un arbre couvrant, à la façon de l'algorithme de Prim, en rajoutant toujours l'arête qui a la valeur de $s(P_i \to P_j)$ maximale.

Arbre $\leftarrow \emptyset$
$V' \leftarrow \{P_{\text{source}}\}$
Tant que $V' \neq V$ **:**
  choisir l'arête (i, j) telle que $i \in V'$, $j \notin V'$ et $s(P_i \rightarrow P_j)$ est maximal
  $V' \leftarrow V' \cup \{v\}$
  Arbre $\leftarrow$ Arbre $\cup \{(u, v)\}$

Comme ces deux heuristiques peuvent être utilisées partant d'une solution du programme linéaire selon le modèle un-port bidirectionnel ou le modèle multi-port borné, nous disposons de quatre heuristiques, que nous noterons OP-LP-PRUNE et OP-LP-GROW pour le modèle un-port, et MP-LP-PRUNE et MP-LP-GROW pour le modèle multi-port.

D'autres part, nous allons comparer ces heuristiques avec les stratégies optimales pour chacun des modèles de communication, en décomposant la solution du programme linéaire en un ensemble pondérés d'arbres de diffusion. Dans la suite, cette stratégie optimale sera nommée OP-LP-MULTI-TREE pour le modèle un-port et MP-LP-MULTI-TREE pour le modèle multi-port.

5.3 Expérimentations

Afin de tester notre méthodologie pour le cas de la diffusion et pour comparer les différentes stratégies développées ici – à un arbre, à plusieurs arbres, selon différents modèles – nous avons conduit une série d'expériences sur une plate-forme distribuée. Comme il nous fallait de préférence une plate-forme à grande échelle, avec un réseau irrégulier et autant que possible hétérogène, sur laquelle on souhaitait éviter la présence de pare-feu pour pouvoir ouvrir des connections TCP entre sites distants, nous avons choisi d'utiliser la grille de recherche Grid5000 pour conduire ces expériences. Nous détaillons l'implantation que nous avons réalisée et les résultats obtenus dans cette partie.

5.3.1 Méthodologie

Pour tester ces différentes stratégies pour la diffusion pipelinée de messages, nous avons développé une implantation distribuée générique, qui permet d'effectuer une diffusion de messages selon un ou plusieurs arbres pondérés. Le contrôle de cette application distribuée s'apparente à une application maître-esclaves : un processus maître contrôle l'exécution de plusieurs esclaves, ou agents. Chaque agent possède donc un processus léger (ou *thread*) qui dialogue avec le maître, pour recevoir les consignes d'exécutions et renvoyer les résultats obtenus (le débit de réception qu'il a observé).

En fonction des consignes reçues du maître, un agent crée pour chaque arbre de diffusion, les processus léger suivants :

- si l'agent n'est pas la source de la diffusion, un processus de réception est créé, qui stocke les messages reçus pour cet arbre dans une mémoire tampon,
- si l'agent n'est pas une feuille de l'arbre, un processus léger d'émission est créé pour chaque fils du nœud dans l'arbre : c'est lui qui se charge de transmettre les messages stockés dans la mémoire tampon à ce fils.

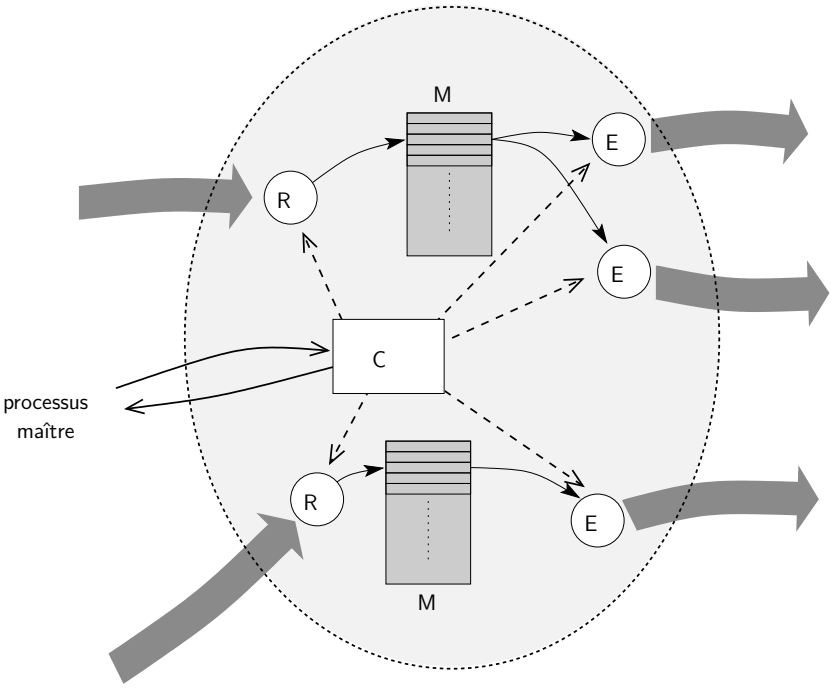


FIG. 5.1 – Structure d'un agent communiquant, pour deux arbres de diffusions. Les processus légers E,R et C sont respectivement les émetteurs, récepteurs et le coordinateur qui reçoit les consignes du maître et lui renvoie le débit de réception. M désigne la mémoire tampon pour chaque arbre de diffusion.

La figure 5.1 représente l'architecture du programme obtenu.

Une mémoire tampon est donc créée pour stocker les messages reçus qui doivent être retransmis. Cette mémoire est de taille fixe (capable de contenir 20 messages dans nos expériences) et elle est utilisée de façon circulaire (du type «round-robin»). Avant de réutiliser une case de cette mémoire pour stocker un nouveau message arrivé, il faut s'assurer que tous les agents émetteurs ont bien réémis le message qui s'y trouvait. Ceci est fait à l'aide de sémaphores, en utilisant un mécanisme classique de producteurs/consommateurs.

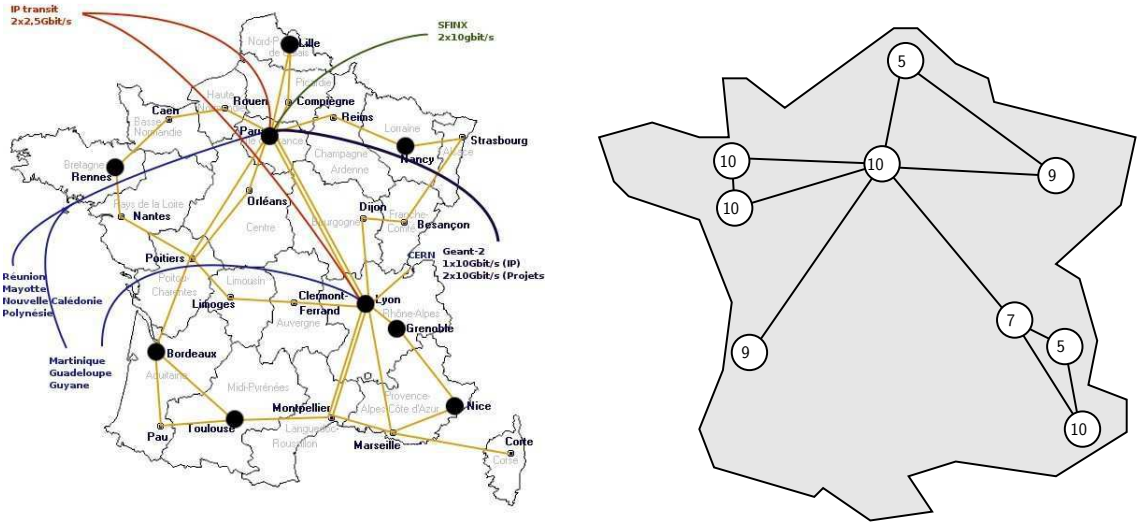
Plusieurs arbres peuvent être utilisés simultanément. Pour éviter que les messages de différents arbres ne s'interchangent, chaque arbre utilise un port de communication TCP différent : lors de l'établissement des connexions, un processus récepteur qui vient d'être créé signifie au maître son adresse IP et le port sur lequel il écoute. Ces informations sont ensuite transmises au processus émetteur qui doit lui envoyer les messages de cet arbre, qui peut alors initialiser la connexion. De plus, les messages sont étiquetés avec leur numéro dans la série et l'arbre sur lequel ils sont diffusés, ce qui permet de vérifier la cohérence de ces informations à chaque réception d'un nouveau message. Lorsque plusieurs arbres sont créés, le nombre de messages à diffuser sur chaque arbre est calculé pour être proportionnel au débit qu'on désire lui affecter.

On fournit au processus maître un fichier de description de la plate-forme à utiliser, qui lui permet de contacter chaque agent, et un fichier détaillant les tests à effectuer. C'est lui qui exécute les algorithmes des différentes heuristiques présentées ci-dessus, qui détermine et distribue aux agents les consignes permettant d'effectuer la stratégie voulue. Pour chaque diffusion,

nous utilisons une série de 10 000 messages de taille 20 000 octets, ce qui conduit à des tests de quelques minutes. Ces messages sont générés de façon aléatoire par le nœud qui se trouve à la source de la diffusion. À la fin d'une opération de diffusion, le maître reçoit le débit de réception de chaque nœud et calcule le minimum de ces débits, qui est considéré comme le débit de la diffusion. Il est ensuite possible de redémarrer une autre opération de diffusion, avec d'autres paramètres, sans déployer à nouveau toute l'application.

Topologie de tests

Le projet Grid5000 vise à construire une grille de calcul française pour la recherche, en rassemblant des machines localisées dans neuf sites en France. Le nombre de processeurs, qui doit atteindre 5000 à terme, est à l'heure actuel de près de 2500. Les sites et leur réseau d'interconnexion par Renater 4 sont représentées sur la figure 5.2(a). Dans la période où nous avons conduit nos simulations, nous avons pu avoir accès à 75 machines, réparties sur huit sites (Rennes, Nancy, Orsay, Lyon, Grenoble, Lille, Sophia-Antipolis et Bordeaux). Ces machines sont réparties comme illustré sur la figure 5.2(b). Chaque ensemble de machines dans un même site est modélisé comme un graphe complètement connecté, et nous ajoutons des liens distants en nous inspirant de la topologie réseau de Grid5000. Le choix de ces liens longue distance est un peu arbitraire, car rien ne nous empêcherait d'ouvrir une connexion entre par exemple Rennes et Sophia-Antipolis, mais nous tentons de prendre en compte les informations sur la topologie du réseau de la figure 5.2(a). Le graphe d'interconnexion de ces sites utilisé par la suite est décrit à la figure 5.2(b).



(a) Le réseau Renater 4 reliant les sites de Grid5000.

(b) Topologie utilisée pour les tests

FIG. 5.2 – Plate-forme de test, avec le nombre de machines utilisées sur chaque site. Les deux cercles pour Rennes représentent deux grappes de calcul situées sur un même site géographique, mais considérées comme deux sites distincts.

Pour pouvoir utiliser les différentes stratégies de diffusion, nous devons instancier les paramètres de la plate-forme. Pour le modèle de communication un-port, nous devons par exemple connaître le coût d'envoi d'un message de taille unité sur chaque arête, c'est-à-dire mesurer ces

valeurs sur la plate-forme. Ces mesures sont effectuées par l'application elle-même, en utilisant un petit arbre de diffusion : pour tester la capacité de l'arête (i, j), nous créons un arbre avec $P_{\mathsf{source}} = P_i$ et comportant une seule arête (i, j). Une fois le débit de réception mesuré et renvoyé au maître par P_j, le maître connaît la bande-passante de l'arête (i, j). Ainsi que nous l'avons énoncé dans la présentation du modèle multi-port (partie 5.1), nous supposons que le coût de transmission d'un message de taille unité est l'inverse de la bande-passante.

Pour le modèle multi-port borné, le maître doit connaître la bande-passante de chaque lien, déjà mesurée, et la bande-passante total en sortie et en entrée de chaque processeur. Pour calculer la bande-passante en sortie d'un processeur P_i, nous créons également des petits arbres de diffusion, enracinés en P_i. Un arbre est créé pour chaque arête sortante (i, j) de P_i, qui ne comporte que cette arête. Nous mesurons le débit total en sortie de P_i et considérons qu'il s'agit de la bande-passante $B_{\text{out}}(P_i)$. Remarquons qu'il aurait été tentant d'utiliser un seul arbre de diffusion, dont la source serait P_i et qui aurait comme arêtes toutes les arêtes sortantes de P_i dans G. Cependant, à cause de la taille limitée de la mémoire tampon contenant les messages à envoyer, toutes les connexions sortantes verraient rapidement leur débit aligné sur le débit minimal d'une connexion. Nous préférons donc créer des arbres différents pour chaque arête, qui ont chacun leur propre mémoire tampon, et ne sont donc pas limités par les autres arêtes. Nous procédons de même pour la bande-passante $B_{\text{in}}(P_i)$ en entrée d'un processeur P_i.

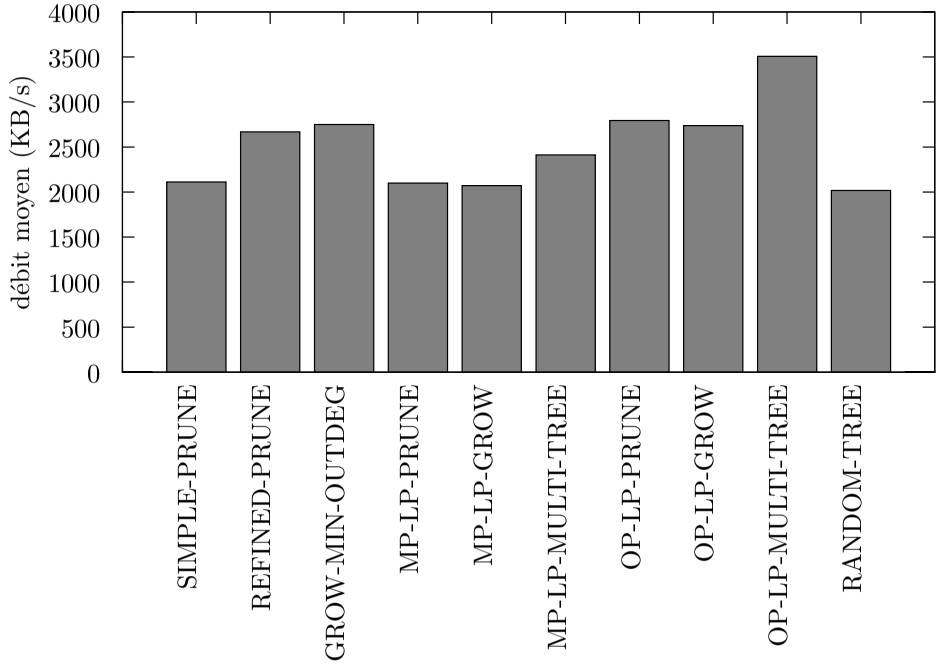
Toutes ces mesures préliminaires de bande-passante sont effectuées en diffusant 1000 messages de tailles 20 000 octets, ce qui prend quelques secondes pour chaque test. On peut se permettre d'utiliser un nombre de messages plus petit que pour mesurer le débit d'une diffusion, car l'obtention du régime permanent est beaucoup plus rapide sur une seule arête, ou un petit nombre d'arêtes entrant ou sortant d'un nœud, que sur un ou plusieurs arbres de diffusion couvrant tous les nœuds.

5.3.2 Résultats et discussion

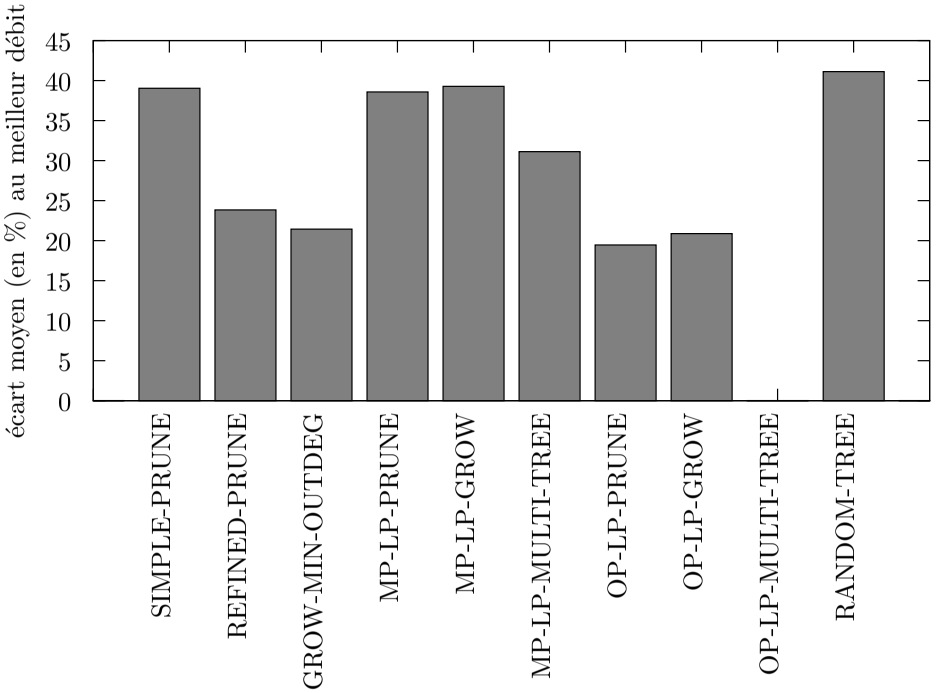
Nous présentons ici les résultats obtenus. La figure 5.3(a) présente la moyenne des débits obtenus par chacune des heuristiques et la figure 5.3(b) présente l'écart moyen à la meilleure heuristique pour chaque test. En plus des heuristiques précédemment développées, nous incluons une heuristique de création aléatoire d'arbre (nommée RANDOM-TREE) : celle-ci supprime aléatoirement des arêtes du graphe de plate-forme jusqu'à obtenir un arbre de diffusion.

Nous pouvons faire les observations suivantes :

- L'heuristique qui réalise les meilleurs résultats est celle qui construit un ensemble pondéré d'arbres, pour modèle un-port bidirectionnel (OP-LP-MULTI-TREE). Celle ci réalise la meilleure performance dans tous les cas.
- Les heuristiques créant un seul arbre de diffusion à partir de la solution du programme linéaire en un-port (OP-LP-PRUNE et OP-LP-GROW) sont les plus performantes parmi les heuristiques à un seul arbre, avec environ 20% d'écart par rapport à OP-LP-MULTI-TREE, suivies de très près par les heuristiques GROW-MIN-OUTDEG (21%) et REFINED-PRUNE (23%).
- Les heuristiques utilisant le résultat du programme linéaire sous le modèle multi-port borné présentent d'assez mauvais résultats : 31% d'écart à l'optimal pour la décomposition en un ensemble d'arbres pondérés (MP-LP-MULTI-TREE) et environ 39% d'écart pour les heuristiques à un seul arbre (MP-LP-PRUNE et MP-LP-GROW).



(a) Moyenne des débits



(b) Écart relatif à la meilleure solution

FIG. 5.3 – Résultats des expérimentations sur Grid5000.

– Ces deux dernières heuristiques, ainsi que la plus simple (SIMPLE-PRUNE) donnent des résultats à peine meilleurs que la création aléatoire d'un arbre de diffusion, qui réalise 41% d'écart à l'optimal.

On peut tout d'abord être surpris par la relativement bonne performance de la création d'arbre aléatoire (heuristique RANDOM). Cependant cette heuristique bénéficie de la topologie que nous avons construite : comme nous avons créé des liens entre différents sites uniquement aux endroits qui nous paraissaient opportuns en fonction de la topologie du réseau, cette heuristique ne peut pas prendre de «trop mauvaises» décisions : tout au plus pourra-t-elle entraîner une congestion sur quelques liens ou en sortie de quelques processeurs. D'autre part, elle utilise notre mécanisme d'agent distribués et de pipelines de messages et peut donc difficilement être considérée comme une façon totalement naïve de procéder à la diffusion d'une grande quantité de données, malgré sa dénomination.

D'autre part, on voit que les heuristiques fondées sur le modèle multi-port borné se comportent très mal. Ceci peut à notre avis s'expliquer par deux considérations :

– Notre mesure de la bande-passante en entrée ou en sortie d'un processeur n'est sans doute pas très précise : alors que d'un test à l'autre, la mesure de la bande-passante sur des liens distants ou locaux varie peu, la bande-passante en entrée ou en sortie d'un processeur présente une plus grande variabilité.
– Le modèle multi-port n'est sans doute pas très adapté pour la modélisation de plusieurs connexions TCP initiées par différents processus depuis un même nœud. Nous avons par exemple remarqué que lorsqu'un processeur P_i possédait un lien sortant local avec une forte bande-passante B et un lien longue-distance avec une bande-passante plus faible b (ces deux bandes-passantes étant mesurées depuis P_i), si les deux liens sont utilisés en même temps (lors de la mesure de la bande-passante en sortie de P_i), le résultat obtenu est très inférieur à B. Cependant, le modèle multi-port borné stipule que $B_{\text{out}}(P_i)$ doit être au moins égal à B. Il semblerait donc que le modèle un-port prenne mieux en compte le temps d'occupation du processeur P_i par une connexion de faible bande-passante et son influence sur les autres connexions.

Donc bien qu'on puisse penser que le modèle multi-port borné est le plus adapté pour modéliser plusieurs transferts simultanés empruntant des connexions TCP, le modèle un port décrit mieux le comportement obtenu, supposant une sérialisation inévitable des communications. Ceci justifie a posteriori notre étude des communications collectives dans ce modèle.

5.3.3 Poursuite des expérimentations

On peut remarquer que les débits obtenus sont de l'ordre de 4 ou 5 Mo/s. Ce débit peut paraître faible en regard de la capacité du réseau d'interconnexion de Grid5000, mais c'est en moyenne l'ordre de grandeur du débit obtenu pour un transfert utilisant une seule connexion TCP entre deux sites distants. Cette limitation est principalement due à la taille de la fenêtre de congestion de TCP, c'est-à-dire la fenêtre contenant des messages qui ont été envoyés mais dont l'acquittement n'est pas encore parvenu ; le débit obtenu correspond à la limitation $W_{\max}/RTT$ bien connue de TCP [82], avec une taille de fenêtre de congestion $W_{\max}$ de l'ordre de 20Ko, et un temps d'aller-retour (*round trip time*, RTT) de quelques millisecondes, ce que l'on observe entre deux sites.

Comme nous utilisons une connexion TCP pour une communication entre deux agents, nous ne pouvons espérer dépasser cet ordre de grandeur. Nous sommes actuellement en train

de modifier notre implantation pour pouvoir dépasser cette limitation. Deux solutions sont envisageables :

- D'un point de vue système, on envisage de reconfigurer la taille de la fenêtre de congestion de TCP pour l'adapter à la capacité des liens de communication ; ceci nécessite de disposer de droits particuliers sur les machines utilisées, afin de pouvoir ajuster des paramètres que seul l'administrateur des machines peut habituellement modifier. Bien que cela soit possible sur Grid5000, en déployant une image système particulière, peu de plates-formes offrent autant de liberté. Enfin, la configuration de TCP est très dépendante du système utilisé.
- Plutôt que modifier la configuration TCP, nous pouvons utiliser plusieurs connexions simultanées pour atteindre la capacité réellement disponible sur un lien de communication. En effet, plusieurs connexions peuvent coexister sur un même lien de communication sans entraîner de congestion, tant que leur débit total ne dépasse pas le débit offert par le réseau ; nous développerons un tel modèle dans le chapitre 7, pour l'exécution de tâches divisibles sur une grille de calcul. On peut donc utiliser plusieurs connexions concurrentes pour un même transfert. Nous sommes donc en train d'adapter notre implantation pour nous rapprocher du modèle décrit dans la section 7.2.

On peut souhaiter qu'à terme, la configuration de TCP sur les plates-formes distribuées sera effectuée par les administrateurs de la plate-forme, ou bien que d'autres protocoles que TCP plus adaptés à ces réseaux seront utilisés, afin qu'il soit possible d'obtenir un débit proche du débit physique annoncé sans recourir à de telles stratégies.

Conclusion

Dans cette partie, nous avons présenté les expérimentations que nous avons menées pour évaluer l'intérêt de nos stratégies de diffusion pipelinée. Nous avons introduit un nouveau modèle de communication, le modèle multi-port borné et montré qu'il pouvait être utilisé dans notre étude des communications collectives. Les résultats des expérimentations présentées ici montrent que malgré tout, le modèle un-port bidirectionnel convient mieux à l'élaboration de stratégies de communications collectives, et parmi les stratégies fondées sur ce modèle, la stratégie décomposant la solution du programme linéaire donne de meilleurs résultats, comme attendu.

Ces résultats pourrait être complétés par de nombreuses autres expérimentations, et il serait sans doute intéressant d'intégrer quelques unes de ces stratégies à une bibliothèque de communication pour la grille, afin de comparer ces heuristiques à des bibliothèques existantes comme MagPIe [69] ou ECO [78].

Seconde partie

Ordonnancement d'applications multiples

Chapitre 6

Collections de tâches indépendantes en maître-esclaves

6.1 Introduction

Dans ce chapitre, nous nous intéressons à l'exécution concurrente de plusieurs applications sur une plate-forme de calcul distribuée. Ces applications se partagent les ressources de calcul et de communication disponibles, et nous voulons assurer une exécution efficace et équitable. Nous visons des plates-formes de type maître-esclaves : un processeur «maître» distribue des tâches à différents processeurs «esclaves» travaillant pour lui. Une plate-forme de ce type est souvent modélisée par un graphe en étoile : le maître, au centre, est relié à tous les autres processeurs. Nous utilisons un modèle un peu plus général, qui convient mieux aux plates-formes à grande échelle : les esclaves avec qui le maître communique peuvent eux-mêmes déléguer du travail à d'autres esclaves. Le graphe de la plate-forme est donc un arbre.

Nous considérons que chaque application est constituée d'un grand nombre de tâches de même taille et tous les fichiers de données associées à ces tâches sont initialement présents à la racine de l'arbre. Ceci correspond à un ordonnanceur centralisé qui distribue des tâches, comme le fait BOINC [103], qui distribue des tâches pour différentes applications telles que SETI@home, ClimatePrediction.NET et Einstein@Home. Ces applications peuvent être de nature différente, comme du traitement de fichiers, de l'analyse d'image, ou des opérations sur des matrices. Chaque application a donc deux paramètres caractéristiques, une taille de fichier d'entrée et une quantité de calcul par fichier. À chaque application est ainsi associé un rapport coût de communication sur coût de calcul, identique pour toutes les tâches d'une même application, mais variable d'une application à l'autre. Ce rapport se révèlera un paramètre important pour l'ordonnancement.

L'ordonnancement de tâches indépendantes a déjà été étudié, en particulier pour une seule application, sur des plate-forme maître-esclaves : voir par exemple [92, 91] pour une approche fondée sur l'étude d'un flot dans le réseau (utilisant un modèle multiple-port non borné), ou [60] pour une stratégie adaptative. D'autres travaux [56, 98] présentent des méthodes pour faciliter l'implantation d'ordonnanceurs maître-esclaves pour les tâches indépendantes. Enfin, pour le même modèle de plate-forme que nous utilisons, une stratégie optimale simple a été décrite dans [17, 10] pour l'ordonnancement d'un seul type de tâches indépendantes ; il a également été montré [70] qu'une implantation dynamique de cette stratégie donnait un débit proche de

l'optimal, en utilisant une mémoire de taille limitée. Nous la décrirons plus loin lorsque nous l'utiliserons pour construire des heuristiques décentralisées.

Nous étudions ce problème dans la perspective du régime permanent et nous intéressons donc à maximiser le débit de chaque application (le nombre de tâches effectuées par unité de temps), plutôt que de considérer le temps total d'exécution de toutes les tâches. D'autre part, nous considérons le cas général où les applications sont munies de priorités $w^{(k)}$, qui quantifient leur importance relative : si une application $A^{(1)}$ a une priorité $w^{(1)} = 3$ et une application $A^{(2)}$ a une priorité $w^{(2)} = 1$, on cherchera à traiter trois fois plus de tâches de $A^{(1)}$ que de $A^{(2)}$.

Pour chaque application $A^{(k)}$, on appelle $\nu^{(k)}(t)$ le nombre de tâches de cette application effectuées en temps t. À tout instant t, on peut définir le débit de l'application $A^{(k)}$: $\alpha^{(k)}(t) = \nu^{(k)}(t)/t$. En régime permanent, nous considérerons que $\alpha^{(k)}$ est le nombre moyen de tâches effectuées par unité de temps. Afin de garantir une exécution équitable, nous cherchons à maximiser le minimum des débits pondérés : $\min_k \alpha^{(k)}(t)/w^{(k)}$. Cette fonction objective correspond à l'équité max-min [25, 79] entre les différentes applications, munies de coefficients $1/w^{(k)}$.

Nous allons nous intéresser à l'élaboration d'ordonnanceurs centralisés et distribués. Nous verrons que dans le cas centralisé, lorsque nous pouvons rassembler une information complète et fiable sur la plate-forme sur le maître, nous pouvons calculer le débit optimal. Cependant, sur des plates-formes à grande échelle, en particulier lorsque les ressources disponibles évoluent, un ordonnancement centralisé n'est pas souhaitable. Nous étudions donc des ordonnanceurs locaux, s'appuyant uniquement sur des informations locales (vitesse des processeurs et capacités des liens). Nous cherchons à savoir si de tels ordonnanceurs décentralisés peuvent atteindre le débit optimal. Nous proposons plusieurs heuristiques décentralisées, et nous les comparons par simulation à un ordonnanceur centralisé.

6.2 Modélisation de la plate-forme et des applications

Plate-forme La plate-forme est modélisée par un graphe $G = (V, E)$; comme nous considérons une plate-forme maître-esclaves, nous supposons que G est un arbre. La racine de l'arbre est le processeur «maître» P_{source} et les autres nœuds de l'arbre sont des processeurs «esclaves», $P_1, \dots P_p$. Pour plus de facilité, on notera $P_{p(i)}$ le processeur père de P_i dans l'arbre. Comme précédemment, le coût d'une communication de taille unité sur le lien (P_i, P_j) est noté $c_{i,j}$ et nous supposons un modèle de coût linéaire pour les communications et les calculs. La communication d'un message de taille L entre $P_{p(i)}$ et P_i prendra donc un temps $L \times c_{p(i),i}$. Nous nous plaçons dans le cadre du modèle un-port bidirectionnel : un processeur peut simultanément effectuer une réception et une émission. On considère un modèle de coûts de calcul sans affinités entre processeurs et tâches : on note z_i le temps nécessaire à l'exécution d'une tâche de taille unité sur le processeur P_i.

Applications Nous considérons K applications $A^{(1)}, \dots, A^{(K)}$. Initialement, la racine P_{source} possède toutes les données nécessaires aux applications. Comme précisé plus haut, chaque application $A^{(k)}$ est munie d'une priorité $w^{(k)}$ et est composée d'un ensemble de tâches indépendantes, de même taille. On considérera que ces ensembles de tâches sont de taille suffisamment grande pour atteindre un régime permanent, et on ne prend pas en compte le nombre total de tâches pour chaque application. Une tâche de l'application $A^{(k)}$ sera appelée tâche de type k ;

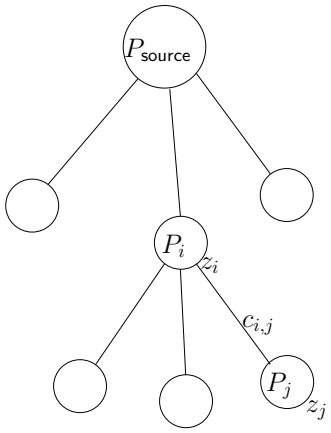


FIG. 6.1 – Plate-forme cible.

elle nécessite $\mathsf{flops}^{(k)}$ unités de calcul, de telle sorte que l'exécution d'une tâche de type k sur le processeur P_i nécessite $\mathsf{flops}^{(k)} \times z_i$ unités de temps. De plus, les fichiers de données nécessaires à une tâche de type k sont de taille $\mathsf{size}^{(k)}$, de sorte qu'il faut $\mathsf{size}^{(k)} \times c_i$ unités de temps pour que $P_{p(i)}$ envoie une tâche de type k à P_i.

Nous supposons que seuls les transferts des données envoyées par le maître sont de taille conséquente et que les résultats renvoyés par les esclaves sont de taille négligeable. Nous étudierons dans la partie 6.3.4 la généralisation avec messages de retour. Le rapport communication sur calcul pour l'application $A^{(k)}$ est $\mathsf{size}^{(k)}/\mathsf{flops}^{(k)}$.

Fonction objective En notant $\nu^{(k)}(t)$ le nombre de tâches de type k effectuée en temps t, notre objectif est de maximiser

$$\limsup_{t\to\infty} \min_k \left\{ \frac{\nu^{(k)}(t)}{w^{(k)} \cdot t} \right\}$$

Cependant, comme nous nous intéressons au régime permanent, nous définissons le débit moyen $\alpha^{(k)}$ de tâches de type k effectuées par unité de temps et nous définissons la fonction objective suivante :

$$\text{MAXIMISER} \ \min_k \left\{ \frac{\alpha^{(k)}}{w^{(k)}} \right\}$$

Nous appelons *débit équitable* le minimum des débits pondérés des applications.

6.3 Calcul de la solution optimale

Dans cette partie, nous introduisons une méthode fondée sur la programmation linéaire, pour calculer le débit optimal du régime permanent et reconstruire un ordonnancement qui réalise ce débit. Nous proposons ensuite une caractérisation de la solution optimale pour des plates-formes en forme d'étoile.

6.3.1 Programme linéaire

Notre approche s'inspire de la méthode introduite pour calculer le débit optimal d'une opération de distribution de données. Ce qu'on cherche à faire ici se résume en effet à distribuer des fichiers, de tailles différentes et à effectuer les calculs correspondants. La différence avec la distribution de données présentée précédemment est que nous ne savons pas a priori à quels processeurs une application doit être envoyée, et nous devons prendre en compte des tailles de fichier différentes.

Nous appelons $\mathsf{send}^{(k)}_{P_i \to P_j}$ le nombre de tâches de type k transmises sur le lien (P_i, P_j) en une unité de temps, pour un fils P_j de P_i ($P_i = P_{p(j)}$). De plus, nous appelons $\alpha_i^{(k)}$ le nombre moyen de tâches de type k traitées sur le processeur P_i en une unité de temps. Nous pouvons écrire les contraintes suivantes :

– Comme l'exécution d'une tâche de type k sur P_i prend un temps $\mathsf{flops}^{(k)} \times z_i$, on peut borner la quantité de calcul effectuée sur le processeur P_i en une unité de temps :

$$\sum_k \alpha_i^{(k)} \times \mathsf{flops}^{(k)} \times z_i \leqslant 1$$

– Un lien (P_i, P_j) est occupé pendant $\mathsf{send}^{(k)}_{P_i \to P_j} \times \mathsf{size}^{(k)} \times c_{i,j}$ unités de temps pour envoyer des tâches de type k. La contrainte un-port en sortie du processeur P_i s'écrit donc

$$\sum_{(i,j) \in E} \sum_k \mathsf{send}^{(k)}_{P_i \to P_j} \times \mathsf{size}^{(k)} \times c_{i,j} \leqslant 1$$

– Au nœud P_i, il y a conservation des tâches de type k : celles qui sont reçues par le père sont consommées sur place ou envoyées à un processeurs fils. En notant $P_{p(i)}$ le processeur père de P_i dans l'arbre, on a :

$$\mathsf{send}^{(k)}_{P_{p(i)} \to P_i} = \alpha_i^{(k)} + \sum_{(i,j) \in E} \mathsf{send}^{(k)}_{P_i \to P_j}$$

– Enfin, on peut calculer le débit d'une application en fonction du nombre de tâches de cette application effectuées sur chaque processeur :

$$\alpha^{(k)} = \sum_{P_i \in V} \alpha_i^{(k)}$$

En rassemblant ces contraintes, on obtient le programme linéaire suivant :

$$
\begin{array}{l}
\text{MAXIMISER } \rho_{\text{opt}}, \\
\text{SOUS LES CONTRAINTES}
\end{array}
\left\{
\begin{array}{lll}
\text{(6.1a)} & \forall P_i \in V & \displaystyle\sum_k \alpha_i^{(k)} \times \mathsf{flops}^{(k)} \times z_i \leqslant 1 \\
\text{(6.1b)} & \forall P_i \in V & \displaystyle\sum_{(i,j)\in E} \sum_k \mathsf{send}^{(k)}_{P_i \to P_j} \times \mathsf{size}^{(k)} \times c_{i,j} \leqslant 1 \\
\text{(6.1c)} & \forall P_i \in V,\ \forall A^{(k)} & \displaystyle\mathsf{send}^{(k)}_{P_{p(i)} \to P_i} = \alpha_i^{(k)} + \sum_{(i,j)\in E} \mathsf{send}^{(k)}_{P_i \to P_j} \\
\text{(6.1d)} & \forall A^{(k)} & \displaystyle\alpha^{(k)} = \sum_{P_i \in V} \alpha_i^{(k)} \\
\text{(6.1e)} & \forall A^{(k)} & \displaystyle\rho_{\text{opt}} \leqslant \frac{\alpha^{(k)}}{w^{(k)}} \\
\text{(6.1f)} & \forall P_i \in V,\ \forall A^{(k)} & \alpha_k^{(i)} \geqslant 0 \\
 & \forall (P_i, P_j) \in E,\ \forall A^{(k)} & \mathsf{send}^{(k)}_{P_i \to P_j} \geqslant 0
\end{array}
\right. \tag{6.1}
$$

On peut remarquer qu'avec la contrainte (6.1e), la fonction objective est équivalent à maximiser $\min_k \left\{ \frac{\alpha^{(k)}}{w^{(k)}} \right\}$, comme écrit précédemment.

Comme pour les primitives de communications collectives, nous montrons que ce programme linéaire définit une borne supérieure sur le débit réalisable.

Théorème 6.1. *Le programme linéaire 6.1 donne une borne supérieure pour le débit équitable d'un ordonnancement pour le problème de collections de tâches indépendantes sur la plate-forme G.*

Démonstration. Considérons un ordonnancement quelconque pour ce problème. Nous appelons $n_i^{(k)}(t)$ le nombre de tâches de type k effectuées sur le processeur P_i pendant les t premières unités de temps et $S^{(k)}_{P_i \to P_j}(t)$ le nombre de tâches de type k envoyées sur l'arête (P_i, P_j) dans cet intervalle de temps. Parmi ces tâches, on considère les $S'^{(k)}_{P_i \to P_j}(t)$ tâches qui ont ensuite été calculées sur un des processeurs : en effet, il est probable qu'au temps t, des tâches qui ont été envoyées par le maître sont encore stockées dans la mémoire d'un processeur et n'ont pas été calculées. On a donc $S'^{(k)}_{P_i \to P_j}(t) \leqslant S^{(k)}_{P_i \to P_j}(t)$. Ces quantités vérifient les contraintes suivantes :

– Le processeur P_i ne peut pas calculer pendant plus de t unités de temps :

$$\forall P_i \in V, \quad \sum_k n_i^{(k)}(t) \times \mathsf{flops}^{(k)} \times z_i \leqslant t$$

– Le port de communication en sortie de P_i ne peut être occupé pendant plus de t unités de temps :

$$\forall P_i \in V, \quad \sum_{(i,j)\in E} \sum_k S^{(k)}_{P_i \to P_j}(t) \times \mathsf{size}^{(k)} \times c_{i,j} \leqslant t$$

En particulier, comme $S' \leqslant S$, on a

$$\forall P_i \in V, \quad \sum_{(i,j)\in E} \sum_k {S'}^{(k)}_{P_i \to P_j}(t) \times \mathsf{size}^{(k)} \times c_{i,j} \leqslant t$$

– En P_i, les tâches reçues et comptabilisées dans S' sont soit effectuées sur places, soit envoyées à un fils et comptabilisées dans la variables S' correspondante :

$$\forall P_i \in V, \forall A^{(k)} \quad {S'}^{(k)}_{P_{p(i)} \to P_i}(t) = n_i^{(k)}(t) + \sum_{(i,j)\in E} {S'}^{(k)}_{P_i \to P_j}(t)$$

– Le nombre total de tâches de type k effectuées en temps t est :

$$\forall A^{(k)}, \quad \nu^{(k)}(t) = \sum_{P_i \in V} n_i^{(k)}(t)$$

Les quantités ${S'}^{(k)}_{P_i \to P_j}(t)/t$ et $n_i^{(k)}(t)/t$ définissent une solution du programme linéaire, donc en particulier

$$\min_k \frac{\nu^{(k)}(t)}{t \cdot w^{(k)}} \leqslant \rho_{\text{opt}}$$

Comme on peut faire se raisonnement pour toute valeur de t, on a

$$\limsup_{t\to\infty} \min_k \frac{\nu^{(k)}(t)}{t \cdot w^{(k)}} \leqslant \rho_{\text{opt}}$$

Donc le débit équitable d'un ordonnancement est borné par ρ_{opt}. ■

6.3.2 Construction d'un ordonnancement périodique

On pourrait s'appuyer sur la méthode développée dans les chapitres précédents pour reconstruire un ordonnancement périodique, puisque notre problème est proche de la distribution de données. Cependant, en utilisant la structure en arbre de la plate-forme, on peut obtenir facilement une description simple de l'ordonnancement. En effet, nous n'avons pas à préciser l'ordonnancement précis des communications. Puisque qu'un processeur P_i ne reçoit des données que depuis son père $P_{p(i)}$, on peut laisser le processeur $P_{p(i)}$ choisir à quel moment il veut envoyer des données à P_i. Il suffit donc de décrire globalement quelle quantité de données doit être envoyée à chaque fils.

Une fois obtenue une solution optimale $(\mathsf{send}^{(k)}_{P_i \to P_j}, \alpha_i^{(k)})$ réalisant un débit équitable ρ_{opt}, on écrit $\mathsf{send}^{(k)}_{P_i \to P_j} = \frac{\alpha(i,j,k)}{\beta(i,j,k)}$ et $\alpha_k^{(i)} = \frac{\gamma(i,k)}{\delta(i,k)}$ sous forme irréductible, puis on calcule le plus petit commun multiple des dénominateurs :

$$L = \text{ppcm}\left\{ \underset{(i,j,k)}{\text{ppcm}}\, \beta(i,j,k), \underset{(i,k)}{\text{ppcm}}\, \delta(i,k) \right\}.$$

L est la durée d'une période. Pendant une période, un processeur reçoit $L \times \mathsf{send}^{(k)}_{P_{p(i)} \to P_i}$ tâches de l'application k depuis son père, calcule $L \times \alpha_k^{(i)}$ tâches pour cette application, et envoie $L \times \mathsf{send}^{(k)}_{P_i \to P_j}$ tâches de cette application à chacun de ses fils P_j. Les tâches calculées

Pour étape $= 1, 2, \ldots$ **:**
  Si étape $\geqslant h(P_i) - 1$ et $P_i \neq P_{\mathsf{source}}$ **Alors**
    Pour $k = 1, \ldots, k$ **:**
      Recevoir $L \times \mathsf{send}^{(k)}_{P_{p(i)} \to P_i}$ tâches de type k, les placer dans mem_1.
  Si étape $\geqslant h(P_i)$ **Alors**
    Pour $k = 1, \ldots, k$ **:**
      Traiter $L \times \alpha^{(i)}_k$ tâches de type k de mem_2.
      Pour chaque fils P_j de P_i **:**
        Envoyer $L \times \mathsf{send}^{(k)}_{P_i \to P_j}$ tâches de type k de mem_2 au processeur P_j.
  $\mathsf{mem}_2 \leftarrow \mathsf{mem}_1$.

Algorithme 6.1: Ordonnancement périodique optimal.

ou envoyées lors d'une période correspondent aux tâches reçues pendant la période précédente. On note $h(P_i)$ la hauteur dans l'arbre du processeur P_i telle que $h(P_{\mathsf{source}}) = 1$. Formellement, chaque processeur P_i exécute l'algorithme 6.1.

Pour prendre en compte un nombre de tâches fini pour chaque application, il suffit de modifier légèrement dans le code précédent en remplaçant «recevoir N tâches» par «recevoir au plus N tâches», «traiter N tâches» par «traiter au plus N tâches», etc. On s'arrête alors lorsqu'il n'y a plus aucune tâche dans le système. En régime permanent, cet ordonnancement atteint bien le débit optimal ρ_{opt}.

On peut noter que l'on suppose ici qu'un processeur P_i a une mémoire capable de contenir deux fois la quantité de messages reçus en une période, ce qui peut représenter une grande quantité de données. La mémoire nécessaire à cet ordonnancement est donc potentiellement très grande, ce qui peut le rendre inutilisable en pratique. Nous verrons plus tard comment éviter ce problème.

6.3.3 Caractérisation pour une plate-forme en étoile

Nous nous intéressons ici à un cas particulier, où la plate-forme est un arbre à un seul niveau, encore appelé étoile : les fils de P_{source} dans l'arbre n'ont eux-mêmes pas de fils. Dans ce cas particulier, nous pouvons mettre en évidence une structure particulière de l'ordonnancement optimal : les applications ayant un rapport communication sur calcul important doivent être effectuées sur les processeurs qui sont reliés au maître par un lien de communication efficace, c'est-à-dire avec un faible coût de communication c_i. On note $r^{(k)} = \mathsf{size}^{(k)}/\mathsf{flops}^{(k)}$ le rapport communication sur calcul pour une application $A^{(k)}$. Les applications sont alors réparties par tranches, comme représenté sur la figure 6.2 : l'application possédant le rapport $r^{(k)}$ le plus élevé est exécutée sur la tranche de processeurs avec les liens de communication les plus efficaces, etc.

Pour simplifier les notations, on note c_i le coût d'une communication sur le lien $(P_{\mathsf{source}}, P_i)$, c'est-à-dire $c_i = c_{(P_{\mathsf{source}}, P_i)}$. En outre, nous considérons que les calculs effectués sur le maître sont en fait exécutés sur un nouvel esclave P_0, de coût de communication nul : $c_0 = 0$. Le théorème suivant montre la structure particulière d'un ordonnancement optimal.

Théorème 6.2. *On trie et on renumérote les processeurs par coût de communication croissant, de telle sorte que* $c_0 \leqslant c_1 \leqslant \cdots \leqslant c_p$*, et les applications par rapport communication/calcul*

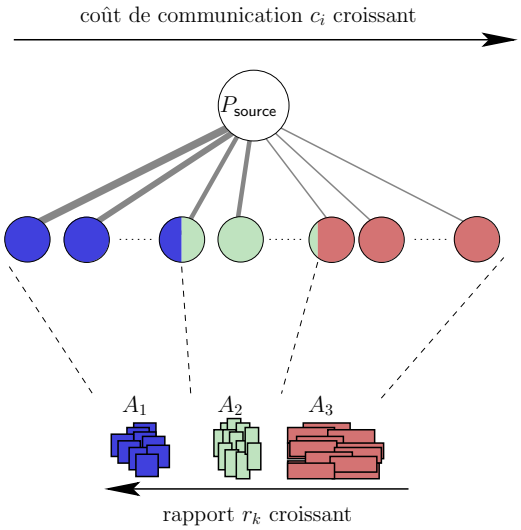


FIG. 6.2 – Répartition par tranches pour une plate-forme en étoile

décroissant : $r_1 \geqslant r_2 \geqslant \cdots \geqslant r_K$. Alors, il existe des indices $a_0, \ldots, a_K$ tels que, dans une solution optimale, seuls les processeurs $P_i, i \in [a_{k-1}, a_k]$ exécutent des tâches de l'application $A^{(k)}$.

Démonstration. Considérons un ordonnancement optimal et supposons qu'il ne respecte pas la structure voulue. Il existe donc deux applications $A^{(k)}$ et $A^{(l)}$ $(k < l)$ et deux processeurs P_i et P_j $(i < j)$ tels que :

$$\alpha_l^{(i)} > 0 \quad \alpha_k^{(j)} > 0.$$

En d'autres termes, l'application $A^{(k)}$ est exécutée sur P_j, alors que l'application $A^{(l)}$ est exécutée sur P_i. Comme $k < l$ et $i < j$, on a :

$$r^{(k)} \geqslant r^{(l)} \text{ i.e. } \frac{\mathsf{size}^{(k)}}{\mathsf{flops}^{(k)}} \geqslant \frac{\mathsf{size}^{(l)}}{\mathsf{flops}^{(l)}} \quad \text{et} \quad c_i \leqslant c_j$$

On construit maintenant une autre solution, fondée sur la solution décrite par α, mais dans laquelle P_i et P_j échangent quelques tâches des applications $A^{(k)}$ et $A^{(l)}$. Plus précisément, dans notre nouvelle solution notée $\widetilde{\alpha}$, β_k tâches de l'application $A^{(k)}$ sont déplacées de P_j vers P_i, et β_l tâches de l'application $A^{(l)}$ sont déplacées de P_i vers P_j. On a donc :

$$\text{en } P_i \begin{cases} \widetilde{\alpha}_k^{(i)} = \alpha_k^{(i)} + \beta_k \\ \widetilde{\alpha}_l^{(i)} = \alpha_l^{(i)} - \beta_l \end{cases} \quad \text{et en } P_j \begin{cases} \widetilde{\alpha}_k^{(j)} = \alpha_k^{(j)} - \beta_k \\ \widetilde{\alpha}_l^{(j)} = \alpha_l^{(j)} + \beta_l \end{cases}$$

Pour choisir les valeur de β_k et β_l, nous faisons en sorte de ne pas modifier le temps de calcul des processeurs P_i et P_j, en assurant que β_k tâches de $A^{(k)}$ représentent la même quantité de calcul que β_l tâches de $A^{(l)}$:

$$\beta_k \times \mathsf{flops}^{(k)} = \beta_l \times \mathsf{flops}^{(l)}$$

Ainsi le temps nécessaire au processeur P_i pour effectuer les calculs correspondants aux applications $A^{(k)}$ et $A^{(l)}$ dans le nouvel ordonnancement est

$$\begin{aligned}\left(\widetilde{\alpha}_k^{(i)} \times \mathsf{flops}^{(k)} + \widetilde{\alpha}_l^{(i)} \times \mathsf{flops}^{(l)}\right) \times z_i &= \left((\alpha_k^{(i)} + \beta_k) \times \mathsf{flops}^{(k)} + (\alpha_l^{(i)} - \beta_l) \times \mathsf{flops}^{(l)}\right) \times z_i \\ &= \left(\alpha_k^{(i)} \times \mathsf{flops}^{(k)} + \alpha_l^{(i)} \times \mathsf{flops}^{(l)}\right) \times z_i\end{aligned}$$

soit le temps passé dans l'ordonnancement original pour calculer des tâches de $A^{(k)}$ et $A^{(l)}$. Il en est de même pour P_j. Pour que la transformation soit valide, il faut que

$$0 \leqslant \beta_k \leqslant \alpha_k^{(j)} \text{ et } 0 \leqslant \beta_l \leqslant \alpha_l^{(i)}.$$

On pose donc :

$$\beta_l = \min\left\{\alpha_l^{(i)}, \frac{\mathsf{flops}^{(k)}}{\mathsf{flops}^{(l)}}\alpha_k^{(j)}\right\} \text{ et } \beta_k = \frac{\mathsf{flops}^{(l)}}{\mathsf{flops}^{(k)}}\beta_l$$

Montrons que ce nouvel ordonnancement est valide. Nous avons vu que le temps de calcul des processeurs est inchangé. Il nous reste à vérifier que le temps de communication du maître est inférieur au temps de communication du maître dans l'ordonnancement original. En notant T le temps de communication dans l'ordonnancement original et $\widetilde{T}$ dans le nouvel ordonnancement, on a :

$$\begin{aligned}\widetilde{T} &= T + \underbrace{(\beta_k \times \mathsf{size}^{(k)} - \beta_l \times \mathsf{size}^{(l)}) \times c_i}_{\text{différences pour } P_i} + \underbrace{(\beta_l \times \mathsf{size}^{(l)} - \beta_k \times \mathsf{size}^{(k)}) \times c_j}_{\text{différences pour } P_j} \\ &= T + (\beta_k \times \mathsf{size}^{(k)} - \beta_l \times \mathsf{size}^{(l)}) \times (c_i - c_j) \\ &= T + (\beta_k \times r^{(k)} \times \mathsf{flops}^{(k)} - \beta_l \times r^{(l)} \times \mathsf{flops}^{(l)}) \times (c_i - c_j) \\ &= T + \beta_k \times \mathsf{flops}^{(k)} \times (r^{(k)} - r^{(l)}) \times (c_i - c_j) \quad \text{car } \beta_k \times \mathsf{flops}^{(k)} = \beta_l \times \mathsf{flops}^{(l)}\end{aligned}$$

Comme $c_i \leqslant c_j$ et $r_k \geqslant r_l$, on a $\widetilde{T} \leqslant T$. Dans le cas où, lors du calcul de β_l par un minimum, $\beta_l = \alpha_l^{(i)}$, alors $\widetilde{\alpha}_l^{(i)} = 0$. Dans l'autre cas, $\beta_k = \alpha_k^{(j)}$ et $\widetilde{\alpha}_k^{(j)} = 0$.

Nous procédons de même pour tout couple d'applications sur tout couple de processeurs, jusqu'à ce que l'ordonnancement respecte la structure voulue, en au plus $K^2 \times p^2$ étapes. ■

Remarque Le résultat précédent caractérise la structure d'une solution optimale. On pourrait donc supposer que de nombreuses autres structures conduisent également à un débit optimal. Cependant, considérons le cas particulier d'une plate-forme où les c_i sont tous distincts deux-à-deux, de même que les r_k. Comme dans la preuve précédente, on considère un ordonnancement optimal et on suppose qu'il ne respecte pas la structure voulue. On effectue la même transformation que précédemment, en échangeant des tâches des applications $A^{(k)}$ et $A^{(l)}$ sur les processeurs P_i et P_j. Le temps d'occupation du maître dans le nouvel ordonnancement vérifie :

$$\widetilde{T} = T + \beta_k \times \mathsf{flops}^{(k)} \times (r^{(k)} - r^{(l)}) \times (c_i - c_j)$$

Comme on a supposé les c_i et le r_k tous distincts, $c_i < c_j$ et $r^{(k)} > r^{(l)}$. Comme $\beta_k > 0$, on a $\widetilde{T} < T$. On a donc construit un ordonnancement de débit optimal, mais qui utilise le port

de communication du maître pendant une durée strictement plus petite. Si, dans l'ordonnancement original, il existe un processeur qui n'est pas utilisé à plein régime, alors nous pouvons utiliser ce temps libre pour lui envoyer une petite fraction de tâches de toutes les applications et ainsi augmenter le débit total. Ceci est absurde avec l'optimalité du débit de l'ordonnancement original.

Donc dans le cas, certes particulier, d'une plate-forme en étoile où tous les c_i et les r_k sont distincts et où la ressource limitante n'est pas la puissance de calcul cumulée des processeurs, alors toute solution optimale à une structure comme celle décrite dans le théorème 6.2.

Généralisation Nous n'avons pas pu généraliser le résultat obtenu pour une plate-forme en étoile à une plate-forme quelconque. Si un processeur P_i est relié au maître avec un lien efficace, cela n'implique pas qu'il en est de même pour les descendants de P_i : en particulier ils se peut que ses fils dans l'arbre aient tous des liens de communication avec de forts coûts de communication ; dans ce conditions, envoyer uniquement des tâches à fort rapport communication/calcul à P_i serait un mauvais choix, il faut également lui envoyer des tâches de faible rapport communication/calcul, pour qu'il puisse les déléguer à ses descendants dans l'arbre.

6.3.4 Prise en compte des messages de retour

Les résultats que nous avons présentés jusqu'ici ne prennent pas en compte les messages contenant les résultats, dont on peut supposer qu'il doivent être rassemblés sur le maître. Nous présentons ici une généralisation du calcul de la solution optimale tenant compte de ces messages. Nous notons $\mathsf{size}_{\mathrm{res}}^{(k)}$ la taille d'un message de retour pour l'application $A^{(k)}$. L'exécution d'une tâche de l'application $A^{(k)}$ suppose donc l'envoi d'un message de taille $\mathsf{size}^{(k)}$ du maître jusqu'au processeur choisi pour l'exécution, le traitement d'une tâche de type $\mathsf{flops}^{(k)}$ et l'envoi d'un message de retour de taille $\mathsf{size}_{\mathrm{res}}^{(k)}$ depuis le processeur de calcul jusqu'au maître. En régime permanent, le nombre de messages de retour émanant d'un sous-arbre est égal au nombre de tâches traitées par ce sous-arbre, donc égal au nombre de messages de données reçus par ce sous-arbre. En d'autres termes, comme nous avons noté $\mathsf{send}_{P_i \to P_j}^{(k)}$ le nombre de tâches de type k reçus par le processeur P_j (et donc traitées par le sous-arbre enraciné en P_j), $\mathsf{send}_{P_i \to P_j}^{(k)}$ est également le nombre de messages de retour convoyés par l'arête (j, i). Notons que nous devons ajouter des arêtes dans E pour transmettre les messages de retour, de sorte que le graphe de plate-forme n'est plus un arbre dirigé (mais le graphe non-dirigé sous-jacent est un arbre).

Nous cherchons à adapter le programme linéaire calculant le débit optimal. Précédemment, nous nous contentions de borner les communications sortantes d'un processeur, c'est-à-dire le nombre de messages émis en direction des fils :

$$\sum_{(i,j)\in E} \sum_{k} \mathsf{send}_{P_i \to P_j}^{(k)} \times \mathsf{size}^{(k)} \times c_{i,j} \leqslant 1$$

Dans cette contrainte, nous devons maintenant également prendre en compte le temps d'envoi des résultats rassemblés en P_i et envoyés à son père $P_{p(i)}$ (au nombre de $\mathsf{send}_{P_{p(i)} \to P_i}^{(k)}$ pour $A^{(k)}$) :

$$\sum_{(i,j)\in E} \sum_{k} \mathsf{send}_{P_i \to P_j}^{(k)} \times \mathsf{size}^{(k)} \times c_{i,j} + \mathsf{send}_{P_{p(i)} \to P_i}^{(k)} \times \mathsf{size}_{\mathrm{res}}^{(k)} \times c_{i,p(i)} \leqslant 1$$

Alors que sans messages de retour, le temps de réception d'un processeur n'était pas limitant (car inférieur au temps d'émission de son père), il faut désormais le prendre en comte. Un processeur reçoit d'une part les messages de données de son père et d'autre par les messages de retour de ses fils :

$$\mathsf{send}^{(k)}_{P_{p(i)}\to P_i} \times \mathsf{size}^{(k)} \times c_{p(i),i} + \sum_{(i,j)\in E} \sum_k \mathsf{send}^{(k)}_{P_i\to P_j} \times \mathsf{size}^{(k)}_{\text{res}} \times c_{j,i} \leqslant 1$$

Nous rassemblons ces contraintes dans un programme linéaire :

$$\begin{array}{l}
\textsc{Maximiser } \rho_{\text{opt}} = \min_k \left\{ \frac{\alpha^{(k)}}{w^{(k)}} \right\}, \\
\textsc{sous les contraintes} \\
\left\{ \begin{array}{lll}
\text{(6.2a)} & \forall P_i \in V & \displaystyle\sum_k \alpha_i^{(k)} \times \mathsf{flops}^{(k)} \times z_i \leqslant 1 \\
\text{(6.2b)} & \forall P_i \in V & \displaystyle\sum_{(i,j)\in E} \sum_k \mathsf{send}^{(k)}_{P_i\to P_j} \times \mathsf{size}^{(k)} \times c_{i,j} \\
 & & \quad + \mathsf{send}^{(k)}_{P_{p(i)}\to P_i} \times \mathsf{size}^{(k)}_{\text{res}} \times c_{i,p(i)} \leqslant 1 \\
\text{(6.2c)} & \forall P_i \in V & \mathsf{send}^{(k)}_{P_{p(i)}\to P_i} \times \mathsf{size}^{(k)} \times c_{p(i),i} \\
 & & \quad + \displaystyle\sum_{(i,j)\in E} \sum_k \mathsf{send}^{(k)}_{P_i\to P_j} \times \mathsf{size}^{(k)}_{\text{res}} \times c_{j,i} \leqslant 1 \\
\text{(6.2d)} & \forall P_i \in V, \forall A^{(k)} & \mathsf{send}^{(k)}_{P_{p(i)}\to P_i} = \alpha_i^{(k)} + \displaystyle\sum_{(i,j)\in E} \mathsf{send}^{(k)}_{P_i\to P_j} \\
\text{(6.2e)} & \forall A^{(k)} & \alpha^{(k)} = \displaystyle\sum_{P_i\in V} \alpha_i^{(k)} \\
\text{(6.2f)} & \forall P_i \in V, \forall A^{(k)} & \alpha_k^{(i)} \geqslant 0 \\
 & \forall (P_i,P_j) \in E, \forall A^{(k)} & \mathsf{send}^{(k)}_{P_i\to P_j} \geqslant 0
\end{array} \right.
\end{array} \tag{6.2}$$

Comme précédemment, nous pouvons montrer que ce programme linéaire calcule une borne sur le débit réalisable.

Théorème 6.3. *Le programme linéaire 6.2 donne une borne supérieure pour le débit équitable d'un ordonnancement pour le problème de collections de tâches indépendantes avec retour sur la plate-forme G.*

Démonstration. La preuve est similaire à la preuve du théorème 6.1. ■

La reconstitution d'un ordonnancement précis des communications est plus délicate avec des messages de retour. Comme un processeur doit émettre des données à la fois vers ses fils et son père et recevoir de même, nous ne pouvons pas laisser le père gérer les communications avec ses fils, comme on pouvait le faire sans messages de retour.

Puisque la reconstruction directe est délicate, nous préférons nous appuyer sur les résultats obtenus dans le chapitre 3 concernant les communications dans le modèle un-port bidirectionnel. Nous avons en effet montré que lorsque les communications respectent les contraintes un-port

décrites dans la partie 3.2.1, on pouvait trouver un ordonnancement valide de ces communications, en s'appuyant sur la décomposition en couplages du graphe biparti des communications G_B. Les contraintes (6.2b) et (6.2c) du programme linéaire précédent impliquent que les communications respectent le modèle un-port bidirectionnel, donc on peut utiliser la méthode du chapitre 3, afin de reconstruire un ordonnancement valide de ces communications.

Comme, mis à part ce problème de reconstruction d'un ordonnancement des communications, la prise en compte des messages de retour ne change pas la complexité du problème, nous n'allons pas les considérer par la suite.

6.4 Heuristiques décentralisées

Comme nous l'avons remarqué dans la partie précédente, l'utilisation pratique de l'algorithme périodique décrit plus haut comporte quelques difficultés :

- La période de l'ordonnancement peut être grande, car elle est calculée comme le plus petit commun multiple des dénominateurs des quantités apparaissant dans la solution du programme linéaire. Ceci implique qu'il faut envoyer de grandes quantités de données par période, qui doivent être stockées dans la mémoire des processeurs. Pour éviter ce problème, nous voudrions construire une heuristique dynamique, fondée sur la solution du programme linéaire et donc réalisant un bon débit, mais sans utiliser autant de mémoire.
- De plus, il est délicat de modifier l'ordonnancement périodique pour prendre en compte des modifications de l'état de la plate-forme, compte tenu des décisions d'ordonnancement déjà prises, et de la quantité de tâches stockées dans la mémoire des processeurs.
- Calculer la solution du programme linéaire implique de rassembler au niveau du maître toutes les informations sur la plate-forme : bande-passante des liens et vitesse des processeurs, alors que ces quantités sont variables. Quand la taille de la plate-forme est importante, collecter des informations actualisées peut devenir difficile, voir impossible. C'est pourquoi nous aimerions pouvoir calculer une solution localement, c'est-à-dire prendre les décisions d'ordonnancement locales (en P_i, quelles tâches envoyer à quel fils de P_i, quelles tâches calculer sur place) sur des critères purement locaux : vitesse de calcul de P_i, bande-passante des liens menant à chacun des fils.

Nous allons donc décrire et évaluer plusieurs heuristiques décentralisées, en cherchant à quantifier deux aspects :

- Pour une solution fondée sur la solution du programme linéaire, quel est l'impact du contrôle décentralisé sur le débit ?
- Peut-on espérer, à l'aide d'un calcul complètement décentralisé de la solution, en n'utilisant que des paramètres locaux, réaliser un bon débit par rapport à l'optimal ?

6.4.1 Ordonnancement «à la demande»

Dans la suite, on suppose que les processeurs ont une mémoire qui leur permet de conserver un certain nombre M de tâches en attente d'être traitées localement ou déléguées à un autre processeur. Ce nombre est le même pour tous les processeurs et ne prend pas en compte la taille des tâches. Afin de ne pas dépasser la taille limite de la mémoire, les heuristiques d'ordonnancement que nous proposons fonctionnent «à la demande» : lorsqu'un processeur a de la place dans sa mémoire pour recevoir de nouvelles tâches (c'est-à-dire lorsque le nombre de tâches dans

la mémoire est inférieur à un seuil M/α), il envoie une requête à son père. Celui-ci envoie des tâches uniquement vers des processeurs qui en ont fait la demande. Ainsi, on peut assurer qu'un processeur ne recevra jamais de tâches qui ne pourraient pas être stockées. L'algorithme 6.2 présente le schéma général d'une heuristique à la demande décentralisée que nous utilisons.

```
S'il y a de la place dans la mémoire, envoyer une requête au père (en vérifiant qu'on n'a pas déjà envoyé trop de requêtes au père).
Recevoir les requêtes des fils, s'il y en a.
Déplacer les tâches reçues par le père (s'il y en a), dans la mémoire.
Choisir un couple (type de tâches k, exécutant P_j), où l'exécutant peut être soit le processeur local, soit un de ses fils, en utilisant la politique voulue, éventuellement en prenant en compte les requêtes et le type des tâches reçues.
S'il y a une tâche T de type k disponible dans la mémoire Alors
    Si P_j est le processeur local Alors
        Déplacer la tâche T dans la file d'attente du processeur local.
    Sinon
        Si on a reçu une requête du processeur P_j Alors
            Envoyer la tâche T au processeur P_j, dès que le port de communication est libre.
            Supprimer la requête de P_j qu'on satisfaite.
        Sinon
            Attendre la réception d'une tâche ou d'une requête.
Sinon
    Attendre la réception d'une tâche ou d'une requête.
```

Algorithme 6.2: Ordonnanceur à la demande, exécuté en chaque nœud.

6.4.2 Heuristique «à la demande» reposant sur le programme linéaire (HPL)

Dans un premier temps, nous utilisons le résultat du programme linéaire pour construire une heuristique. Celle-ci n'est donc pas «complètement» décentralisée puisqu'elle repose sur un calcul préalable centralisé d'une solution du programme linéaire. En revanche, dès qu'une solution optimale du programme linéaire est obtenue, le mécanisme de contrôle est, lui, décentralisé. En chaque processeur P_i, nous connaissons pour chaque application, la fraction de tâches qui doivent être exécutées localement et la fraction à envoyer à chaque processeur fils. On note $f_j^{(k)}$ la nombre de tâches à déléguer au processeur j en une unité de temps (ou à effectuer localement si $i = j$), dans la solution du programme linéaire :

$$f_i^{(k)} = \alpha_k^{(i)} \text{ et pour } j \neq i, f_j^{(k)} = \mathsf{send}_{P_i \to P_j}^{(k)}$$

On note $n_j^{(k)}$ le nombre de tâches de l'application k déjà déléguées au processeur j (local ou non). On utilise alors un mécanisme d'équilibrage unidimensionnel, décrit dans [16], pour choisir la prochaine décision d'ordonnancement : on choisit l'application k_0 et le processeur i_0 tels que :

$$\frac{n_{i_0}^{(k_0)}}{f_{i_0}^{(k_0)}} = \min_{i,k} \frac{n_i^{(k)}}{f_i^{(k)}}.$$

Si la taille de la mémoire est suffisamment grande, cette heuristique est supposée converger vers le débit optimal : seul le contrôle décentralisé par requêtes la différencie de l'ordonnancement périodique optimal. Nous espérons donc qu'elle réalisera des performances proches de l'optimal.

Nous abordons maintenant l'élaboration d'heuristiques complètement décentralisées.

6.4.3 Heuristique gloutonne simple (FIFO)

La première heuristique que nous considérons est très simple : elle consiste à servir les requêtes des fils dans leur ordre d'arrivée. Une tâche reçue par le père sera affectée, soit au processeur local si celui-ci est inactif, soit au premier processeur qui a demandé du travail, quelque soit le type de la tâche. Le maître s'assure que les différentes applications sont traitées équitablement en appliquant un équilibrage unidimensionnel utilisant les priorités des applications. En notant $n(k)$ le nombre de tâches de l'application $A^{(k)}$ déjà envoyées par le maître, la prochaine tâche envoyée par le maître sera du type k_0, où

$$\frac{n(k_0)}{w^{(k_0)}} = \min_k \frac{n(k)}{w^{(k)}}$$

Cette stratégie, très simple, est utilisée à titre de comparaison : comme elle n'utilise aucune information sur la plate-forme ou la taille des tâches, nous ne pouvons espérer qu'elle réalise de bonnes performances.

6.4.4 Heuristique mono-application à gros grain (MAGG)

Pour cette stratégie, nous nous basons sur des résultats antérieurs, obtenus pour le même type de problème mais avec une seule application. En effet, on connaît la solution optimal du problème d'ordonnancement d'une application consistant en un ensemble de tâches indépendantes, sur une plate-forme maître-esclaves en arbre. Ces résultats, décrits dans [17, 10], montrent qu'il suffit de prendre en compte la bande-passante de ses fils pour réaliser l'ordonnancement, en favorisant les processeurs avec lesquels on communique efficacement. Plus formellement, considérons tout d'abord le cas d'une plate-forme en étoile, où encore une fois, la puissance de calcul du maître est déplacée sur un processeur fils P_0 de coût de communication nul ($c_0 = 0$). On renumérote les processeurs, en les triant par par coût de communication croissant : $c_0 \leqslant c_1 \leqslant c_2 \leqslant \cdots \leqslant c_p$. Alors, on sait qu'il suffit d'utiliser les $m+1$ premiers processeurs pour obtenir une solution optimale, où m est le plus grand indice tel que

$$\sum_{i=1}^{m} \frac{c_i \times \mathsf{size}^{(i)}}{z_i \times \mathsf{flops}^{(i)}} \leqslant 1$$

Les m premiers processeurs sont utilisés à plein temps pour traiter des tâches, alors que le processeur m est utilisé uniquement lorsque le bus de communication est libre. Cette stratégie, privilégiant la bande-passante des processeurs fils et donc couramment nommée *bandwidth-centric*. Il a été montré que cette stratégie est optimale pour de nombreux modèles de communication, en particulier pour notre modèle un-port bidirectionnel avec recouvrement calcul/communication. Elle se généralise facile au cas d'un arbre : tout sous-arbre peut être assimilé à un processeur, de vitesse de calcul équivalente au nombre de tâches que le sous-arbre est capable de traiter en unité de temps. Des simulations ont également montré que le débit optimal pouvait être atteint

dès qu'on autorisait une mémoire capable de contenir un nombre suffisant de tâches sur chaque processeur (voir [70]).

Nous souhaitons donc étendre cette stratégie *bandwidth-centric*, que nous savons optimale pour une application, au cas avec plusieurs applications. Pour ceci, nous construisons une nouvelle application A_{agg} en agrégeant des tâches des différentes applications. Nous supposons ici que les priorités $w^{(k)}$ données aux applications sont des entiers. Une tâche de l'application A_{agg} est composée de $w^{(k)}$ tâches de l'application $A^{(k)}$, pour $k = 1, \ldots, K$. Cette nouvelle application est ensuite ordonnancée en utilisant l'approche centrée sur la bande-passante présenté ci-dessus.

Même si nous savons que la stratégie utilisée pour ordonnancer une application est optimale, son adaptation à plusieurs applications n'est pas optimale. En particulier dans le cas d'une étoile, cette heuristique consiste à envoyer la même proportion de tâche de chaque application à chaque processeur fils. Or nous avons montré dans la partie précédente que sur certaines plates-formes, seules les tâches de l'application de plus fort rapport communication/calcul devaient être envoyées aux processeurs de coût de communication le plus faible. Sur ces plates-formes, notre stratégie mono-application ne peut espérer réaliser le débit optimal.

6.4.5 Ordonnanceurs parallèles non-coopératifs (OPNC)

Nous voulons ici encore utiliser la stratégie *bandwidth-centric* optimale pour une application, décrite précédemment. Cependant, nous utilisons maintenant un ordonnanceur par application, soit K ordonnanceurs indépendant au total. Sur chaque processeur, K ordonnanceurs locaux sont exécutés, traitant chacun une application, et appliquant une stratégie *bandwidth-centric* avec ses propres requêtes. En particulier, lorsqu'un de ces ordonnanceurs calcule le temps de traitement d'une tâche sur le processeur, il ignore que des tâches d'autres applications ont peut-être été exécutées en même temps, ce qui conduit à des mesures peu fiables des coûts de calcul et de communication.

L'équité entre les applications est assurée par le maître, qui distribue les tâches aux différents ordonnanceurs avec une stratégie d'équilibrage unidimensionnel, reposant sur les priorités des applications.

Dans les simulations qui suivent, nous avons mis en œuvre la contrainte un-port pour les communications pour chaque ordonnanceur, qui ne peut effectuer qu'une seule émission à la fois. Cependant, nous n'imposons pas cette contrainte un-port globalement entre les différents ordonnanceurs de cette stratégie non-coopérative : il se peut que deux ordonnanceurs de type différents, présents sur le même processeur, effectuent chacun un envoi au même instant. Dans ce cas, nous imposons que le débit total en sortie du processeur ne soit pas supérieur à sa bande-passante de sortie. De la même façon, il se peut que la puissance de calcul d'un processeur soit partagée pour plusieurs tâches simultanées, une de chaque type. Ceci donne à cette stratégie un avantage par rapport aux autres. En effet, autoriser des communications concurrentes revient en quelque sorte à autoriser la préemption des communications, et il a été montré dans [34] qu'autoriser la préemption de communications diminuait considérablement la taille de la mémoire nécessaire pour obtenir un débit optimal.

6.4.6 Heuristique décentralisée centrée sur les données (HDCD)

Cette dernière heuristique tente, par un contrôle décentralisé, de converger vers une solution optimale. Cette stratégie utilise un ordonnanceur très proche de l'heuristique basée sur le résultat du programme linéaire : chaque processeur P_i connaît des valeurs $\alpha_k^{(i)}$ et $\mathsf{send}_{P_i \to P_j}^{(k)}$ qui lui indiquent pour chaque application la quantité de tâches qu'il doit exécuter localement et celle qu'il doit transmettre à chacun de ses fils. Ces pondérations sont ensuite utilisées pour répondre aux requêtes, avec un mécanisme d'équilibrage unidimensionnel semblable à celui de l'heuristique HLP. Seul le calcul décentralisé de ces poids diffère de l'heuristique HLP.

Nous nous appuyons sur le constat que l'application la plus difficile à ordonnancer est celle qui a le plus grand rapport communication/calcul, du fait du modèle de communication un-port. Notre heuristique commence donc par ordonnancer uniquement cette application, en utilisant la stratégie *bandwidth-centric*, puis nous introduisons des tâches d'autres applications, afin de retrouver une exécution équitable. Pour ce faire, le maître effectue les opérations suivantes. Dans la suite, nous désignons par $A^{(k_{\max})}$ l'application qui possède à un instant donné le plus grand débit pondéré et $A^{(k_{\min})}$ celle qui possède le plus petit débit pondéré. On va donc chercher augmenter le débit de $A^{(k_{\min})}$, éventuellement au prix d'une dégradation du débit de $A^{(k_{\max})}$.

- **Échange de communications.** Supposons que $r^{(k_{\max})} > r^{(k_{\min})}$, ce qui est le cas lorsqu'on n'a ordonnancé que des tâches de l'application de plus grand rapport communication/calcul. Lorsqu'un processeur n'est pas entièrement utilisé en calcul, il le reporte à son père. Le père peut alors substituer des tâches de type $k_{\max}$ au profit de tâches de type $k_{\min}$. Lors de la substitution, nous prenons garde à ne pas modifier le temps de communication : pour $\delta_{k_{\max}}$ tâches en moins de type $k_{\max}$, nous envoyons $\delta_{k_{\min}}$ tâches de type $k_{\min}$, avec $\mathsf{size}^{(k_{\max})} \times \delta_{k_{\max}} = \mathsf{size}^{(k_{\min})} \times \delta_{k_{\min}}$. De plus $\delta_{k_{\max}}$ et $\delta_{k_{\min}}$ sont limités par la puissance de calcul du processeur et sont tels qu'après la transformation, le débit pondéré de $A^{(k_{\min})}$ est au plus égal au débit pondéré de $A^{(k_{\min})}$.
- **Utilisation des communications libres.** Supposons qu'il existe un processeur P_i qui ne soit pas complètement utilisé en calcul ($\sum_k \alpha_k^{(i)} < 1$) et tels que tous les processeurs sur la route $P_{\mathsf{source}} \to \cdots \to P_i$ ne sont pas complètement utilisés en communication, c'est-à-dire que pour tout processeur P_u sur cette route,

$$\sum_{(u,v)\in E} \sum_k \mathsf{send}_{P_u \to P_v}^{(k)} \times \mathsf{size}^{(k)} \times c_{u,v} < 1$$

  Alors le maître peut prendre la décision d'envoyer quelques tâches de l'application minoritaire $A^{(k_{\min})}$ à P_i. La quantité de tâches envoyées est bornée à la fois par le temps de calcul disponible sur le processeur P_i et par le temps de communication disponible sur chaque processeur de la route $P_{\mathsf{source}} \to \cdots \to P_i$. De plus, après cette modification $A^{(k_{\min})}$ ne doit pas avoir un débit pondéré supérieur au débit pondéré de $A^{(k_{\max})}$.
- **Désaturation des communications** Il se peut que le maître soit entièrement saturé en communications et que les processeurs soit complètement utilisés en calcul, rendant impossible l'échange de communication décrit ci-dessus. Ceci peut se produire car les processeurs sélectionnés par la stratégie *bandwidth-centric* appliquée initialement à l'application de plus fort rapport communication/calcul sont les plus performants en communication, mais pas nécessairement en calcul. Si l'exécution n'est pas équitable, il faut alors utiliser d'autres processeurs, moins puissant en communication, mais plus performant pour les calculs et donc plus adaptés aux applications de faible rapport communication/calcul.

Pour ce faire, il nous faut libérer du temps de communication sur le maître. Nous choisissons alors la branche de l'arbre utilisée qui a le plus grand coût de communication et nous diminuons le nombre de tâches de l'application $A^{(k_{\max})}$ envoyées sur cette branche. Le temps de communication libre qui apparaît est alors mis à profit pour l'application $A^{(k_{\min})}$ en utilisant la technique précédente.

– **Échange de tâches sur le maître** Il peut arriver que les tâches de l'application majoritaire $A^{(k_{\max})}$ soit effectuées sur un seul processeur. Dans ce cas, il faut substituer des tâches de $A^{(k_{\min})}$ à la place de celles de $A^{(k_{\max})}$ sur ce processeur, tout en respectant les contraintes de calcul sur le processeur concerné et de communication sur la route menant de P_{source} à ce processeur.

Les opérations précédentes sont donc effectuées sur la plate-forme, dans l'ordre dans lequel nous les avons présentées, jusqu'à ce que une équité relative soit atteinte, c'est à dire par exemple jusqu'à ce que :

$$\frac{\max_k \left\{ \frac{\rho^{(k)}}{w^{(k)}} \right\} - \min_k \left\{ \frac{\rho^{(k)}}{w^{(k)}} \right\}}{\min_k \left\{ \frac{\rho^{(k)}}{w^{(k)}} \right\}} < 0.05$$

On pourrait avoir l'impression que certaines de ces opérations nécessitent une connaissance globale de la plate-forme, cependant toutes les opérations sont effectuées soit en remontant de l'information jusqu'au maître, soit diffusant sur une branche en les décisions du maître. Par exemple pour l'opération d'utilisation des communications libres, un processeur P_i qui n'est pas complètement utilisé envoie cette information, avec la quantité de calcul disponible, à son père. Cette information remonte jusqu'au maître, avec une autre information : la quantité de communication minimum disponible sur la route empruntée. Toutes ces opérations peuvent être effectuées avec une mécanisme distribué semblable.

6.5 Évaluation par simulation

Dans cette partie, nous présentons les résultats obtenues en comparant les heuristiques décrites précédemment par simulation.

6.5.1 Méthodologie

Mesure du débit équitable Nous souhaitons évaluer les différentes heuristiques dans leur régime permanent. Cependant, nous allons tester ces heuristiques en utilisant un nombre fini de tâches. Dans ces conditions, il est difficile d'estimer quand le régime permanent est atteint, surtout lorsque les algorithmes utilisés ne sont pas périodiques. Nous adoptons une vision pragmatique : en supposant que la simulation commence au temps 0, nous notons T le premier instant où une application est terminée (toutes ses tâches ont été effectuées). Nous supposons alors que le la phase de régime permanent correspond à l'intervalle $[\epsilon T, (1-\epsilon)T]$ (pour $\epsilon \in [0, 0.5]$). En notant $\nu^{(k)}(t)$ le nombre de tâches de l'application $A^{(k)}$ effectuées au temps t, on peut définir le débit de l'application $A^{(k)}$ dans la phase de régime permanent :

$$\rho^{(k)} = \frac{\nu^{(k)}((1-\epsilon)T) - \nu^{(k)}(\epsilon T)}{(1-2\epsilon)T}.$$

Le paramètre ϵ nous permet d'éliminer les instabilités des phases initiales et finales. En pratique, nous choisirons $\epsilon = 0.1$. Par la suite, nous appellerons $\rho^{(k)}$ le *débit expérimental* de l'application $A^{(k)}$, pour ne pas confondre avec le débit théorique calculé par programmation linéaire. De même le minimum des débit expérimentaux pondérés par les priorités est appelé le *débit équitable expérimental*.

Génération de plates-formes Les plates-formes utilisées dans nos simulations sont des arbres aléatoires décrits par deux paramètres : le nombre total de nœud n et le degré maximal d'un nœud $deg_{\max}$. Nous utilisons l'algorithme 6.3 pour générer ces arbres, qui effectue un parcours en largeur.

```
1:  nœuds_à_traiter ← ∅
2:  Insérer P_source dans nœuds_à_traiter
3:  Tant que nœuds_à_traiter ≠ ∅ :
4:      P_i = premier élément de nœuds_à_traiter
5:      Si n = 0 Alors
6:          stop      {plus de nœuds à ajouter}
7:      Répéter
8:          nb_fils ← entier aléatoire de [0, deg_max]
9:          nb_fils ← min{n, nb_fils}
10:     jusqu'à ce que |nœuds_à_traiter| > 0 ou nb_fils > 0
            {on évite ainsi que le dernier nœud de nœuds_à_traiter n'ait aucun
        fils alors qu'il faudrait encore ajouter des nœuds.}
11:     Pour u ∈ [1, nb_fils] :
12:         n ← n − 1
13:         Créer un nœud P_j relié à P_i
14:         Ajouter P_j dans nœuds_à_traiter
```

Algorithme 6.3: Génération de plates-formes en arbres. En vérifiant que la dernière branche traîtée possède un nombre non nul de fils (ligne 10), on assure que l'arbre créé aura exactement n nœuds.

Pour les simulations, nous générons des arbres possédant 5, 10, 20, 50 ou 100 nœuds. Le degré maximal d'un nœud vaut 2, 5, 10 ou 15, et pour chaque configuration possible (avec $deg_{\max} \leqslant n$), nous générons 10 arbres de mêmes paramètres, ce qui donne 150 arbres en tout.

Nous assignons ensuite aux processeurs et aux liens de communication des valeurs de vitesse de calcul, de latence et de bande-passante. Ces valeurs proviennent de mesures sur des plates-formes réelles. La vitesse de calcul des processeurs va de 22.1 Mflop/s (correspondant à un Pentium Pro à 200Mhz) à 171.7 Mflop/s (un Athlon 1800). La capacité des liens de communication va de 110kb/s à 7 Mb/s et la latence de 6ms à 10s. Le simulateur que nous utilisons prend en compte à la fois la latence et la capacité des liens pour déterminer la bande-passante effective d'un transfert, même si notre modélisation ne prend pas en compte la latence des liens.

Applications Une application est principalement décrite par son rapport communication/calcul. Nous générons des applications dont les rapports vont de 0.001 (correspondant à une multiplication de matrices 3500×3500), jusqu'à 4.6 (correspondant à une addition de matrices de cette taille). Pour choisir les types d'application, nous posons $r_{\min} = 0.001$ et nous

choisissons $r_{\max}$ dans $[0.001, 4.6]$. Les rapports communication/calcul des applications sont alors choisies dans $[r_{\min}, r_{\max}]$. Pour plus de simplicité, nous choisissons d'utiliser $K = 2$ applications, de même priorité $w^{(1)} = w^{(2)} = 1$.

Implantation des heuristiques Les cinq heuristiques ont été implantées dans le simulateur SimGrid [36, 73]. Les valeurs des coûts de communication $c_{i,j}$ et de calcul z_i utilisées pour le programme linéaire ou pour d'autres heuristiques ont été mesurées dans le simulateur, en mesurant le temps nécessaire au transfert d'un message de taille connue, ou le temps d'exécution d'une tâche de taille connue.

Nos heuristiques à la demande s'appuient sur un mécanisme de requêtes, qui a été intégré dans la simulation, en utilisant de petites tailles de messages (quelques octets). Nous avons pris garde à ce qu'aucun inter-blocage ne perturbe ce mécanisme, même lors de variations de charge sur la plate-forme. Nous utilisons 200 tâches par application et nous avons vérifié à la main en utilisant un très grand nombre de tâche (2000) que 200 tâches suffisaient pour faire apparaître une phase de régime permanent.

6.5.2 Résultats expérimentaux

Comparaison des débits expérimental et théorique Pour les heuristiques HLP, MAGG et HDCD, nous pouvons calculer le débit théorique : il s'agit soit du débit du programme linéaire, soit du débit théorique de la stratégie *bandwidth-centric* appliquée à la macro-application de MAGG, soit du débit obtenu sur le maître après convergence du calcul distribué des pondérations dans HDCD. Ceci nous permet de comparer le débit théorique par rapport au débit expérimental sur ces deux heuristiques. De nombreux facteurs peuvent entraîner une différence de ces débits, comme la latence induite par le mécanisme de requêtes, ou bien une phase d'initialisation qui serait plus longue que les 10% que nous utilisons. En fait, il s'avère que seule la taille de la mémoire disponible en chaque nœud affecte le débit expérimental de façon significative.

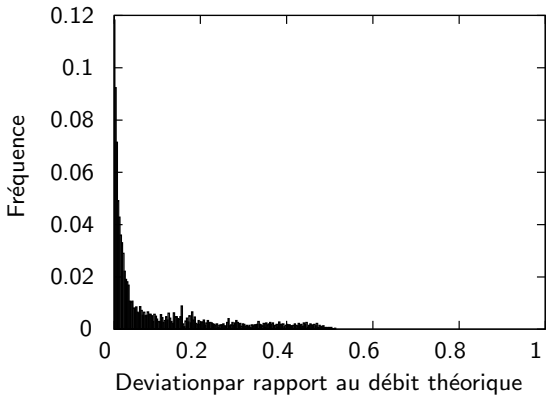
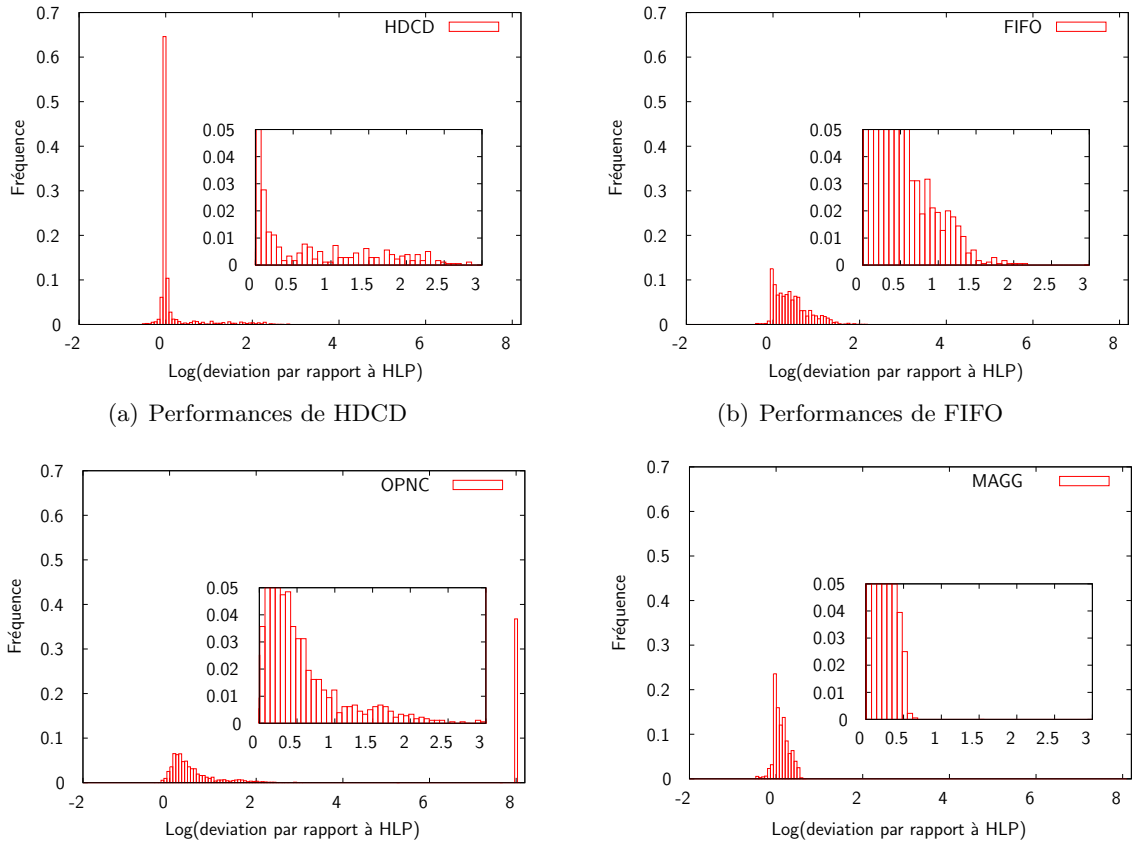


FIG. 6.3 – Distribution de la déviation du rapport débit expérimental/débit théorique, pour les heuristiques HLP et MAGG

Dans nos simulations, nous utilisons en général une taille de mémoire lui permettant de contenir 10 tâches de n'importe quelle application. Dans ce cas, la figure 6.3 présente la distribution de la déviation du débit équitable expérimental par rapport au débit équitable théorique,

pour ces deux heuristiques (les deux heuristiques ont une distribution semblable). La déviation moyenne vaut 9.4 %. Quand nous portons la taille de la mémoire à 200 tâches (et nous augmentons le nombre total de tâches par application à 2000) alors la déviation moyenne chute à 0.33%. Même si cette taille de mémoire, correspondant quasiment à une mémoire illimitée, permet d'obtenir un débit plus proche du débit théorique, nous considérons qu'elle n'est pas très réaliste, et nous conservons une taille de 10 tâches pour la suite des expériences.



(a) Performances de HDCD

(b) Performances de FIFO

(c) Performances de OPNC

(d) Performances de MAGG

FIG. 6.4 – Performances des heuristiques, rapportées à celle de HLP

.

Performances des heuristiques Nous utilisons l'heuristique HLP comme référence dans la comparaison des heuristiques, puisqu'elle fournit une bonne estimation du débit atteignable pour un ordonnanceur décentralisé. Pour chaque configuration expérimentale, correspondant à une plateforme et un intervalle $[r_{\min}, r_{\max}]$ dans lequel sont choisis les rapports communication/calcul des applications, nous calculons le rapport entre le débit expérimental de l'heuristique à comparer et le débit expérimental de l'heuristique HLP. La figure 6.4 présente, pour chaque heuristique testée, la distribution du logarithme népérien de ce rapport. Ainsi, les valeurs positives correspondent au cas où HLP se comporte mieux que l'heuristique testée, tandis que les valeurs négatives représentent les cas où l'heuristique testée a un meilleur débit que HLP.

Tout d'abord, on peut remarquer que les valeurs positives sont beaucoup plus représentées, ce

qui indique la supériorité de HLP. D'autre part, on peut voir sur la figure 6.4(a) que l'heuristique HDCD est très proche de HLP, bien qu'elle utilise un calcul distribué des pondérations utilisées par l'ordonnanceur. Cependant, en moyenne, l'heuristique HDCD est 1.164 fois plus mauvaise que HLP, ce qui est légèrement plus que ce que réalise MAGG (voir figure 6.4(d)) qui est 1.156 fois plus mauvais que HLP. La raison est que bien que HDCD soit en général très proche de HLP, dans quelques rares cas, elle réalise un débit beaucoup plus mauvais, jusqu'à 16 fois pire, alors que MAGG est beaucoup plus stable : dans le pire cas, elle est seulement 2 fois plus mauvaise que HLP.

Sans surprise, la stratégie non-coopérative OPNC réalise de mauvais résultats (figure 6.4(c). Dans de nombreuses situations (35% des cas), une des application est particulièrement défavorisée et son débit expérimental est proche de zéro. Le logarithme du rapport avec HLP a été normalisé à 8 dans ce cas, pour apparaître sur la figure. Ces faibles résultats montrent que la prise en compte de l'équité au niveau de l'ordonnanceur est importante.

Enfin, la stratégie gloutonne simple FIFO (figure 6.4(b)) est en moyenne 1.56 plus mauvaise que HLP et dans le pire cas 8 fois plus mauvaise. Sur de petites plates-formes, FIFO donne des résultats comparable à MAGG. En revanche, sur des plates-formes plus étendues (50 à 100) nœuds, MAGG se comporte beaucoup mieux (rapport 1.24) que FIFO (rapport 2.04).

6.6 Conclusion

Dans ce chapitre, nous avons étudié le problème de l'ordonnancement de plusieurs applications concurrentes consistant chacune en un ensemble de tâches indépendantes de même taille, sur une plate-forme maître-esclaves organisé en arbre. Nous avons montré comment calculer, avec un contrôle centralisé, le débit optimal en régime permanent et comment reconstruire un ordonnancement qui réalise ce débit. Pour combler les lacunes de l'ordonnancement centralisé, nous proposons des heuristiques décentralisées, s'appuyant sur des informations locales. Nous évaluons ces heuristiques par simulation et montrons que les heuristiques les plus élaborées réalisent un débit proche de celui réalisé par une heuristique fondée sur la solution optimale centralisée. Un autre avantage de ces heuristiques décentralisées par rapport à un ordonnancement centralisé est qu'elle peuvent facilement s'adapter à des variations de performances de la plate-forme : les pondérations qui servent de guides à l'ordonnanceur en chaque nœud sont recalculées en permanence, ce qui rend ces heuristiques naturellement adaptatives.

Nous montrons également que l'équité entre applications doit être prise en compte dans l'ordonnancement local, sous peine de défavoriser complètement une application : quand plusieurs ordonnanceurs cohabitent sans collaborer, l'exécution n'est plus équitable.

Nous avons également réfléchi à une stratégie décentralisée fondée sur un algorithme décentralisé proposé par Awerbuch et Leighton dans [7, 6] pour résoudre le problème de «multi-commodity flow». Ce problème consiste à tenter de router plusieurs flots de messages concurrents dans un réseau, en respectant les capacités limités des liens de communication. Pour utiliser cet algorithme, nous identifions chaque type de message du problème «multi-commodity flow» avec une de nos applications $A^{(k)}$. Nous avons résolu plusieurs problèmes, comme prendre en compte les contraintes de communication du modèle un-port et les contraintes de calcul sur les nœuds, étant données les différentes tailles en calcul et communication des tâches. Nous sommes en mesure de montrer que l'adaptation de l'algorithme pour l'ordonnancement d'application mul-

tiple converge vers une solution optimale. Cependant, plusieurs problèmes restent à résoudre, qui rendent ce travail incomplet :

- L'algorithme d'Awerbuch et Leighton nécessite de connaître un débit réalisable avant d'être effectué. Plus précisément, sachant qu'un débit $(1+2\epsilon)\rho$ est réalisable, l'algorithme fournit une solution de débit $(1+\epsilon)\rho$. Il nous faudrait donc un algorithme décentralisé pour calculer le débit optimal, ou une approximation du débit optimal, pour pouver utiliser ensuite cette méthode.
- La vitesse de convergence de cet algorithme est relativement faible. La figure 6.5 montre par exemple une petite plate-forme avec un maître et cinq processeurs esclaves, ainsi que l'évolution du nombre de tâches envoyées sur les liens de communication, lors de la simulation de l'exécution de l'algorithme adapté sur cette plate-forme avec deux applications concurrentes. Quand le nombre de tâches envoyées devient constant, l'algorithme a convergé vers le débit optimal. Chaque étape représente une synchronisation locale entre tout couple de processeurs voisins. On voit que même sur une plate-forme de cette taille, la convergence de l'algorithme nécessite au moins une centaine d'étapes de synchronisation. Par comparaison, les opérations nécessaires pour rassembler toute l'information sur le maître ne demandent que quelques synchronisations.

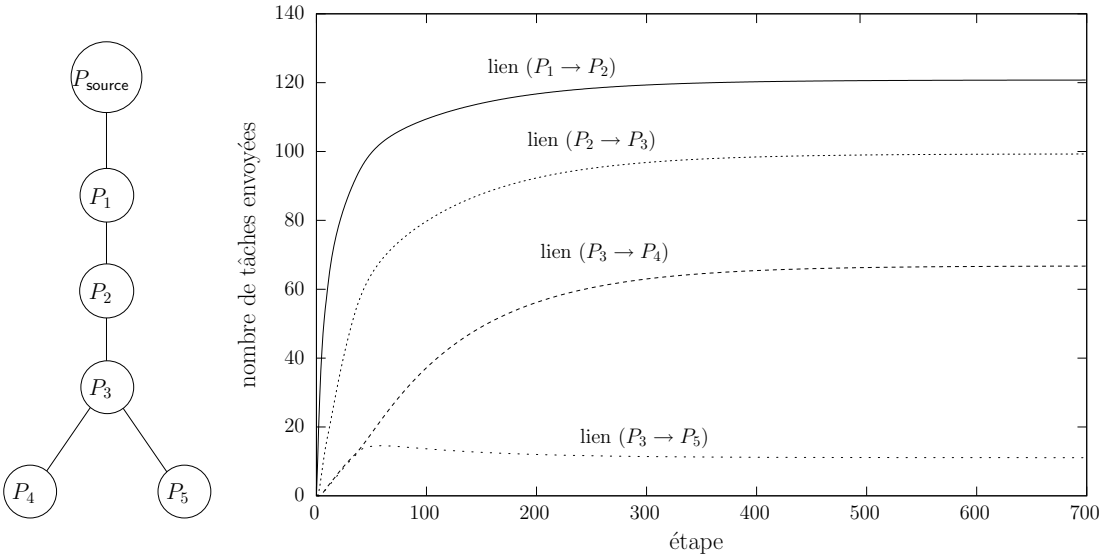


FIG. 6.5 – Simulation de l'algorithme de «multi-commodity flow» adapté au problème d'ordonnancement, sur une petite plate-forme.

Cependant, l'algorithme proposé par Awerbuch et Leighton a l'avantage de s'adapter naturellement aux variations de la plate-forme. Il serait donc intéressant de l'utiliser sur des plates-formes de grandes tailles, si nous parvenons à résoudre les deux problèmes évoqués, en concevant un algorithme distribué de calcul du débit optimal et en accélérant sa vitesse de convergence. Cet algorithme décentralisé fournirait des pondérations, qui pourraient ensuite être utilisées dans une heuristique comme celle fondée sur la solution du programme linéaire. Les résultats obtenus n'étant que préliminaires, nous ne les présentons pas ici et renvoyons le lecteur interessé au rapport de recherche correspondant [18].

Chapitre 7

Tâches divisibles pour plates-formes à grande échelle

7.1 Introduction

Dans ce chapitre, nous présentons l'étude d'un autre problème d'ordonnancement sur une plate-forme à grande échelle. Nous nous intéressons ici à la modélisation des applications comme *tâches divisibles*, introduite par Robertazzi dans [29]. Dans ce modèle, on suppose qu'une application consiste en une quantité de travail qui peut être découpée en un nombre quelconque de sous-tâches, de tailles quelconques et indépendantes entre elles. Ceci correspond à une tâche parfaitement parallèle : on peut répartir son exécution sur un ensemble quelconque de processeurs sans surcoût de calcul. Ce modèle est une approximation réaliste d'applications consistant en un grand nombre de tâches indépendantes de petite taille, et il a ainsi été utilisé pour résoudre un grand nombre de problèmes d'ordonnancement pour des applications scientifiques, dans des domaines comme le traitement d'images, la bio-informatique ou encore l'exploitation de banques de données, (voir par exemple [28, 84, 53]). La souplesse de ce modèle a permis de l'utiliser pour rechercher des ordonnancements sur des plates-formes de type «maître-esclaves», pour exécuter une seule application, sur des plates-formes en étoile avec réseau homogène et des processeurs hétérogènes [93]. Dans [40], ces résultats sont étendus à un réseau hétérogène. Dans [85, 1] sont étudiées des stratégies pour ordonnancer une collection de tâches divisibles, sur une plate-forme hétérogène, avec un modèle de grappe de calcul très détaillé.

Nous voulons étudier l'ordonnancement de telles tâches divisibles sur une plate-forme à grande échelle, de type grille de calcul. Cependant, il n'est pas réaliste de penser qu'une grille de calcul puisse être utilisée pour effectuer une seule application. Au contraire, un grand nombre d'applications se partagent en général les ressources d'une telle grille, et la majeure de ces applications peuvent être modélisées comme des tâches divisibles. Par exemple, une grille de calcul comme la *CDF Analysis Farm* [104] permet l'exécution concurrente de plusieurs applications qui sont presque toutes des tâches divisibles.

Une première étude pour l'exécution concurrente de plusieurs applications a été proposée dans [27]. Dans cette étude, la plate-forme consiste en un bus de données reliant un maître à plusieurs processeurs esclaves. Dans [101], les auteurs présentent une architecture virtuelle producteur-consommateur où plusieurs maîtres sont reliés à des processeurs hétérogènes. Les auteurs décrivent une stratégie pour répartir et équilibrer le travail entre les processeurs. Mal-

heureusement, les résultats n'ont pas un intérêt pratique évident du fait du modèle de communication utilisé : les auteurs supposent qu'un nombre illimité de connexions peuvent avoir lieu simultanément, ce qui n'est pas très réaliste. Enfin, dans [102], les auteurs étudient comment appliquer le paradigme des tâches divisible au calcul sur grilles. Ils décrivent un modèle maître-esclaves dans lequel l'architecture se résume à un réseau en étoile, sans prendre en compte le fait que les processeurs sont géographiquement dispersés dans différents sites.

Nous cherchons donc à établir un nouveau modèle plus réaliste pour le réseau d'interconnexion d'une grille de calcul, adapté à l'étude de l'ordonnancement d'applications divisibles. Sur une telle plate-forme, nous cherchons à effectuer plusieurs applications divisibles en assurant une exécution efficace et équitable. Nous montrons que le problème considéré est NP-complet, et nous proposons plusieurs heuristiques pour le résoudre, que nous comparons par simulation.

Comme dans les chapitres précédents, nous allons chercher à optimiser le débit du régime permanent, c'est-à-dire le nombre de tâches effectuées par unité de temps, tout en assurant une certaine équité entre les applications.

7.2 Modélisation de la plate-forme et des applications

7.2.1 Réseau d'interconnexion longue-distance

Nous cherchons ici à obtenir une modélisation d'une plate-forme de calcul distribuée de type «grille» qui capture précisément ses caractéristiques, mais qui est suffisamment simple pour permettre l'élaboration d'algorithmes efficaces.

Modèle fluide Lors d'un transfert entre deux sites d'une grille de calcul, le message à transférer est découpé en paquets de petite taille, qui sont ensuite transmis de façon pipelinée sur les liens du réseau. Bien que ce comportement à base de paquets soit utilisé dans les simulateurs de réseau comme NS [108], la complexité de la modélisation ces mécanismes rend l'élaboration et l'analyse d'ordonnancements difficile. Au contraire, on peut modéliser le comportement macroscopique des transferts utilisant des mécanismes de paquets [79]. Plusieurs modèles ont en particulier été proposés pour calculer l'allocation de bande-passante à des connexions TCP [42, 48, 80].

D'autre part, sur des réseaux longue-distance, l'établissement d'une connexion entre deux sites prend un temps non négligeable, et la latence doit parfois également être prise en compte pour l'exécution des applications : dans [35], les auteurs remarquent que le lancement d'une tâche avec l'intergiciel Globus [49] peut demander jusqu'à 25 secondes.

Ces observations nous conduisent à utiliser un modèle fluide : des connexions entre sites distants sont établies au début de l'exécution, et sont conservées jusqu'à sa fin. Ainsi, la latence due à l'établissement des connexions et au déploiement des calculs n'intervient qu'une seule fois, au début de l'ordonnancement. Même si elle est de taille importante, elle est difficile à quantifier et incompressible : on ne peut pas éliminer le coup de déploiement d'une application. Par la suite, nous utiliserons donc des connexions permanentes et nous ne prendrons en compte aucune latence.

Partage de bande-passante Une des caractéristiques que nous voulons prendre en compte est la différence entre les réseaux locaux et les réseaux longue-distance utilisés. Une des différences

essentielles concerne le partage de bande-passante pour plusieurs transferts se partageant un lien de communication. La figure 7.1 présente le résultat de mesures effectuées par H. Casanova et Y. Yang [35], en effectuant plusieurs transferts simultanés entre deux machines. Les deux machines sont soit situées dans le même laboratoire («Local»), soit entre l'université de San Diego et un site distant : l'université de Delft, Pays-Bas («Netherlands»), l'université de Virginie, États-Unis («UVA») ou l'université de Washington, États-Unis («UW»). Cette figure présente la bande-passante d'un transfert en fonction du nombre de transferts concurrents (c'est-à-dire du nombre de connexions TCP simultanées). Ces résultats sont normalisés par la bande-passante obtenue par une connexion.

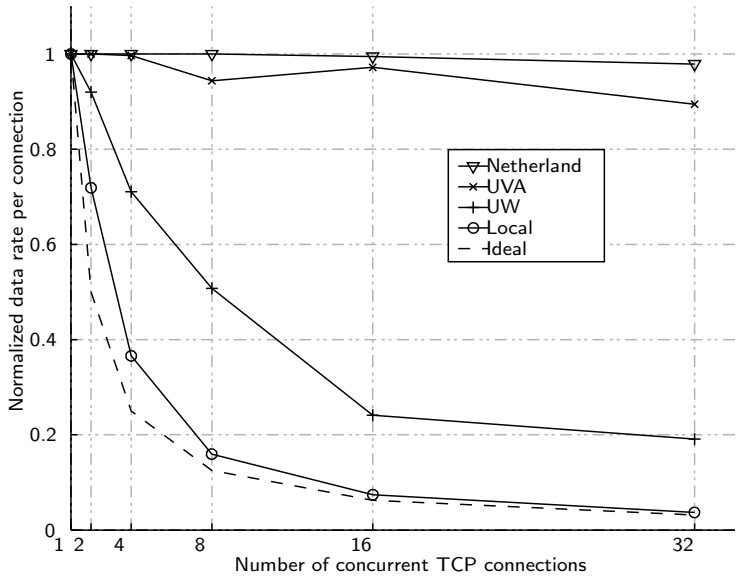


FIG. 7.1 – Évolution de la bande-passante d'une connexion en fonction du nombre de communications concurrentes, pour différents transferts. Publié par H. Casanova et Y. Yang dans [35].

On peut distinguer deux types de comportement :

- Pour les transferts locaux, le partage de bande-passante est très proche du partage théorique (noté «Ideal»), qui attribue une bande-passante B/x à chaque connexion, lorsque x transferts traversent un lien de bande-passante B. Nous utiliserons ce modèle pour les liens locaux.
- Pour les connexions longue-distance («UVA» et «Netherlands»), lorsque plusieurs connexions sont utilisées simultanément, elles se voient toutes allouer la même bande-passante qu'à une seule connexion. Ce comportement à des causes multiples. En particulier la bande-passante d'une connexion TCP est limitée par la taille de la fenêtre de congestion. D'autre part, un lien longue-distance est en général utilisé par un très grand nombre de connexions, ce qui fait que la congestion entre les connexions d'une même application est négligeable.

Pour tirer profit du comportement des liens longue-distance, les applications réalisant des communications sur les grilles de calcul s'autorisent à ouvrir plusieurs connexions TCP pour effectuer un même transfert, comme le fait GridFTP [4]. Nous voulons également prendre en compte ce comportement. Nous supposerons donc que les liens longue-distance peuvent supporter plusieurs connexions simultanées sans dégradation de performances : sur un lien longue-distance toute les connexions se voient attribuer la même bande-passante. Cependant, le nombre

de connexions simultanées sur un même lien doit également être bornée, d'une part parce que le comportement sans congestion ne serait plus justifié pour un très grand nombre de connexions, d'autre part parce que nous ne voulons pas que les connexions d'une application représentent une fraction trop importante de la charge d'un lien, mais également parce que l'interface de la machine émettrice n'est sans doute pas capable de gérer plus d'un nombre déterminé de connexions.

Routage fixe Dans les études précédentes, nous avons supposé que les tâches ou les messages à transmettre pouvait emprunter n'importe quelle route entre les machines pour atteindre leur destination. Nous utilisons même parfois plusieurs routes concurrentes afin d'améliorer le débit. Afin de se rapprocher de l'utilisation réelle des plates-formes de calcul, nous allons ici supposer que le routage est fixé : si un site A souhaite déléguer du travail à un autre site B, les données empruntent une route déterminée. De même, la grappe B ne peut à son tour envoyer le travail reçu de A à un troisième site C, car cela reviendrait à utiliser un routage au niveau applicatif, ce que l'on s'interdit ici. Cette contrainte va certainement conduire à des performances plus faibles, mais elle nous paraît importante pour se conformer à l'utilisation réelle des plates-formes.

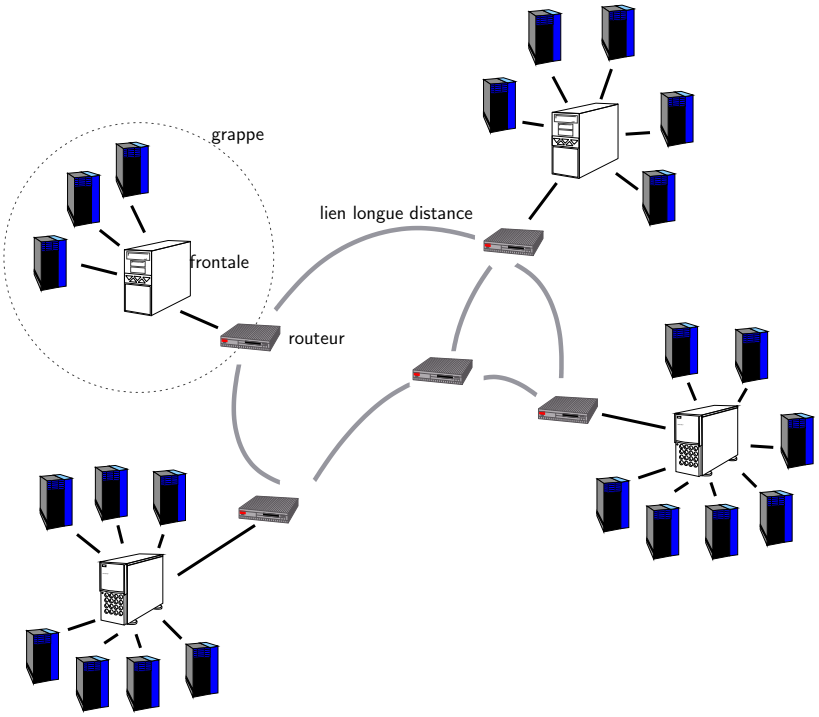


FIG. 7.2 – Exemple de plate-forme à grande échelle.

Notations Comme illustré à la figure 7.2, notre plate-forme consiste en une collection de grappes de calcul G_k, chacune munie d'un processeur «frontal» M_k, connecté à un routeur local R_k par un lien local de capacité limité. Ces routeurs locaux sont connectés entre eux, en utilisant des liens longue-distance. On modélise le réseau d'interconnexion de ces routeurs par un graphe $\mathcal{G}_{\text{ic}} = (\mathcal{R}, \mathcal{B})$ composé de routeurs (les nœuds de $\mathcal{R}$) et de liens longue-distance (les arêtes de $\mathcal{B}$). Notons que l'ensemble $\mathcal{R}$ contient à la fois des routeurs reliant une grappe au réseau

longue-distance, et des routeurs intermédiaires, qui n'appartiennent à aucune grappe. Chaque lien longue-distance $\lambda \in \mathcal{B}$ est muni de deux paramètres : la bande passante $\mathsf{bw}(\lambda)$ disponible pour toute connexion empruntant ce lien, et le nombre maximal de connexions $\mathsf{max\text{-}connect}(\lambda)$ qui peuvent être ouvertes simultanément sur ce lien. On suppose ainsi que chacune de ces connexions se verra attribuer une quantité fixe de bande-passante, égale à $\mathsf{bw}(\lambda)$, et ce pour un certain nombre de connexion $\mathsf{max\text{-}connect}(\lambda)$, à partir duquel il n'est plus possible d'ouvrir de nouvelles connexions.

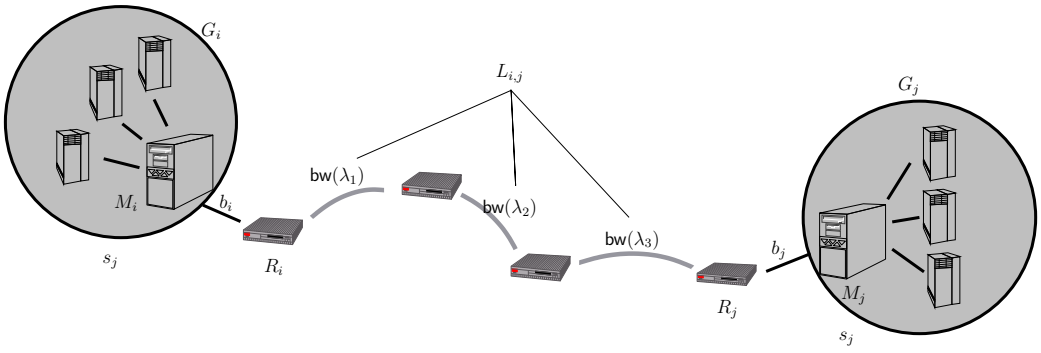


FIG. 7.3 – Notations pour la modélisation de la plate-forme.

Enfin, nous supposons que le routage dans le graphe inter-grappe est fixé. La table de routage nous indique, pour chaque paire de grappes (G_k, G_l), la liste $L_{k,l}$ des liens longue-distance utilisés par une connexion de G_k à G_l, c'est-à-dire du routeur R_k au routeur R_l. La figure 7.3 résume les notations utilisées.

7.2.2 Éléments de calcul

Les ressources de calcul consistent en K grappes de calcul G_k, $1 \leqslant k \leqslant K$. Pour être tout à fait général, nous devrions représenter chaque grappe G_k par un graphe muni de coûts de calcul et de coûts de communication, mais nous utilisons un modèle simplifié. Pour chaque grappe, nous n'utilisons que le processeur frontal M_k, connecté au routeur local R_k (un des routeurs de $\mathcal{R}$). Nous considérons qu'une grappe de calcul est équivalente à un seul processeur M_k représentant la capacité de calcul cumulée de toute la grappe, comme représenté sur la figure 7.3. On sait que, dans le contexte des tâches divisibles, une plate-forme maître-esclaves en étoile est équivalente à un seul processeur, dont la vitesse de calcul s_k peut être calculée comme dans [83, 14, 10]. Il a même été montré que ce procédé pouvait être étendu à une topologie en arbre, qui peut elle aussi être assimilée à un seul processeur agrégeant toute la capacité de calcul de la plate-forme [14, 13, 19]. Notre modèle peut donc prendre en compte tous les réseaux locaux organisés en arbre. Nous n'avons besoin que de deux paramètres pour caractériser chaque grappe de calcul : la vitesse de calcul cumulée s_k de M_k, et la bande-passante du lien de communication local reliant M_k à R_k, notée b_k. Ce lien de communication local est modélisé de la façon suivante : un nombre quelconque de connexions peuvent partager ce lien, cependant la somme des bandes-passantes des connexions empruntant ce lien, quelque soit leur direction (entrant ou sortant de la grappe G_k) ne peut dépasser b_k. Remarquons que ce modèle peut prendre en compte le cas où la route de M_k à R_k emprunte successivement plusieurs liens locaux ; dans ce cas b_k sera le minimum de la bande-passante de chacun de ces liens. Nous ne supposerons pas ici que les machines (frontale ou routeur) respectent le modèle un-port, mais

que plusieurs connexions simultanées peuvent exister entre ces différents éléments, pourvu que les contraintes de capacité soient respectées. En particulier, on supposera qu'il est possible de contrôler le débit de chaque connexion, en utilisant des méthodes comme celle décrite dans [41].

Nous avons vu précédemment que la latence pouvait également être un paramètre important pour les calculs. Cependant, nous considérons que le déploiement des calculs, comme l'ouverture de connexions TCP, se fait uniquement au début de l'ordonnancement. Nous ne nous intéressons pas précisément à l'ordonnancement des calculs sur une grappe, mais nous supposons que lorsque plusieurs applications doivent s'exécuter sur une même grappe, celles-ci peuvent être effectuées simultanément, en affectant une partie des machines de la grappe à chaque application. Nous obtenons ainsi un modèle parfaitement «fluide», ou la vitesse de calcul d'une grappe peut être répartie entre plusieurs applications (une sorte de «time-sharing» de la grappe). Cette vision fluide est compatible avec une modélisation des applications sous forme de tâches divisibles.

7.2.3 Applications

Nous considérons K applications modélisées comme des tâches divisibles, une application par grappe : on considère que la grappe G_k possède initialement toutes les données de l'application $A^{(k)}$. Chaque grappe est donc à l'origine d'une application mais elle va chercher à déléguer une partie du travail à d'autres grappes moins chargées qu'elle. Comme au chapitre précédent, l'application $A^{(k)}$ est munie d'une priorité $w^{(k)}$, qui quantifie son importance relative. Pour chaque application $A^{(k)}$, on appelle $\nu^{(k)}(t)$ le nombre de tâches de cette application effectuées en temps t. À tout instant t, on peut définir le débit de l'application $A^{(k)}(t) : \alpha^{(k)} = \nu^{(k)}(t)/t$. Afin de garantir une exécution équitable, nous cherchons à maximiser le minimum des débits pondérés : $\min_k \alpha^{(k)}(t)/w^{(k)}$, correspondant à l'équité max-min entre les différentes applications, munies de coefficients $1/w^{(k)}$.

À une unité de travail pour une application $A^{(k)}$ donnée, correspondent une quantité de données $\mathsf{size}^{(k)}$ qui doit être transmise au processeur effectuant ce travail, et une quantité de calculs $\mathsf{flops}^{(k)}$ qui doit être traitée sur ce processeur.

7.3 Solution optimale en régime permanent

7.3.1 Programme linéaire

Comme précédemment, nous exprimons le calcul du débit optimal en régime permanent, en écrivant des contraintes sur les quantités moyennes de tâches communiquées sur le réseau, et traitées sur les grappes.

Nous appelons $\alpha_l^{(k)}$ la quantité de travail de l'application $A^{(k)}$ effectuée sur la grappe G_l en une unité de temps, et $\beta_{k,l}$ le nombre de connexions ouvertes à partir de la grappe G_k pour envoyer du travail de l'application $A^{(k)}$ à la grappe G_l. Avec les notations précédentes, l'exécution de $\alpha_l^{(k)}$ unités de travail de l'application $A^{(k)}$ sur la grappe G_l prend un temps $\alpha_l^{(k)} \times \mathsf{flops}^{(k)}/s_l$. Afin de calculer le temps nécessaire à une communication entre les grappes G_k et G_l, nous calculons la bande-passante $b_{k,l}$ qui sera allouée à une connexion entre ces deux sites : c'est le minimum des bandes-passantes disponible sur chacun des liens longue-distance

traversés :

$$b_{k,l} = \min_{\lambda \in L_{k,l}} \mathsf{bw}(\lambda)$$

Cette quantité représente la bande-passante maximale allouée à une connexion, à cause des capacités limitées des liens longue-distance ; la congestion au niveau des liens locaux en sortie de G_k et en entrée de G_l doit également être prise en compte pour déterminer la bande-passante d'une communication, et donc la quantité maximale de travail de $A^{(k)}$ pouvant être déléguée à G_l.

Nous sommes maintenant en mesure d'énoncer des contraintes que doivent respecter les variables $\alpha_l^{(k)}$ et $\beta_{k,l}$ en régime permanent :

- Tout d'abord, le temps total de calcul d'une grappe G_k ne doit pas être supérieur à une unité de temps :

$$\forall G_k, \quad \sum_l \frac{\alpha_k^{(l)} \times \mathsf{flops}^{(l)}}{s_k} \leqslant 1$$

- La quantité de travail effectué sur la plate-forme pour l'application $A^{(k)}$ est la somme des débits pour cette application sur les différentes grappes :

$$\forall G_k, \quad \alpha^{(k)} = \sum_l \alpha_l^{(k)}$$

- D'autre part, la quantité de données transitant sur le lien de communication local reliant M_k à R_k est bornée par la capacité limitée de ce lien :

$$\forall G_k, \quad \underbrace{\sum_l \frac{\alpha_l^{(k)} \times \mathsf{size}^{(k)}}{b_k}}_{\text{communications sortantes}} + \underbrace{\sum_l \frac{\alpha_k^{(l)} \times \mathsf{size}^{(l)}}{b_k}}_{\text{communications entrantes}} \leqslant 1$$

- Enfin sur les liens longue-distance, il faut respecter les deux contraintes que nous avons présentées plus haut. D'abord le nombre de connexions utilisant ce lien est borné :

$$\forall \lambda \in \mathcal{B}, \quad \sum_{k,l, \text{ tels que } \lambda \in L_{k,l}} \beta_{k,l} \leqslant \mathsf{max\text{-}connect}(\lambda)$$

- La bande-passante d'une connexion est également limitée, chaque connexion utilisant un lien λ ne peut obtenir une bande-passante supérieure à $\mathsf{bw}(\lambda)$. En utilisant $b_{k,l}$, la bande passante maximale sur la route $G_k \to G_l$ introduite précédemment, on peut donc borner la quantité de données sur cette route :

$$\forall G_k, G_l, \quad \alpha_l^{(k)} \times \mathsf{size}^{(k)} \leqslant \beta_{k,l} \times b_{k,l}$$

Avec comme fonction objective la maximisation du débit équitable (comme dans le chapitre

précédent), on obtient le programme linéaire suivant :

$$
\begin{array}{l}
\text{MAXIMISER } \rho_{\text{opt}}, \\
\text{SOUS LES CONTRAINTES} \\
\left\{
\begin{array}{lll}
\text{(7.1a)} & \forall G_k, & \displaystyle\sum_l \frac{\alpha_k^{(l)} \times \mathsf{flops}^{(l)}}{s_k} \leqslant 1 \\
\text{(7.1b)} & \forall G_k, & \alpha^{(k)} = \displaystyle\sum_l \alpha_l^{(k)} \\
\text{(7.1c)} & \forall G_k, & \displaystyle\sum_l \frac{\alpha_l^{(k)} \times \mathsf{size}^{(k)}}{b_k} + \sum_l \frac{\alpha_k^{(l)} \times \mathsf{size}^{(l)}}{b_k} \leqslant 1 \\
\text{(7.1d)} & \forall \lambda \in \mathcal{B}, & \displaystyle\sum_{k,l,\lambda \in L_{k,l}} \beta_{k,l} \leqslant \mathsf{max\text{-}connect}(\lambda) \\
\text{(7.1e)} & \forall G_k, G_l, & \alpha_l^{(k)} \times \mathsf{size}^{(k)} \leqslant \beta_{k,l} \times b_{k,l} \\
\text{(7.1f)} & \forall G_k, G_l,\ \forall \lambda \in L_{k,l}, & b_{k,l} \leqslant \mathsf{bw}(\lambda) \\
\text{(7.1g)} & \forall A^{(k)} & \rho_{\text{opt}} \leqslant \dfrac{\alpha^{(k)}}{w^{(k)}} \\
\text{(7.1h)} & \forall G_k, G_l, & \alpha_l^{(k)} \geqslant 0,\ \beta_{k,l} \geqslant 0,\ \beta_{k,l} \in \mathbb{N}
\end{array}
\right.
\end{array}
\tag{7.1}
$$

Contrairement aux programmes linéaires que nous avons utilisés jusqu'à présent, ce programme linéaire comporte des variables entières (les $\beta_{k,l}$) et des variables rationnelles (les $\alpha_l^{(k)}$). La résolution d'un programme linéaire mixte (mêlant variables entières et rationnelles) est un problème NP-complet ; nous n'avons donc a priori pas de méthode de résolution efficace de ce programme linéaire.

Les variables entières que nous utilisons correspondent à des nombres de connexions ouvertes entre deux sites. Pour que la valeur objective du programme linéaire 7.1 soit bien une borne sur le débit atteignable, il faut contraindre un ordonnancement à utiliser un nombre constant de connexions entre deux sites.

Utiliser un nombre de connexions entre deux sites variables dans le temps consiste donc en une relaxation supplémentaire, en considérant des $\beta_{k,l}$ rationnels. Dans ce cas, la méthode présentée dans les chapitres précédents peut être appliquée : on peut résoudre le programme linéaire en rationnels, et reconstruire un ordonnancement périodique, dont la période est composée d'intervalles de temps, de telle sorte que le nombre de connexions est déterminé pour un intervalle de temps et change durant la période : c'est toujours ce que nous avons fait jusque là en décomposant les communications en couplages. Cependant, cette relaxation ne correspond pas au modèle que nous utilisons dans ce chapitre, qui suppose au contraire que les connexions utilisées sont d'un nombre constant dans le temps.

Théorème 7.1. *Le programme linéaire 7.1 donne une borne supérieure pour le débit équitable d'un ordonnancement d'applications divisibles de caractéristiques* $(\mathsf{flops}^{(k)}, \mathsf{size}^{(k)}, w^{(k)})$ *sur la plate-forme décrite par les paramètres* $(\mathcal{R}, \mathcal{B}, s, b, \mathsf{bw}, \mathsf{max\text{-}connect})$*, et qui utilise un nombre de connexions entre toute paire de grappes* (G_k, G_l) *constant dans le temps.*

Démonstration. Considérons un ordonnancement quelconque pour le problème de l'exécution d'applications divisibles sur cette plate-forme, qui utilise un nombre constant de connexions

entre chaque paire de sites. Pour toute paire de grappes (G_k, G_l), on note $\beta_{k,l}$ le nombre de connexions utilisées entre ces deux sites. Pour que l'ordonnancement soit valide, ces β ne doivent pas dépasser les valeurs définies pour les liens longue-distance :

$$\forall \lambda \in \mathcal{B}, \quad \sum_{k,l,\lambda \in L_{k,l}} \beta_{k,l} \leqslant \mathsf{max\text{-}connect}(\lambda)$$

On appelle $N_l^{(k)}(t)$ la quantité de travail de l'application $A^{(k)}$ traitée sur la grappe G_l. La quantité de travail de l'application $A^{(k)}$ effectuée en temps t est alors

$$\nu^{(k)}(t) = \sum_l N_l^{(k)}(t)$$

En un temps t, la grappe G_k ne peut pas effectuer plus de $s_k \times t$ unités de travail, donc on a :

$$\sum_l N_k^{(l)}(t) \times \mathsf{flops}^{(l)} \leqslant s_k \times t$$

De même, le lien de communication local reliant M_k à R_k peut transporter des tâches d'une taille au plus $b_k \times t$:

$$\sum_l N_l^{(k)}(t) \times \mathsf{size}^{(k)} + \sum_l N_k^{(l)}(t) \times \mathsf{size}^{(l)} \leqslant b_k \times t$$

En notant $b_{k,l}$ la bande-passante maximale sur la route $G_k \to G_l$, sachant que $\beta_{k,l}$ connexions sont utilisées sur cette route, on peut borner la quantité de données transmises de G_k à G_l :

$$\sum_l N_l^{(k)}(t) \times \mathsf{size}^{(k)} \leqslant b_{k,l} \times t$$

Ainsi, $\alpha_l^{(k)} = N_l^{(k)}(t)/t$ et $\beta_{k,l}$ forment une solution du programme linéaire 7.1, donc

$$\min_k \left\{ \frac{\nu^{(k)}(t)}{w^{(k)}} \right\} \leqslant \rho_{\text{opt}}.$$

Comme on peut faire ce raisonnement pour toute valeur de t, on a :

$$\limsup_{t \to \infty} \min_k \left\{ \frac{\nu^{(k)}(t)}{w^{(k)}} \right\} \leqslant \rho_{\text{opt}}$$

Donc le débit de l'ordonnancement considéré est inférieur à ρ_{opt}. ■

Reconstruction d'un ordonnancement Notons que contrairement à ce que nous avons fait jusqu'à présent, nous n'avons pas besoin de reconstruire un ordonnancement précis des calculs et des communications, grâce à la simplicité du modèle choisi. Nous avons en effet supposé que plusieurs communications pouvaient avoir lieu simultanément, et nous utilisons le modèle divisible pour les calculs : dans une période d'une unité de temps, nous pouvons donc supposer que la grappe G_k effectue $\alpha_k^{(l)}$ unités de travail pour l'application $A^{(l)}$, quantité de travail reçue pendant l'unité de temps précédente. Une allocation (α, β) est donc une solution complète du problème. Dans un modèle plus réaliste, comme celui des tâches indépendantes, il faudrait prendre en compte une certaine granularité, et reconstruire un ordonnancement comme nous l'avons fait dans les chapitres précédents. Cependant, il serait alors plus cohérent de modéliser une grappe de façon plus réaliste, comme un ensemble de machines en maître-esclaves, et nous perdrions la simplicité de notre modèle.

7.3.2 Difficulté du problème d'optimisation

Nous définissons le problème de décision associé au problème d'optimisation décrit dans le programme linéaire mixte 7.1.

Définition 7.1 (TACHES-DIVISIBLES-DISTRIBUÉES)**.** *Étant donnés une plate-forme décrite par les paramètres* $(\mathcal{R}, \mathcal{B}, s, b, \mathsf{bw}, \mathsf{max\text{-}connect})$*, un ensemble d'applications de caractéristiques* $(\mathsf{flops}^{(k)}, \mathsf{size}^{(k)}, w^{(k)})$ *et une borne* K*, existe-t-il une allocation* (α, β) *respectant les contraintes du programme linéaire 7.1, et réalisant un débit équitable* $\rho \geqslant K$*.*

Théorème 7.2. *Le problème TACHES-DIVISIBLES-DISTRIBUÉES est NP-complet.*

Démonstration. Vérifions tout d'abord que ce problème est bien NP. Étant donnée une allocation (α, β), nous pouvons vérifier que les contraintes du programme linéaire 7.1 sont satisfaites, et que le débit réalisé est supérieur ou égal à K.

Pour montrer la NP-complétude de ce problème, nous utilisons une réduction à partir du problème ENSEMBLE-INDÉPENDANT-MAXIMAL, connu comme un problème NP-complet [51, problème MAXIMUM-INDEPENDENT-SET]. Nous considérons une instance arbitraire $\mathcal{I}_1$ de ENSEMBLE-INDÉPENDANT-MAXIMAL : étant donnés un graphe non-orienté $G = (V, E)$ et une borne entière B, existe-t-il un sous-ensemble V' de V de cardinal au moins B, tel que deux sommets de V' ne sont pas reliés par une arête de E ? La figure 7.4 représente une telle instance. Nous construisons alors une instance $\mathcal{I}_2$ de notre problème comme suit :

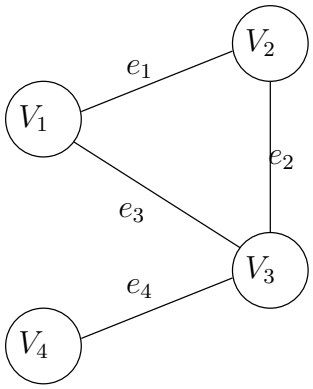


FIG. 7.4 – Exemple de graphe pour l'instance $\mathcal{I}_1$ du problème ENSEMBLE-INDÉPENDANT-MAXIMAL

- Notons $V = \{V_1, \ldots, V_n\}$. La plate-forme de $\mathcal{I}_2$ consiste en $n+1$ grappes $G_0, G_1, \ldots, G_n$. Nous construisons également un ensemble $route(k)$ pour chaque grappe G_k, pour l'instant vide et qui nous servira à déterminer les routes entre deux grappes.
  La grappe G_0 a un lien local (M_0, R_0), de bande-passante $g_0 = n$ et une puissance de calcul nulle s_0, alors que pour les autres grappes G_k $(k > 1)$, $g_k = s_k = 1$.
- Notons $E = \{e_1, \ldots, e_m\}$. Pour chaque arête $e_k = (V_i, V_j)$, nous ajoutons à la plate-forme :
  - deux routeurs Q_k^a et Q_k^b dans $\mathcal{R}$
  - un lien longue-distance distance entre eux $\lambda_k^{ab} = (Q_k^a, Q_k^b)$ dans $\mathcal{B}$, avec $\mathsf{max\text{-}connect}(\lambda_k^{ab}) = 1$ et $\mathsf{bw}(\lambda_k^{ab}) = 1$.
  - nous ajoutons l'élément k à $route(i)$ et $route(j)$

– Maintenant que les ensembles $route(i)$ sont construits, nous les utilisons pour calculer le routage entre deux grappes. Dans l'instance que nous construisons, il n'y a de routes qu'entre G_0 et les autres grappes. Pour chaque ensemble $route(i) = \{k_1, k_2, \ldots, k_{|route(i)|}\}$, nous commençons par rajouter les liens longue-distance suivants :
  – $\lambda^i_1 = (R_0, Q^a_{k_1})$
  – $\lambda^i_j = (Q^b_{k_{j-1}}, Q^a_{k_j})$ pour $j = 2 \ldots |route(i)|$
  – $\lambda^i_{|route(i)+1|} = (Q^b_{k_{|route(j)|}}, R_i)$

  Tout ceci nous permet de définir l'ensemble de liens qui composent la route de G_0 à G_i :

$$L_{0,i} = \left\{ \lambda^i_1, \lambda^{ab}_1, \lambda^i_2, \lambda^{ab}_2, \ldots, \lambda^{ab}_{|route(i)|}, \lambda^i_{route(|i|)+1} \right\}$$

  ce qui signifie que la route de G_0 à G_i passe par les routeurs suivants :

$$R_0 \to Q^a_{k_1} \to Q^b_{k_1} \to Q^a_{k_2} \to Q^b_{k_2} \to \cdots \to Q^a_{|route(i)|} \to Q^b_{|route(i)|} \to R_i$$

  La plate-forme de $\mathcal{I}_2$ obtenue à partir de l'instance $\mathcal{I}_1$ décrite sur la figure 7.4 est représentée sur la figure 7.5.

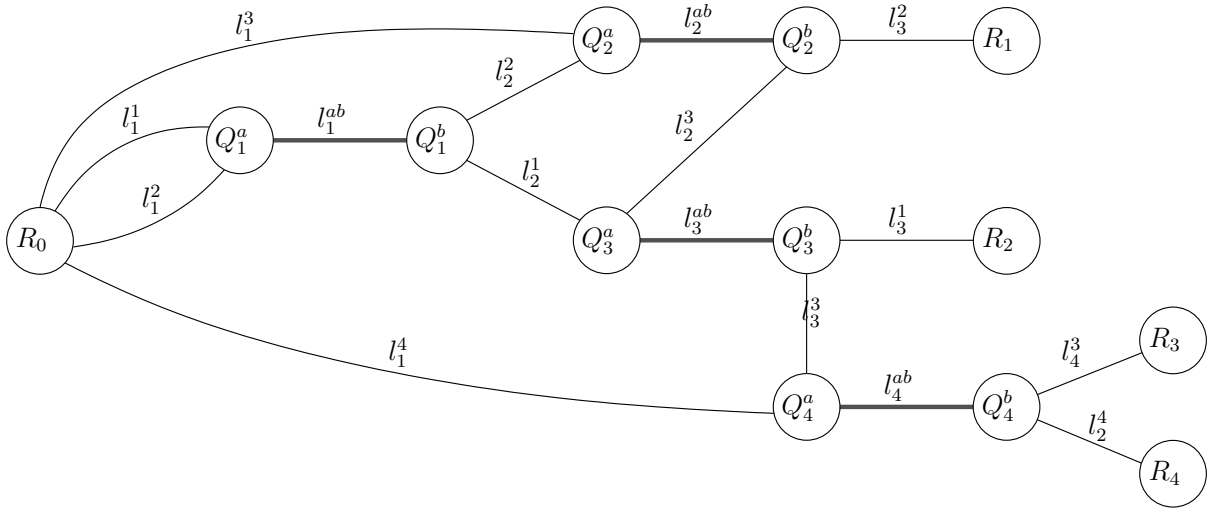


FIG. 7.5 – Plate-forme de l'instance $\mathcal{I}_2$ de notre problème correspondant à l'instance $\mathcal{I}_1$ de la figure 7.4. Les liens partagés λ^{ab}_k (correspondant aux arêtes e_k dans l'instance $\mathcal{I}_1$) sont représentés avec des traits épais.

Pour finir de construire l'instance $\mathcal{I}_2$, nous choisissons les priorités $w^{(0)} = 1$ et $w^{(k)} = 0$ pour $k > 1$ (seule la grappe G_0 possède du travail, qu'elle va chercher à distribuer aux autres), les caractéristiques de l'application $A^{(0)}$ sont $\mathsf{size}^{(0)} = 1$ et $\mathsf{flops}^{(0)} = 1$, et la borne sur le débit vaut $K = B$.

Nous commençons par établir une propriété de la plate-forme que nous venons de construire :

Lemme 7.1. *Dans la plate-forme de l'instance $\mathcal{I}_2$, deux routes (G_0, G_i) et (G_0, G_j) partagent un lien longue-distance si et seulement si l'arête (V_i, V_j) appartient au graphe G de l'instance $\mathcal{I}_1$.*

Démonstration. Supposons que l'arête $e_k = (V_i, V_j)$ appartienne au graphe G. Alors par construction, k a été ajouté aux ensembles $route(i)$ et $route(j)$, donc le lien longue-distance λ^{ab}_k appartient aux routes $L_{0,i}$ et $L_{0,j}$.

Supposons maintenant que les routes $L_{0,i}$ et $L_{0,j}$ partagent un même lien. $L_{0,i}$ est formé de deux type de liens : les λ_k^i qui ne sont partagés avec aucune autre route, et les λ_k^{ab}. Il existe donc k tel que λ_k^{ab} appartient à $L_{0,i}$ et à $L_{0,j}$. Par construction, ceci signifie que k appartient à $route(i)$ et à $route(j)$, donc que l'arête e_k relie V_i et V_j. ■

Nous montrons qu'il existe une solution à $\mathcal{I}_1$ si et seulement si il existe une solution à $\mathcal{I}_2$.

– Supposons qu'il existe un sous-ensemble indépendant V' de V, de taille au moins B. Nous construisons une allocation à partir de V' :

$$\begin{aligned} \forall G_i \neq G_0, \qquad & \alpha_i^{(0)} = \beta_{0,i} = \begin{cases} 1 & \text{si } V_i \in V' \\ 0 & \text{sinon} \end{cases} \\ \forall G_i \neq G_0, \forall G_j, \qquad & \alpha_j^{(i)} = \beta_{i,j} = 0 \\ & \alpha_0^{(0)} = 0. \end{aligned}$$

Comme V' est un ensemble indépendant, il n'existe pas d'arête de E entre deux sommets de V', donc il n'existe pas de lien longue-distance distance partagé par des connexions (G_i, G_j) tels que $\alpha_j^{(i)} = 1$. Chaque lien longue-distance est utilisé par au plus une connexion, et comme $\mathsf{max\text{-}connect}(l) = 1$ pour tout lien, la contrainte (7.1d) est respectée. Il y a $|V'| \leqslant n$ connexions utilisées sortant de G_0, et aucune entrante, donc comme $b_0 = n$, la contrainte (7.1c) est respectée en G_0. Pour les autres grappes, au plus une connexion entre dans chaque grappe, et $b_k = 1$ donc la contrainte est également vérifiée. Chaque grappe G_k utilisée doit calculer une unité de travail par unité de temps, et $s_k = 1$, donc la contrainte (7.1a) est vérifiée.
Ainsi (α, β) est une solution valide, de débit $|V'| \geqslant B = K$, donc $\mathcal{I}_2$ a une solution.

– Supposons maintenant que $\mathcal{I}_2$ admette une solution (α, β). Comme G_0 a une puissance de calcul nulle, son travail doit être délégué aux autres grappes. Chaque autre grappe à une puissance de calcul lui permettant de traiter une unité de travail par unité de temps, donc il existe au moins $K = B$ grappes utilisées pour traiter des tâches de l'application $A^{(0)}$, donc au moins K connexions utilisées depuis G_0. Comme $\mathsf{max\text{-}connect}(l)$ pour chaque lien longue-distance, deux routes utilisées ne peuvent partager un lien de communication. Donc pour tout couple de grappes utilisées G_i, G_j, les routes $L_{0,i}$ et $L_{0,j}$ ne partagent pas de lien longue-distance. D'après le lemme ci-dessus, pour tout couple de sommets V_i, V_j correspondants, le graphe G ne comporte par d'arête (V_i, V_j). L'ensemble de sommets V' correspondant aux grappes utilisées est donc de taille au moins B, et indépendant. On vérifie donc que V' est solution de $\mathcal{I}_1$. ■

7.4 Heuristiques

Comme nous ne pouvons pas résoudre directement le problème d'optimisation lié au programme linéaire, nous proposons des heuristiques pour trouver des solutions approchées.

7.4.1 Stratégie gloutonne par étapes (SGE)

La première stratégie que nous proposons procède par étapes, et alloue de nouvelles ressources pour une des K applications à chaque étape. Plus précisément, à chaque étape une

application $A^{(k)}$ est d'abord choisie, puis on choisit une grappe G_l, et enfin une quantité de travail de l'application $A^{(k)}$ à traiter sur la grappe G_l. Ces trois choix s'appuyent sur les intuitions suivantes :

- À un instant donné, on doit choisir l'application qui possède le plus petit débit pondéré, c'est à dire l'application $A^{(k)}$ qui réalise le minimum $\min_k \left\{ \frac{\alpha^{(k)}}{w^{(k)}} \right\}$. Cependant, lorsqu'on n'a encore alloué aucune ressource, tous les débits sont nuls. On choisit alors l'application de priorité maximale.
- Une fois l'application choisie, on compare l'intérêt d'effectuer du travail pour cette application localement, ou sur une autre grappe. On choisit la grappe (locale ou distante) la plus intéressante.
- Pour allouer une certaine quantité de travail, on prend garde de ne pas mobiliser toutes les ressources de calcul de la grappe choisie, afin de garder des ressources disponibles pour les autres applications.

Comme précédemment, on note $b_{k,l}$ la bande-passante disponible pour une connexion sur la route entre G_k et G_l :

$$g_{k,l} = \min_{\lambda \in L_{k,l}} \mathsf{bw}(\lambda)$$

Cette heuristique est décrite par l'algorithme suivant :

1. On note $L = \{A^{(1)}, \ldots, A^{(K)}\}$. On initialise $\alpha_l^{(k)} = 0$ et $\beta_{k,l} = 0$.
2. Si $L = \emptyset$, terminer.
3. **Choix de l'application** – On choisit l'application de L qui a le rapport $\alpha_k / w^{(k)}$ minimal. En cas d'égalité, on choisit l'application avec la plus grande priorité $w^{(k)}$. On note $A^{(k)}$ cette application.
4. **Choix de la grappe cible** – Pour toute grappe $G_m \neq G_k$, telle que pour tout lien longue-distance $\lambda \in L_{k,m}$ $\mathsf{max\text{-}connect}(\lambda) > 0$, on calcule le travail qui peut être effectué pour $A^{(k)}$ sur G_m en utilisant une seule connexion :

   $$\mathsf{gain}_m = \min \left\{ \frac{b_k}{\mathsf{size}^{(k)}}, \frac{b_{k,m}}{\mathsf{size}^{(k)}}, \frac{b_m}{\mathsf{size}^{(k)}}, \frac{s_m}{\mathsf{flops}^{(k)}} \right\}$$

   Pour les autres grappes distantes G_m vers lesquelles on ne peut plus ouvrir de connexion, on pose $\mathsf{gain}_m = 0$.
   Pour la grappe locale, on pose $\mathsf{gain}_k = \frac{s_k}{\mathsf{flops}^{(k)}}$. On choisit G_l tel que

   $$\mathsf{gain}_l = \max_m \{\mathsf{gain}_m\}$$

   Si $\mathsf{gain}_l = 0$ (ce n'est plus possible d'ajouter du travail pour l'application $A^{(k)}$), alors on supprime $A^{(k)}$ de l'ensemble L, et on retourne à l'étape 2.
5. **Choix de la quantité de travail** – Si $k \neq l$ (calcul distant) alors on pose $\mathsf{alloc} \leftarrow \mathsf{gain}_l$. Sinon, on choisit

   $$\mathsf{alloc} \leftarrow \max_{m \neq k} \left\{ \min \left\{ \frac{b_m}{\mathsf{size}^{(m)}}, \frac{b_{m,k}}{\mathsf{size}^{(m)}}, \frac{b_k}{\mathsf{size}^{(m)}}, \frac{s_k}{\mathsf{flops}^{(m)}} \right\} \right\}$$

   Ceci signifie que l'on n'attribue à l'application locale pas plus de travail que ce que l'on aurait attribué à une autre application s'effectuant sur $A^{(k)}$, pour éviter de surcharger la grappe G_k dans les premières étapes de l'algorithme.

6. **Mise à jour des variables**
   - On diminue la puissance de calcul de G_l : $s_l \leftarrow s_l - \mathsf{alloc} \times \mathsf{flops}^{(k)}$
   - On alloue le travail de $A^{(k)}$: $\alpha_l^{(k)} \leftarrow \alpha_l^{(k)} + \mathsf{alloc}$
   - S'il s'agit d'un calcul distant ($k \neq l$), on prend en compte la connexion utilisée, et on met à jour les caractéristiques du réseau :

$$\begin{aligned}
&\beta_{k,l} \leftarrow \beta_{k,l} + 1 \\
&\forall \lambda \in L_{k,l},\ \mathsf{max\text{-}connect}(\lambda) \leftarrow \mathsf{max\text{-}connect}(\lambda) - 1 \\
&b_k \leftarrow b_k - \mathsf{alloc} \times \mathsf{size}^{(k)} \\
&b_l \leftarrow b_l - \mathsf{alloc} \times \mathsf{size}^{(k)}
\end{aligned}$$

   Puis on recommence depuis l'étape 2.

7.4.2 Stratégies fondée sur le programme linéaire en rationnels

Le programme linéaire 7.1 avec des variables entières et d'autres rationnelles définit le débit optimal. Si on considère que toutes les variables sont des nombres rationnels, alors la valeur objective du programme linéaire définit une borne supérieure sur le débit, qui n'est pas nécessairement atteignable. En revanche, il est aisé de calculer cette borne supérieure, en utilisant une méthode de résolution classique. Nous proposons ici plusieurs heuristiques qui utilisent cette solution rationnelle du programme linéaire, et essayent de reconstruire une solution à valeurs $\beta_{k,l}$ entières.

Arrondi à l'entier inférieur (PLA)

La méthode la plus simple consiste à arrondir chaque valeur de $\beta_{k,l}$ dans la solution entière à l'entier immédiatement inférieur. Formellement, en notant $\widetilde{\alpha}_l^{(k)}$ et $\widetilde{\beta}_{k,l}$ une solution rationnelle du programme linéaire, on construit la solution :

$$\begin{aligned}
&\forall k,l && \beta_{k,l} = \left\lfloor \widetilde{\beta}_{k,l} \right\rfloor, \\
&\forall k,l && \alpha_l^{(k)} = \min\left\{ \widetilde{\alpha}_l^{(k)}, \left\lfloor \widetilde{\beta}_{k,l} \right\rfloor \times \min_{\lambda \in L_{k,l}} \mathsf{bw}(\lambda) \right\}
\end{aligned}$$

Comme $\widetilde{\alpha}_l^{(k)} \leqslant \alpha_l^{(k)}$, $\widetilde{\beta}_{k,l} \leqslant \beta_{k,l}$ et $\alpha_l^{(k)} \leqslant \beta_{k,l} \times \min_{\lambda \in L_{k,l}} \mathsf{bw}(\lambda)$, cette solution est valide. Nous nommons cette solution PLA, pour Progamme Linéaire Arrondi.

Arrondi et utilisation des capacités résiduelles (PLA+SGE)

Arrondir les valeurs des β rationnels à l'entier inférieur conduit naturellement à laisser des ressources de calcul et de communications inutilisées. Pour mettre à profit les ressources résiduelles, nous appliquons alors l'heuristique SGE précédemment développée.

Arrondi randomisé fondé sur le programme linéaire (ARPL)

De nombreuses méthodes ont été proposées pour résoudre un problème s'exprimant sous la forme d'un programme linéaire mixte, en utilisant une relaxation en rationnels. Parmi elles, on note l'utilisation d'une approximation randomisée. Cette approche est étudiée dans [62, chapter 11] par Motwani, Naor et Raghavan pour résoudre un problème proche du notre, le problème «multicommodity flow», consistant à router plusieurs flots concurrents dans un réseau de capacité limité. En utilisant les bornes de Chernoff, ils montrent que leur algorithme conduit, avec forte probabilité, à une solution de débit optimal. Ce résultat est plutôt de nature théorique : l'algorithme fournit soit une solution valide approchant (ou réalisant) le débit optimal, soit une solution de débit optimal, mais non valide. Dans ce cas, il faut alors modifier la solution pour la rendre valide.

Nous préférons adapter une méthode d'arrondi plus pratique qui en est inspirée, proposée par Coudert et Rivano dans [44] et utilisée pour l'affectation de fréquences dans les réseaux optiques. Nous nous inspirons de cette approche pour élaborer une heuristique d'arrondi randomisé, décrite ci-dessous.

1. Résoudre le programme linéaire 7.1 en rationnels. On appelle $(\widetilde{\alpha}, \widetilde{\beta})$ la solution obtenue.
2. Choisir aléatoirement une route (G_k, G_l) telle que $\widetilde{\beta}_{k,l} > 0$ avec probabilité uniforme.
3. Choisir aléatoirement $X_{k,l} \in \{0, 1\}$ avec une probabilité

$$P[X_{k,l} = 1] = \widetilde{\beta}_{k,l} - \left\lfloor \widetilde{\beta}_{k,l} \right\rfloor$$

4. Calculer la valeur $v = \left\lfloor \widetilde{\beta}_{k,l} \right\rfloor + X_{k,l}$ et ajouter la contrainte $\beta_{k,l} = v$ au programme linéaire.
5. S'il existe une route (G_k, G_l) à laquelle on n'a pas encore assignée de valeur $\beta_{k,l}$, recommencer depuis l'étape 1.

À une étape quelconque de l'algorithme, nous choisissons d'arrondir un $\widetilde{\beta}_{k,l}$ à la valeur entière inférieure ou supérieure. Si nous choisissons d'arrondir à la valeur supérieure, c'est que la probabilité $P[X_{k,l} = 1]$ est non nulle, c'est-à-dire que $\widetilde{\beta}_{k,l} > \left\lfloor \widetilde{\beta}_{k,l} \right\rfloor$. Les contraintes qui imposent une borne supérieure sur les $\beta_{k,l}$ sont les contraintes (7.1d) du programme linéaire, et elle ne font intervenir que des entiers. Donc si $\widetilde{\beta}_{k,l} > \left\lfloor \widetilde{\beta}_{k,l} \right\rfloor$ est une solution valide, alors il existe une solution valide avec $\widetilde{\beta}_{k,l} = \left\lfloor \widetilde{\beta}_{k,l} \right\rfloor + 1$. Si au contraire, nous arrondissons à la valeur entière inférieure, le choix conduit nécessairement à une solution valide. Une fois que toutes les variables $\beta_{k,l}$ ont été contraintes à des valeurs entières, on résout le programme linéaire pour trouver des valeurs de α. Donc l'allocation (α, β) construite par cette heuristique est valide.

On peut remarquer que cette heuristique nécessite de résoudre K^2 programmes linéaires, ce qui la rend plus coûteuse en calculs que les autres heuristiques utilisant le programme linéaire. Nous étudierons dans la partie suivante le temps de calcul de ces heuristiques.

7.5 Simulations

7.5.1 Méthodologie

Nous présentons dans cette partie les résultats des simulations comparant les heuristiques présentées ci-dessus. Nous voudrions pouvoir comparer ces heuristiques à la solution optimale, c'est à dire à la valeur objective du programme linéaire 7.1. Cependant, nous ne pouvons résoudre ce programme linéaire mixte (en entiers et rationnels) pour de nombreuses instances. Nous allons donc comparer les performances de nos heuristiques à la solution du programme linéaire en rationnels, qui constitue une borne supérieure sur le débit atteignable.

Afin de travailler sur des topologies de réseau réalistes, nous avons choisi d'utiliser le générateur de topologie Tiers (décrit dans [33] et déjà utilisé dans la partie 4.3). Les topologies créées par Tiers sont hiérarchiques, et comportent trois catégories de réseaux, connectés entre eux : les réseaux d'échelle mondiale (Wide Area Network – WAN), les réseaux intermédiaires (Metropolitan Area Network – MAN) et les réseaux locaux (Local Area Network – LAN). À l'aide de ce générateur, nous créons 100 topologies hiérarchiques, comportant chacune 40 nœuds WAN et 30 réseaux MAN comportant chacun environ 20 nœuds LAN. Nous ne générons pas de réseaux locaux (LAN), car notre modèle prévoit qu'ils sont regroupés en grappes, et identifiés à un seul nœud. Chaque topologie comporte environ 700 nœuds, et nous choisissons aléatoirement K grappes participantes parmi les nœuds LAN. Pour ces K nœuds, nous calculons le plus court chemin entre toute paire de nœuds, en nombre de liens traversés, et nous supprimons tous les nœuds qui ne se trouvent pas sur un plus court chemin, ne conservant que la topologie qui connecte les sites choisis. Dans la topologie simplifiée se trouvent des nœuds qui n'ont pas été choisis comme sites, mais qui se trouvent sur un plus court chemin entre deux sites ; ce sont des nœuds de $\mathcal{R}$ qui ne sont pas des routeurs locaux.

Une fois la topologie créée, nous affectons des caractéristiques aux ressources. Nous déterminons des intervalles de valeurs possibles pour chacun des paramètres. La bande-passante locale de chaque site b_k et la bande-passante des liens longue-distance $\mathsf{bw}(\lambda)$ sont choisies selon les mesures de bande-passante point à point dans l'Internet publiées dans [71]. Cette étude montre que le logarithme du taux de transmission observé forme une distribution normale centrée en log(2 000kbits/sec), avec un écart type d'environ log(10kbits/sec). Les autres paramètres sont générés avec des distributions uniformes dans les intervalles de valeurs décrits ci-dessous.

paramètre	distribution
K	$5, 7, \ldots, 90$
$\log(\mathsf{bw}(\lambda))$, $\log(b_k)$	normale (moyenne= log(2000), écart-type=log(10))
s_k	uniforme dans $[1000, 10\,000]$
$\mathsf{max\text{-}connect}(\lambda)$	uniforme dans $[1, 10]$
$\mathsf{size}^{(k)}$,$\mathsf{flops}^{(k)}$,$w^{(k)}$	uniforme dans $[1, 10]$

Pour chacune des topologies simplifiées obtenues avec Tiers, nous générons 10 configurations en instanciant les paramètres comme ci-dessus. Au total, nous utilisons près de 30 000 configurations pour nos tests.

7.5.2 Résultats

PLA Comme prévu, on observe dans les simulations que l'heuristique PLA donne de mauvais résultats. Dans la plupart des cas, une partie significative de la capacité du réseau est inutilisée, et dans certains cas, certains $\beta_{k,l}$ sont arrondis à zéro, ce qui conduit à un débit équitable très faible.

SGE et PLA+SGE La comparaison entre ces deux heuristiques est plus intéressante : contrairement à ce que l'on pensait, SGE se comporte globalement mieux que PLA+SGE. Le rapport moyen du débit obtenu par SGE sur celui obtenu par PLA+SGE sur toutes les configurations est 1.18, et SGE obtient de meilleurs résultats que PLA+SGE dans 81% des cas. La figure 7.6 montre le rapport moyen entre le débit obtenu par les différentes heuristiques et la borne supérieure calculée à l'aide du programme linéaire en rationnels, pour chaque taille de configuration K. On voit que SGE obtient des résultats 5% à 10% meilleurs que PLA+SGE dans la plupart des cas, mais lorsque K devient grand, la différence n'est plus significative, et les deux heuristiques ont du mal à rester proche de la borne supérieure.

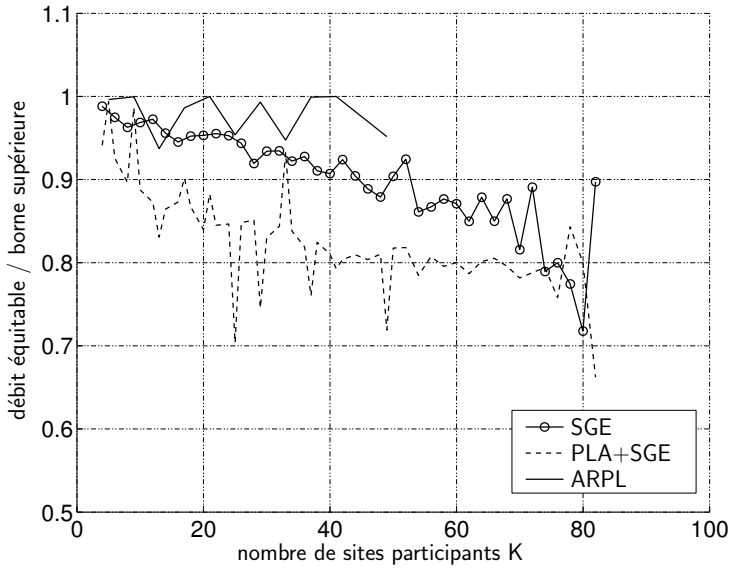


FIG. 7.6 – Comparaison des performances des trois meilleures heuristiques par rapport à la borne supérieure rationnelle.

Pour comprendre pourquoi notre heuristique PLA+SGE donne des résultats moins bons que l'heuristique SGE seule, nous avons étudié précisément les exécutions des simulations. En observant le résultat du programme linéaire en rationnels, nous constatons que dans de nombreux cas, des grappes délèguent la majorité de leur travail à une grappe distante, en utilisant un nombre de connexions rationnel strictement inférieur à un. Cette connexion partage des liens longue-distance avec d'autres connexions pour des grappes dans le même cas. Lors de la seconde phase, où l'heuristique SGE est utilisée pour recycler la capacité du réseau restante, certains de ces $\beta_{k,l}$ sont arrondis à un, d'autres à zéro. Les grappes qui voient leur nombre de connexions sortantes arrondi à zéro sont alors très défavorisées, et leur débit devient faible, ce qui conduit à un débit équitable faible. Au contraire, quand l'heuristique SGE est utilisée depuis le début, elle réussit à conserver une exécution équitable pour toutes les applications.

Il est intéressant de remarquer que les performances comparées de SGE et PLA+SGE dépendent de la topologie choisie. En effet, nous avons également effectué des simulations sur des graphes aléatoires (où deux nœuds quelconques sont reliés avec une probabilité constante). Sur ces topologies, l'heuristique PLA+SGE obtient en moyenne de meilleurs résultats que SGE. Bien que ces topologies ne représentent pas la réalité des réseaux d'interconnexion, ces résultats montrent que la topologie choisie affecte l'intérêt de l'une ou l'autre des heuristiques. Il serait intéressant de mener des expérimentations plus poussées afin de comprendre l'influence de la structure du réseau (inexistante dans les graphes aléatoire, forte dans les réseaux construits avec Tiers) ou de la connectivité du graphe sur les performances de ces heuristiques.

ARPL La figure 7.6 montre également les résultats de l'heuristique randomisée, qui donne de meilleurs résultats que les deux autres heuristiques, même pour un grand nombre de sites participants. Comme cette heuristique demande beaucoup plus de temps de calcul, nous ne l'avons testée que sur un petit nombre de topologies (62), et nous limitons la taille des topologies utilisées à 50 sites.

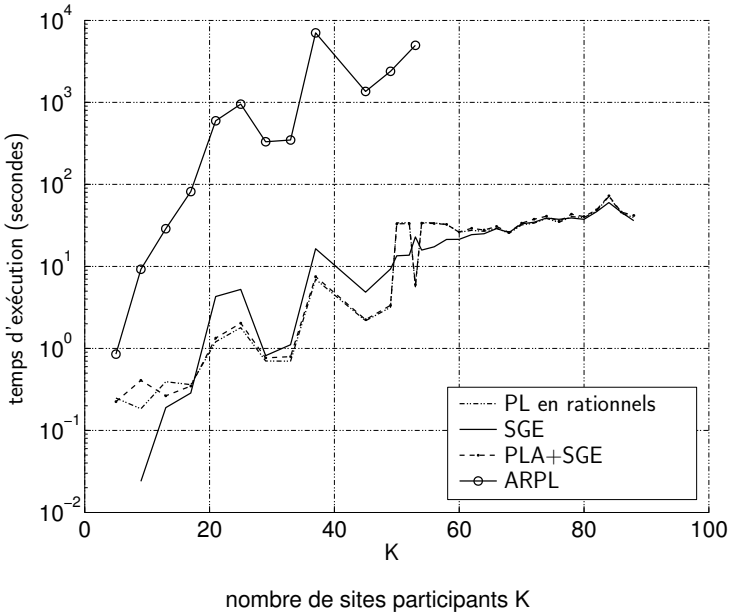


FIG. 7.7 – Temps d'exécution des heuristiques

Temps d'exécution La figure 7.7 montre les différents temps d'exécution de nos heuristiques en fonction du nombre de sites utilisés, sur un processeur Pentium à 1Ghz. On voit que l'heuristique ARPL est très coûteuse : pour $K = 50$ grappes impliquées, chaque exécution de cette heuristique dure environ une heure. En pratique, sauf à avoir des quantités de travail exceptionnellement importantes, on préférera donc utiliser les heuristiques SGE et PLA+SGE.

7.6 Conclusion

Dans ce chapitre, nous avons formulé un nouveau problème d'ordonnancement, pour des applications se présentant sous forme de tâches divisibles exécutées sur une grille de calcul. Nous avons particulièrement mis l'accent sur la modélisation du réseau d'interconnexion et du partage de la bande-passante, en essayant d'obtenir un modèle précis mais suffisamment simple pour pouvoir concevoir des algorithmes.

Nous avons montré que le problème d'ordonnancement était NP-complet si on ne s'autorise à utiliser qu'un nombre constant et borné de connexions entre chaque site, ce qui nous paraît l'hypothèse la plus raisonnable. Nous avons conçu des heuristiques pour résoudre ce problème, que nous avons testées à l'aide de nombreuses simulations, sur des plates-formes réalistes.

Notre modèle suppose qu'une seule application est présente sur chaque site, mais on peut facilement étendre ce travail à plusieurs applications par grappes, de caractéristiques différentes.

Cette étude pourrait être appliquée dans les grilles de calcul organisées sous la forme d'organisations virtuelles, comme proposé dans [50], sortes de communautés dans lesquelles les utilisateurs mettent en commun leurs ressources de calcul pour effectuer leurs applications. Nos heuristiques pourraient ainsi être implantées dans un ordonnanceur centralisé, situé sur un «ressource broker» auprès duquel chaque participant s'inscrit et soumet ses requêtes pour effectuer ses applications. L'optimisation du régime permanent permet d'obtenir des exécutions efficaces pour de grandes quantités de travail, ce qui est généralement le cas sur de telles plates-formes. De plus, l'utilisation de facteur de priorités sur chaque application permet de décrire la politique générale d'utilisation de la plate-forme, dictant par exemple que toutes les applications provenant d'un certain utilisateur doivent avoir une priorité double par rapport à celles d'un autre utilisateur. On peut même imaginer spécifier les priorités en fonction du site effectuant les calculs, avec des quantités $w_l^{(k)}$ qui décrivent les relations entre les différents participants. D'autres contraintes, du type «au plus 10% de la grappe G_k est utilisé pour effectuer des applications provenant de G_l» peuvent facilement être ajoutées au programme linéaire, pour décrire des politiques de partage plus spécifiques. Les heuristiques que nous proposons pourraient également être modifiées dans ce sens.

Dans notre modèle, l'ordonnanceur doit connaître le nombre maximum de connexions possibles sur chaque lien longue-distance. Il peut être difficile de connaître une telle information sur la plate-forme. Dans ce cas, ce paramètre peut être défini par l'administrateur de l'organisation virtuelle, limitant le nombre de connexion entre toute paire de sites à une même valeur. L'ordonnanceur centralisé doit également rassembler la vitesse de calcul des grappes utilisées, et les bandes-passantes des différents liens. Ceci peut être fait soit en utilisant des services existants sur les grilles, comme NWS [100] ou d'autres [47, 106], soit en mesurant le temps d'exécution des applications précédemment ordonnancées. On peut probablement combiner ces deux approches pour obtenir des informations plus fiables, comme dans [37]. Un tel ordonnanceur doit pouvoir s'adapter aux fluctuations des performances de la plate-forme. Ceci peut être fait régulièrement, par exemple au début de chaque période de l'ordonnancement, ou bien une solution peut être recalculée à chaque nouvelle requête de calcul.

Conclusion

Dans cette thèse, nous avons étudié différents problèmes d'ordonnancement sous l'angle de l'optimisation du régime permanent, dans le but de concevoir des algorithmes efficaces pour l'exécution d'applications distribuées sur des grilles de calcul. Nos principales contributions sont les suivantes :

- **Résultats théoriques pour les communications collectives** Avec le cadre d'étude présenté au chapitre 2 qui permet de décrire une primitive de communications collectives par un ensemble de schémas d'allocation, et un modèle de communication par un ensemble de schémas de communication, nous avons pu étudier au chapitre 3 la complexité des communications collectives pipelinées sous différents modèles. Nous avons en particulier montré que pour la distribution de données, la diffusion de données et la réduction, des algorithmes polynomiaux optimaux existe, pour les modèles un-port unidirectionnel et bidirectionnel. Nous avons en revanche montré au chapitre 4 que la diffusion restreinte et le calcul des préfixes sont des problèmes NP-complets avec ces deux mêmes modèles.

- **Algorithmes efficaces pour les communications collectives** À la fin du chapitre 3, dans la partie 3.2.1, nous avons proposé une méthode générale qui permet de séparer la recherche de schémas d'allocation de la recherche de couplages pour le modèle de communication un-port bidirectionnel. Au chapitre 4, nous avons utilisé cette méthode pour obtenir des algorithmes efficaces pour chacune des primitives dont nous avions montré que la complexité était polynomiale. Dans la partie 4.1, nous proposons un algorithme simple pour la distribution de données, proche de l'étude de Bertsimas et Gamarnik ; puis dans la partie 4.2, nous fournissons un algorithme plus complexe pour la diffusion, utilisant un résultat de décomposition d'un graphe en ensemble pondéré d'arbres ; enfin dans la partie 4.3, nous construisons un algorithme efficace pour la réduction, en développant un algorithme de décomposition d'une solution du programme linéaire en ensemble pondéré d'arbres de réduction.

- **Validation expérimentale pour les communications collectives** À de nombreuses reprises, nous avons illustré nos algorithmes par des simulations. Pour la réduction de données, nous avons comparé l'algorithme que nous proposons à un algorithme simple, qui construit un arbre de réduction comme le ferait MPICH, par des simulations sur plusieurs type de plate-forme. Ces simulations montrent l'intérêt d'utiliser une approche à plusieurs arbres qui tient compte de la topologie, comme la notre.
  Pour la diffusion de données, nous avons implanté plusieurs stratégies, à un ou plusieurs arbres de diffusion, s'appuyant sur le modèle un-port bidirectionnel ou sur le modèle multi-port borné, qui semble également réaliste dans ce contexte. Nous avons effectué

des tests sur la plate-forme Grid5000, qui montrent que la solution proposée dans le chapitre 4, fondée sur le modèle un-port bidirectionnel, donne les meilleurs résultats, et justifie le modèle utilisé dans le chapitre 4.

- **Ordonnancement d'applications multiples sur plate-forme distribuée** Dans la deuxième partie de ce manuscrit, nous avons étudié deux cas concrets d'applications distribuées sur une plate-forme hétérogène :
  - Dans le chapitre 6, nous nous sommes intéressés à l'exécution de plusieurs applications consistant chacune en un grand nombre de tâches indépendantes, sur une plate-forme maître-esclaves. Nous avons élaboré une stratégie centralisée, qui permet de calculer le débit optimal et de fournir un ordonnancement qui le réalise. Nous avons également caractérisé la solution optimale pour une plate-forme en étoile. Afin de prendre en compte les contraintes qui interviennent dans l'utilisation d'une plate-forme distribuée, comme le caractère dynamique des performances des ressources, nous avons également conçu des stratégies décentralisées. Nous avons mesuré l'impact du contrôle décentralisé, et du calcul distribué de l'ordonnancement à travers de nombreuses simulations.
  - Dans un deuxième temps, au chapitre 7, nous avons étudié l'ordonnancement de tâches divisibles sur une plate-forme de type «grille». Le modèle des tâches divisibles est une simplification supplémentaire du modèle des tâches indépendante, qui se justifie dans le modèle de plate-forme que nous proposons. Celui-ci tente de se saisir les caractéristiques essentielles des grilles de calcul organisées sous forme de grappes reliées entre elles, en s'intéressant particulièrement à la modélisation du réseau. Nous montrons dans ce chapitre que le problème d'optimisation considéré est NP-complet, et nous proposons plusieurs heuristiques pour le résoudre, que nous testons par simulation.

Ainsi, la relaxation en régime permanent introduite par Bertsimas et Gamarnik [26] permet de donner des solutions efficaces à des problèmes difficiles : la plupart des problèmes d'ordonnancement étudiés sous ce modèle deviennent polynomiaux si on cherche à optimiser le débit plutôt qu'à minimiser le temps d'exécution, que ce soit pour les communications collectives ou l'ordonnancement de tâches. Seuls quelques problèmes résistent encore, comme la diffusion restreinte, le calcul des préfixes, ou l'ordonnancement de graphes de tâches quelconques (comme nous l'avons montré dans [22]). Cependant, pour que la relaxation permette de faciliter l'étude de ces problèmes, il faut qu'elle soit totale, et que ne soient pas introduites des contraintes artificielles : pour le cas de la diffusion par exemple, il n'y a pas de raison de supposer que deux messages à diffuser emprunteront le même arbre de diffusion ; il faut donc considérer la solution comme un ensemble pondéré d'arbres, et non un seul arbre de diffusion, sous peine de se retrouver avec un problème NP-complet. D'autant qu'utiliser plusieurs arbres de diffusion permet d'améliorer le débit. Ceci se fait certes au prix d'un contrôle un peu plus complexe, mais cette difficulté n'est pas insurmontable, comme le montre notre implantation de la diffusion pipelinée.

Nous avons précisé en introduction que nous ne prenions pas les latences en compte, car la taille des messages était dictée par l'application : ainsi dans notre étude, $c_{i,j}$ représente le temps d'envoi d'un message sur une arête (i, j), latence comprise. Ce cadre de travail convient à l'étude de tâches indépendantes, ou de séries de messages à diffuser, distribuer, etc. Mais notre étude peut également fournir des algorithmes efficaces dans le cas d'un seul message de grande taille à diffuser (ou p messages de grande taille à distribuer, etc.). Dans ce cas, nous découpons le message à envoyer en un certain nombre N de petits messages, auxquels nous appliquons les

stratégies précédentes. Le choix de la taille du message va influencer le coût de la latence dans l'ordonnancement : sans découpage, nous ne payons cette latence qu'un nombre constant de fois (dépendant de la profondeur du graphe), mais nous ne pouvons espérer profiter du régime permanent, et plus le nombre de messages N augmente, plus le coût de la latence est important. Il faut alors revenir à la stratégie proposée par Bertsimas et Gamarnik, consistant à choisir N de l'ordre de$\sqrt{L}$, où L est la taille du message initial. Ainsi, le faible impact de la latence permettra de conserver un débit optimal, et le grand nombre de messages permettra d'atteindre ce débit, afin d'obtenir un ordonnancement asymptotiquement optimal.

Un des enseignements que nous pensons pouvoir tirer de cette thèse est qu'il est possible d'obtenir des garanties de performances pour des ordonnancements, même dans un contexte hétérogène, si on sait tirer parti des simplifications possibles dans la modélisation des applications. C'est ce que nous avons fait en considérant la maximisation du débit plutôt que la minimisation du temps d'exécution. Il est possible que certains problèmes NP-complets dans notre étude puissent être encore simplifiés, afin de pouvoir les étudier et leur fournir des solutions garanties. En particulier, calculer le débit d'une diffusion restreinte est un problème NP-complet avec notre formulation, mais si on s'autorise à faire des calculs afin de combiner différents messages, comme le fait le Network Coding, il est probable que ce problème puisse être résolu. Il faudrait alors réfléchir à comment distribuer les calculs de façon équitable, en fonction des nœuds participants et de leur puissance de calcul.

Nous pensons qu'un des enjeux majeurs pour la conception d'ordonnancement pour les grilles de calcul est de supporter la dynamicité des caractéristiques de la plate-forme. En particulier, des stratégies décentralisées sont plus à même de s'adapter à des conditions dynamiques, comme par exemple celles que nous avons développées au chapitre 6 : un calcul décentralisé de l'ordonnancement permet de prendre en compte rapidement un changement local dans la plate-forme. De ce point de vue, nous pensons aussi que l'algorithme d'Awerbuch et Leighton introduit à la fin de ce chapitre est prometteur : même si sa vitesse de convergence semble lente, cet algorithme peut naturellement s'adapter à un changement de la plate-forme. Bien que son adaptation ne soit démontrée que pour le cas où le débit initial est encore réalisable après les changements du réseau, si cette hypothèse n'est pas vérifiée, l'algorithme fournit un ensemble de pondérations sur les arêtes qui sont tout de même proches d'une solution optimale, et peuvent être utilisées en attendant que le nouveau débit optimal soit calculé.

Ainsi, même si les plates-formes de calcul que utilisons sont de plus en plus complexes à modéliser et à étudier, il reste encore des problèmes d'ordonnancement à résoudre, car la complexité des plates-formes impose des applications de plus en plus simples, pour lesquelles on peut espérer garantir des performances. Ceci oblige également à réfléchir du point de vue de l'algorithmique parallèle, pour concevoir des algorithmes efficaces sur des plates-formes hétérogènes, aux processeurs peu fiables, mais sur lesquelles les ressources de calcul disponibles sont abondantes.

Bibliographie

[1] M. Adler, Y. Gong et A. L. Rosenberg. «Optimal sharing of bags of tasks in heterogeneous clusters». Dans *15th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'03)* (2003), ACM Press, pp. 1–10. 8, 137

[2] R. Ahlswede, N. Cai, S.-Y. R. Li et R. W. Yeung. «Network information flow.». *IEEE Transactions on Information Theory* **46**, numéro 4 (2000), 1204–1216. 87

[3] A. Alexandrov, M. F. Ionescu, K. E. Schauser et C. J. Scheiman. «LogGP: incorporating long messages into the LogP model for parallel computation». *J. Parallel and Distributed Computing* **44**, numéro 1 (1997), 71–79. 7

[4] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming et S. Tuecke. «GridFTP: Protocol Extension to FTP for the Grid». Grid Forum Internet-Draft, March 2001. 139

[5] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela et M. Protasi. *Complexity and Approximation*. Springer, Berlin, Germany, 1999. 92

[6] B. Awerbuch et T. Leighton. «Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks». Dans *STOC '94: Proceedings of the 26h ACM symposium on Theory of Computing* (1994), ACM Press, pp. 487–496. 135

[7] B. Awerbuch et T. Leighton. «A simple local-control approximation algorithm for multi-commodity flow». Dans *FOCS '93: Proceedings of the 24th Conference on Foundations of Computer Science* (1993), IEEE Computer Society Press, pp. 459–468. 135

[8] M. Banikazemi, V. Moorthy et D. K. Panda. «Efficient collective communication on heterogeneous networks of workstations». Dans *Proceedings of the 27th International Conference on Parallel Processing (ICPP'98)* (1998), IEEE Computer Society Press. 7

[9] M. Banikazemi, J. Sampathkumar, S. Prabhu, D. Panda et P. Sadayappan. «Communication modeling of heterogeneous networks of workstations for performance characterization of collective operations». Dans *HCW'99, the 8th Heterogeneous Computing Workshop* (1999), IEEE Computer Society Press, pp. 125–133. 8

[10] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand et Y. Robert. «Scheduling strategies for master-slave tasking on heterogeneous processor platforms». *IEEE Trans. Parallel Distributed Systems* **15**, numéro 4 (2004), 319–330. 115, 128, 141

[11] A. Bar-Noy, S. Guha, J. S. Naor et B. Schieber. «Message multicasting in heterogeneous networks». *SIAM Journal on Computing* **30**, numéro 2 (2000), 347–358. 8

[12] A. Bar-Noy, S. Guha, J. Naor et B. Schieber. «Approximating the Throughput of Multiple Machines in Real-Time Scheduling.». *SIAM J. Comput.* **31**, numéro 2 (2001), 331–352. 9

[13] G. Barlas. «Collection-aware optimum sequencing of operations and closed-form solutions for the distribution of a divisible load on arbitrary processor trees». *IEEE Trans. Parallel Distributed Systems* **9**, numéro 5 (1998), 429–441. 141

[14] S. Bataineh, T. Hsiung et T.G.Robertazzi. «Closed form solutions for bus and tree networks of processors load sharing a divisible job». *IEEE Transactions on Computers* **43**, numéro 10 (octobre 1994), 1184–1196. 141

[15] O. Beaumont. *Nouvelles méthodes pour l'ordonnancement sur plates-formes hétérogènes*. Habilitation à diriger des recherches, Université de Bordeaux 1, décembre 2004. 13, 31, 34, 97

[16] O. Beaumont, V. Boudet, A. Petitet, F. Rastello et Y. Robert. «A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers)». *IEEE Trans. Computers* **50**, numéro 10 (2001), 1052–1070. 127

[17] O. Beaumont, L. Carter, J. Ferrante, A. Legrand et Y. Robert. «Bandwidth-centric allocation of independent tasks on heterogeneous platforms». Dans *International Parallel and Distributed Processing Symposium (IPDPS'2002)* (2002), IEEE Computer Society Press. 115, 128

[18] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal et Y. Robert. «Scheduling multiple bags of tasks on heterogeneous master-worker platforms: centralized versus distributed solutions». Research report, LIP, ENS Lyon, France, septembre 2005. 136

[19] O. Beaumont, H. Casanova, A. Legrand, Y. Robert et Y. Yang. «Scheduling divisible loads for star and tree networks: main results and open problems». Rapport technique RR-2003-41, LIP, ENS Lyon, France, septembre 2003. To appear in *IEEE Trans. Parallel and Distributed Systems*. 141

[20] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Pipelining broadcasts on heterogeneous platforms». Dans *International Parallel and Distributed Processing Symposium IPDPS'2004* (2004), IEEE Computer Society Press. 70

[21] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Complexity results and heuristics for pipelined multicast operations on heterogeneous platforms». Dans *2004 International Conference on Parallel Processing (ICPP'2004)* (2004), IEEE Computer Society Press. 92

[22] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogenous platforms». Research Report RR-2004-20, LIP, ENS Lyon, France, avril 2004. 97, 158

[23] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Optimizing the steady-state throughput of broadcasts on heterogeneous platforms heterogeneous platforms». Rapport technique, LIP, ENS Lyon, France, juin 2003. 70

[24] M. Berkelaar. «LP_SOLVE». http://www.cs.sunysb.edu/~algorith/implement/lpsolve/implement.shtml. 4, 64

[25] D. Bertsekas et R. Gallager. *Data Networks*. Prentice Hall, 1987. 116

[26] D. Bertsimas et D. Gamarnik. «Asymptotically optimal algorithm for job shop scheduling and packet routing». *Journal of Algorithms* **33**, numéro 2 (1999), 296–318. 2, 3, 5, 61, 158

[27] V. Bharadwaj et G. Barlas. «Efficient scheduling strategies for processing multiple divisible loads on bus networks». *Journal of Parallel and Distributed Computing* **62** (2002), 132–151. 137

[28] V. Bharadwaj, D. Ghose et T. Robertazzi. «Divisible load theory: a new paradigm for load scheduling in distributed systems». *Cluster Computing* **6**, numéro 1 (2003), 7–17. 137

[29] V. Bharadwaj, D. Ghose, V. Mani et T. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, 1996. 137

[30] P. Bhat, C. Raghavendra et V. Prasanna. «Adaptive communication algorithms for distributed heterogeneous systems». *Journal of Parallel and Distributed Computing* **59**, numéro 2 (1999), 252–279. 7

[31] P. Bhat, C. Raghavendra et V. Prasanna. «Efficient collective communication in distributed heterogeneous systems». Dans *ICDCS'99 19th International Conference on Distributed Computing Systems* (1999), IEEE Computer Society Press, pp. 15–24. 7

[32] M. den Burger, T. Kielmann et H. E. Bal. «Balanced Multicasting: High-throughput Communication for Grid Applications.». Dans *SC* (2005), p. 46. 9

[33] K. L. Calvert, M. B. Doar et E. W. Zegura. «Modeling Internet Topology». *IEEE Communications Magazine* **35**, numéro 6 (juin 1997), 160–163. Available at http://citeseer.nj.nec.com/calvert97modeling.html. 70, 78, 152

[34] L. Carter, H. Casanova, J. Ferrante et B. Kreaseck. «Autonomous Protocols for Bandwidth-Centric Scheduling of Independent-task Applications». Dans *International Parallel and Distributed Processing Symposium IPDPS'2003* (2003), IEEE Computer Society Press. 129

[35] H. Casanova. « Modeling Large-Scale Platforms for the Analysis and the Simulation of Scheduling Strategies». Dans *Proceedings of the 6th Workshop on Advances in Parallel and Distributed Computational Models (APDCM)* (April 2004). 138, 139

[36] H. Casanova. «Simgrid: A Toolkit for the Simulation of Application Scheduling». Dans *Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid'01)* (mai 2001), IEEE Computer Society. 83, 133

[37] H. Casanova et F. Berman. *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley, 2003, chapitre Parameter Sweeps on the Grid with APST. Hey, A. and Berman, F. and Fox, G., editors. 155

[38] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. I. T. Rowstron et A. Singh. «SplitStream: high-bandwidth multicast in cooperative environments.». Dans *SOSP* (2003), pp. 298–313. 9

[39] B. W. Char, K. O. Geddes, G. H. Gonnet, M. B. Monagan et S. M. Watt. *Maple Reference Manual*, 1988. 4, 64

[40] S. Charcranoon, T. Robertazzi et S. Luryi. «Optimizing computing costs using divisible load analysis». *IEEE Transactions on computers* **49**, numéro 9 (septembre 2000), 987–991. 137

[41] D.-M. Chiu, M. Kadansky, J. Provino et J. Wesley. «Experiences in Programming a Traffic Shaper». Dans *ISCC '00: Proceedings of the Fifth IEEE Symposium on Computers and Communications (ISCC 2000)* (Washington, DC, USA, 2000), IEEE Computer Society, p. 470. 142

[42] D. N. Chiu. «Some Observations on Fairness of Bandwidth Sharing». Rapport technique, Sun Microsystems, 1999. 138

[43] T. H. Cormen, C. E. Leiserson et R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990. 44, 45, 48, 69, 104

[44] D. Coudert et H. Rivano. «Lightpath assignment for multifibers WDM optical networks with wavelength translators». Dans *IEEE Global Telecommunications Conference (Globecom'02)* (2002), IEEE Computer Society Press. Session OPNT-01-5. 9, 151

[45] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian et T. von Eicken. «LogP: A practical model of parallel computation». *Communications of the ACM* **39**, numéro 11 (1996), 78–85. 7

[46] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian et T. v. Eicken. «LogP: Towards a realistic model of parallel computation». Dans *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (1993), ACM Press. 7

[47] K. Czajkowski, S. Fitzgerald, I. Foster et C. Kesselman. «Grid Information Services for Distributed Resource Sharing». Dans *Proceedings of the 10th IEEE Symposium on High-Performance Distributed Computing (HPDC-10)* (August 2001). 155

[48] S. Floyd et K. Fall. «Promoting the use of end-to-end congestion control in the Internet». *IEEE/ACM Transactions on Networking* **7**, numéro 4 (1999), 458–472. 138

[49] I. Foster et C. Kesselman. «Globus: A Metacomputing Infrastructure Toolkit». *International Journal of Supercomputer Applications* **11**, numéro 2 (1997), 115–128. 138

[50] I. Foster, C. Kesselman et S. Tuecke. «The Anatomy of the Grid: Enabling Scalable Virtual Organizations». *International Journal of High Performance Computing Applications* **15**, numéro 3 (2001). 155

[51] M. R. Garey et D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979. 21, 90, 103, 146

[52] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek et V. Sunderam. *PVM Parallel Virtual Machine: A Users'Guide and Tutorial for Networked Parallel Computing*. The MIT Press, 1996. 9

[53] D. Ghose et T. Robertazzi, éditeurs. *Special issue on* Divisible Load Scheduling (2003), Cluster Computing, 6, 1. 137

[54] C. Gkantsidis et P. R. Rodriguez. «Network Coding for Large Scale Content Distribution». Dans *INFOCOM* (2005). 87

[55] G. H. Golub et C. F. V. Loan. *Matrix computations*. Johns Hopkins, 1989. 40

[56] J. P. Goux, S. Kulkarni, J. Linderoth et M. Yoder. «An enabling framework for master-worker applications on the computational grid». Dans *Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC'00)* (2000), IEEE Computer Society Press. 115

[57] M. Grötschel, L. Lovász et A. Schrijver. *Geometric Algorithm and Combinatorial Optimization*. Algorithms and Combinatorics 2. Springer-Verlag, 1994. Second corrected edition. 43

[58] N. Hall, W.-P. Liu et J. Sidney. «Scheduling in broadcast networks». *Networks* **32**, numéro 14 (1998), 233–253. 7

[59] Harutyunyan et Shao. «An Efficient Heuristic for Broadcasting in Networks». *JPDC: Journal of Parallel and Distributed Computing* **66** (2006). 7

[60] E. Heymann, M. A. Senar, E. Luque et M. Livny. «Adaptive scheduling for master-worker applications on the computational grid». Dans *Grid Computing - GRID 2000* (2000), R. Buyya et M. Baker, éditeurs, Springer-Verlag LNCS 1971, pp. 214–227. 115

[61] T. Ho, M. Médard et R. Koetter. «An information theoretic view of network management.». Dans *INFOCOM* (2003). 87

[62] D. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997. 151

[63] B. Hong et V. Prasanna. «Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput». Dans *International Parallel and Distributed Processing Symposium IPDPS'2004* (2004), IEEE Computer Society Press. 8, 10, 99, 100

[64] N. T. Karohis, B. Toonen et I. Foster. «MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface». *Journal ofParallel and Distributed Computing* **63**, numéro 5 (2003), 551–563. 9

[65] R. M. Karp. «Reducibility among combinatorial problems». Dans *Complexity of Computer Computations* (NY, 1972), R. E. Miller et J. W. Thatcher, éditeurs, Plenum Press, pp. 85–103. 22, 47, 85

[66] L. G. Khachiyan. «A polynomial algorithm in linear programming». *Doklady Akademiia Nauk SSSR* **224** (1979), 1093–1096. English Translation: *Soviet Mathematics Doklady*, Volume 20, pp. 191–194. 6, 41

[67] S. Khuller, Y. Kim et Y. Wan. «On generalized gossiping and broadcasting». Dans *Proceedings of the 11th Annual European Symposium on Algorithms* (2003), Springer, éditeur, pp. 373–384. 7

[68] T. Kielmann, H. E. Bal, S. Gorlatch, K. Verstoep et R. F. Hofman. «Network performance-aware collective communication for clustered wide area systems». *Parallel Computing* **27**, numéro 11 (2001), 1431–1456. 7

[69] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat et R. A. F. Bhoedjan. «MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems». Dans *Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming PPoPP'99* (Atlanta, GA, 1999), pp. 131–140. 10, 112

[70] B. Kreaseck. *Dynamic autonomous scheduling on Heterogeneous Systems*. Thèse de doctorat, University of California, San Diego, 2003. 115, 129

[71] C. Lee et J.Stepanek. «On Future Global Grid Communications Performance». Dans *HCW'2001, the 10th Heterogeneous Computing Workshop* (2001), IEEE Computer Society Press. 152

[72] A. Legrand. *Algorithmique parallèle hétérogène et techniques d'ordonnancement : approches statiques et dynamiques*. Thèse de doctorat, École Normale Supérieure de Lyon, décembre 2003. 13, 97

[73] A. Legrand, L. Marchal et H. Casanova. «Scheduling Distributed Applications: The SimGrid Simulation Framework». Dans *Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)* (May 2003). 83, 133

[74] R. Libeskind-Hadas, D. Mazzoni et R. Rajagopalan. «Optimal Contention-Free Unicast-Based Multicasting in Switch-Based Networks of Workstations». Dans *IPPS/SPDP* (1998), pp. 358–364. 7

[75] C. Lin. «Efficient Contention-Free Broadcast in Heterogeneous Network of Workstations with Multiple Send and Receive Speeds». Dans *ISCC* (2003), IEEE Computer Society, pp. 1277–1284. 7

[76] P. Liu. «Broadcast scheduling optimization for heterogeneous cluster systems». *Journal of Algorithms* **42**, numéro 1 (2002), 135–152. 7

[77] P. Liu et T.-H. Sheng. «Broadcast scheduling optimization for heterogeneous cluster systems». Dans *SPAA'2000, 12th Annual ACM Symposium on Parallel Algorithms and Architectures* (2000), ACM Press, pp. 129–136. 7

[78] B. Lowekamp et A. Beguelin. «ECO: Efficient collective operations for communication on heterogeneous networks». Dans *10th International Parallel and Distributed Processing Symposium (IPDPS'96)* (1996), IEEE Computer Society Press. 9, 112

[79] L. Massoulié et J. Roberts. «Bandwidth Sharing: Objectives and Algorithms». *Transactions on Networking* **10**, numéro 3 (june 2002), 320–328. 116, 138

[80] M. Mathis, J. Semke et J. Mahdavi. «The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm». *Computer Communications Review* **27**, numéro 3 (1997). 138

[81] A. Munier. *Contribution à l'étude des ordonnancements cycliques*. Thèse de doctorat, Université Paris 6, février 1991. 31

[82] J. Padhye, V. Firoiu, D. Towsley et J. Kurose. «Modeling TCP throughput: a simple model and its empirical validation». Dans *SIGCOMM '98* (New York, NY, USA, 1998), ACM Press, pp. 303–314. 111

[83] T. Robertazzi. «Processor equivalence for a linear daisy chain of load sharing processors». *IEEE Trans. Aerospace and Electronic Systems* **29** (1993), 1216–1221. 141

[84] T. Robertazzi. «Ten reasons to use Divisible Load Theory». *IEEE Computer* **36**, numéro 5 (2003), 63–68. 137

[85] A. L. Rosenberg. «On sharing bags of tasks in heterogeneous networks of workstations: greedier is not better». Dans *Cluster Computing 2001* (2001), IEEE Computer Society Press, pp. 124–131. 8, 137

[86] T. Saif et M. Parashar. «Understanding the behavior and performance of non-blocking communications in MPI». Dans *Proceedings of Euro-Par 2004: Parallel Processing* (2004), LNCS 3149, Springer, pp. 173–182. 10, 99

[87] S. Santini. «We Are Sorry to Inform You ...». *Computer* **38**, numéro 12 (2005), 128–127. 1

[88] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 des *Algorithms and Combinatorics*. Springer-Verlag, 2003. 6, 41, 56, 68, 69

[89] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, New York, 1986. 40

[90] C. Shannon. «A Mathematical Theory of Communication». *Bell System Technical Journal* **27** (1948). 1

[91] G. Shao. *Adaptive scheduling of master/worker applications on distributed computational resources*. Thèse de doctorat, Dept. of Computer Science, University Of California at San Diego, 2001. 115

[92] G. Shao, F. Berman et R. Wolski. «Master/slave computing on the grid». Dans *Heterogeneous Computing Workshop HCW'00* (2000), IEEE Computer Society Press. 115

[93] J. Sohn, T. Robertazzi et S. Luryi. «Optimizing computing costs using divisible load analysis». *IEEE Transactions on parallel and distributed systems* **9**, numéro 3 (mars 1998), 225–234. 137

[94] A. Srinivasan et C.-P. Teo. «A Constant-Factor Approximation Algorithm for Packet Routing and Balancing Local vs. Global Criteria.». *SIAM J. Comput.* **30**, numéro 6 (2000), 2051–2068. 3

[95] R. Thakur et W. Gropp. «Improving the Performance of Collective Operations in MPICH.». Dans *PVM/MPI* (2003), pp. 257–267. 7

[96] The MuPAD Group (B. Fuchssteiner et al.). *MuPAD User's Manual*. John Wiley and sons, 1996. 4, 64

[97] J. Watts et R. Van De Geijn. «A pipelined broadcast for multidimensional meshes». *Parallel Processing Letters* **5**, numéro 2 (1995), 281–292. 7

[98] J. B. Weissman. «Scheduling multi-component applications in heterogeneous wide-area networks». Dans *Heterogeneous Computing Workshop HCW'00* (2000), IEEE Computer Society Press. 115

[99] D. B. West. *Introduction to Graph Theory*. Prentice Hall, 1996. 69

[100] R. Wolski, N. Spring et J. Hayes. «The network weather service: a distributed resource performance forecasting service for metacomputing». *Future Generation Computer Systems* **15**, numéro 10 (1999), 757–768. 155

[101] H. Wong, D. Yu, , V. Bharadwaj et T. Robertazzi. «Data intensive grid scheduling: multiple sources with capacity constraints». Dans *PDCS'2003, 15th Int'l Conf. Parallel and Distributed Computing and Systems* (2003), IASTED Press. 137

[102] D. Yu et T. Robertazzi. «Divisible load scheduling for grid computing». Dans *PDCS'2003, 15th Int'l Conf. Parallel and Distributed Computing and Systems* (2003), IASTED Press. 138

[103] «Berkeley Open Infrastructure for Network Computing». `http://boinc.berkeley.edu`. 115

[104] «CDF Analysis Farms (CAF)». `http://cdfcaf.fnal.gov/`. 137

[105] «Collider Detector at Fermilab (CDF)». `http://www-cdf.fnal.gov/`. 1

[106] «The Ganglia Project». `http://ganglia.sourceforge.net`. 155

[107] «Large Hadron Collider (LHC)». `http://lhc.web.cern.ch/lhc/`. 1

[108] «The Network Simulator - ns-2». `http://www.isi.edu/nsnam/ns`. 138

Annexe A

Notations

Communications collectives

$G = (V, E)$: graphe de plate-forme, où les nœuds de V sont les processeurs et les arêtes de E sont les liens de communication
$c_{i,j} = c_{(P_i,P_j)}$: temps de communication d'un message de taille unité sur le lien (P_i, P_j)
z_i : temps de calcul d'une tâche de taille unité sur P_i
$\mathcal{A}$: ensemble des schémas d'allocation
$\mathcal{C}$: ensemble des schémas de communication

Distribution de données

P_{source} : processeur source de la distribution
V_{cibles} : ensembles de processeurs cibles
m_k : type des messages à destination de P_k
$\mathsf{send}(P_i \to P_j, m_k)$: nombre de messages à destination de P_k transmis sur l'arête (i, j) par unité de temps

Diffusion

P_{source} : processeur source de la distribution
m_k : type des messages à destination de P_k
$\mathsf{send}(P_i \to P_j, m_k)$: nombre de messages à destination de P_k transmis sur l'arête (i, j) par unité de temps
$\mathsf{send}(P_i \to P_j, m_k)$: nombre total de messages transmis sur l'arête (i, j) par unité de temps

Réduction

$\mathcal{R}$: processeurs participant à la réduction
v_i : valeur initiale du processeur $P_{r_i} \in \mathcal{R}$
P_{cible} : cible de la réduction
$v_{[k,m]}$: résultat de la réduction partielle $v_k \oplus \cdots \oplus v_m$
$T_{k,l,m}$: tâche de calcul correspondant à l'opération $v_{[k,m]} = v_{[k,l]} \oplus v_{[l+1,m]}$
$\mathsf{size}([k,m])$: taille d'un message contenant un résultat partiel $v_{[k,m]}$
$\mathsf{flops}(k,l,m)$: taille de la tâche de calcul $T_{k,l,m}$
$\mathsf{send}(P_i \to P_j, v_{[k,m]})$: nombre de messages $v_{[k,m]}$ transmis sur l'arête (i,j) en une unité de temps
$\mathsf{cons}(P_i, T_{k,l,m})$: nombre de tâches $T_{k,l,m}$ effectuée sur le processeur P_i

Diffusion restreinte

P_{source} : processeur source de la distribution
V_{cibles} : ensembles de processeurs cibles
m_k : type des messages à destination de P_k
$\mathsf{send}(P_i \to P_j, m_k)$: nombre de messages à destination de P_k transmis sur l'arête (i,j) par unité de temps
$\mathsf{send}(P_i \to P_j, m_k)$: nombre de messages à destination de P_k transmis sur l'arête (i,j) par unité de temps

Ordonnancement d'applications multiples

Chapitre 6 – Collections de tâches indépendantes en maître-esclave

$A^{(k)}$: application k
$w^{(k)}$: priorité de l'application k
$\nu^{(k)}(t)$: nombre de tâches de l'application k effectuées en temps t
$A^{(k)}$: débit de l'application k
$\mathsf{flops}^{(k)}$: quantité de calcul liée à une tâche de l'application k
$\mathsf{size}^{(k)}$: taille du fichier d'entrée d'une tâche de l'application k
$\mathsf{size}^{(k)}_{\mathrm{res}}$: taille du fichier de sortie d'une tâche de l'application k
$r^{(k)}$: rapport communication sur calcul pour l'application k ($\mathsf{size}^{(k)}/\mathsf{flops}^{(k)}$)
$\mathsf{send}^{(k)}_{P_i \to P_j}$: quantité moyenne de tâches de l'application k envoyées de P_i à P_j pendant une unité de temps
$\alpha_i^{(k)}$: quantité moyenne de tâches de l'application k traitées sur P_i pendant une unité de temps

Chapitre 7 – Tâches divisibles pour la grille

$A^{(k)}$, $w^{(k)}$, $\alpha^{(k)}$, $\nu^{(k)}(t)$: comme ci-dessus

$\mathsf{flops}^{(k)}$: quantité de calcul liée à une unité de travail de l'application k

$\mathsf{size}^{(k)}$: quantité de données nécessaire à l'exécution d'une unité de travail de l'application k

$\mathcal{G}_{\text{ic}} = (\mathcal{R}, \mathcal{B})$: graphe dont les sommets ($\mathcal{R}$) sont les routeurs, et les arêtes ($\mathcal{B}$) sont les liens longue-distance

$\mathsf{bw}(l)$: bande-passante maximale allouée à toute connexion utilisant le lien longue-distance l

$\mathsf{max\text{-}connect}(l)$: nombre maximal de connexion pouvant être utilisées simultanément sur le lien longue-distance l

M_k : processeur frontal de la grappe G_k

R_k : routeur connectant la grappe G_k au reste du réseau

s_k : vitesse de calcul de la grappe G_k

b_k : capacité du lien local (M_k, R_k)

$\alpha_l^{(k)}$: quantité moyenne de travail de l'application k effectué sur la grappe G_l

$\beta_{k,l}$: nombre de connexion ouvertes pour déléguées du travail de la grappe G_k à la grappe G_l

Annexe B

Liste des publications

Revues internationales avec comité de lecture

[1] O. Beaumont, L. Marchal et Y. Robert. «Complexity results for collective communications on heterogeneous platforms». *Int. Journal of High Performance Computing Applications* (2006, à paraître). 13

[2] L. Marchal, Y. Yang, H. Casanova et Y. Robert. «Steady-state scheduling of multiple divisible load applications on wide-area distributed computing platforms». *Int. Journal of High Performance Computing Applications* (2006, à paraître). 14

[3] A. Legrand, L. Marchal et Y. Robert. «Optimizing the steady-state throughput of scatter and reduce operations on heterogeneous platforms». *Journal of Parallel and Distributed Computing* **65**, numéro 12 (2005), 1497–1514. 13

[4] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Steady-state scheduling on heterogeneous clusters». *Int. J. of Foundations of Computer Science* **16**, numéro 2 (april 2005). 13

[5] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Pipelining broadcasts on heterogeneous platforms». *IEEE Trans. Parallel Distributed Systems* **16**, numéro 4 (2005). 13

[6] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Scheduling strategies for mixed data and task parallelism on heterogeneous clusters». *Parallel Processing Letters* **13**, numéro 2 (2003). 13

Conférences internationales avec comité de lecture

[7] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal et Y. Robert. «Centralized versus distributed schedulers for multiple bag-of-task applications». Dans *International Parallel and Distributed Processing Symposium IPDPS'2006* (2006), IEEE Computer Society Press. 14

[8] O. Beaumont, L. Marchal, V. Rehn et Y. Robert. «FIFO scheduling of divisible loads with return messages under the one-port model». Dans *HCW'2006, the 15th Heterogeneous Computing Workshop* (2006), IEEE Computer Society Press. 15

[9] L. Marchal, P. V.-B. Primet, Y. Robert et J. Zeng. «Optimal Bandwidth Sharing in Grid Environment». Dans *The 15th IEEE International Symposium on High Performance Distributed Computing (HPDC'2006)* (2006, to appear), IEEE Computer Society Press. 15

[10] O. Beaumont, L. Marchal et Y. Robert. «Scheduling divisible loads with return messages on heterogeneous master-worker platforms». Dans *International Conference on High Performance Computing HiPC'2005* (2005, to appear), LNCS, Springer Verlag. 15

[11] L. Marchal, P. V.-B. Primet, Y. Robert et J. Zeng. «Optimizing Network Resource Sharing in Grids». Dans *IEEE Global Telecommunications Conference (Globecom'2005)* (2005, to appear), IEEE Computer Society Press. 15

[12] L. Marchal, Y. Yang, H. Casanova et Y. Robert. «A realistic network/application model for scheduling divisible loads on large-scale platforms». Dans *International Parallel and Distributed Processing Symposium IPDPS'2005* (2005), IEEE Computer Society Press. 14

[13] O. Beaumont, L. Marchal et Y. Robert. «Broadcast Trees for Heterogeneous Platforms». Dans *International Parallel and Distributed Processing Symposium IPDPS'2005* (2005), IEEE Computer Society Press. 13

[14] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Independent and Divisible Tasks Scheduling on Heterogeneous Star-schaped Platforms with Limited Memory». Dans *13th Euromicro Conference on Parallel, Distributed and Network-based Processing PDP'2005* (2005), IEEE Computer Society Press, pp. 179–186. 14

[15] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Pipelining broadcasts on heterogeneous platforms». Dans *International Parallel and Distributed Processing Symposium IPDPS'2004* (2004), IEEE Computer Society Press. 13

[16] A. Legrand, L. Marchal et Y. Robert. «Optimizing the steady-state throughput of scatter and reduce operations on heterogeneous platforms». Dans *APDCM'2004, 6th Workshop on Advances in Parallel and Distributed Computational Models* (2004), IEEE Computer Society Press. 13

[17] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Complexity results and heuristics for pipelined multicast operations on heterogeneous platforms». Dans *Proceedings of the 33rd International Conference on Parallel Processing (ICPP'04)* (2004), IEEE Computer Society Press. 13

[18] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Steady-state scheduling on heterogeneous clusters : why and how ?». Dans *6th Workshop on Advances in Parallel and Distributed Computational Models APDCM 2004* (2004), IEEE Computer Society Press. 13

[19] H. Casanova, A. Legrand et L. Marchal. «Scheduling Distributed Applications : the SimGrid Simulation Framework». Dans *Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)* (may 2003).

[20] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogeneous platforms». Dans *HeteroPar'2004 : International Conference on Heterogeneous Computing, jointly published with ISPDC'2004 : International Symposium on Parallel and Distributed Computing* (2004), IEEE Computer Society Press. 13

Conférences nationales avec comité de lecture

[21] O. Beaumont, A.-M. Kermarrec, L. Marchal et Étienne Rivière. «Voronet, un réseau objet-à-objet sur le modèle petit-monde». Dans *CFSE'5 : Conférence Française sur les Systèmes d'Exploitation* (2006). 15

Résumé :
Les travaux présentés dans cette thèse concernent l'ordonnancement pour les plates-formes hétérogènes à grande échelle. Nous nous intéressons principalement aux opérations de communications collectives qui interviennent lors de l'exécution d'applications distribuées : diffusion de données, distribution de données, réduction, etc. Nous étudions ces problèmes dans le cadre de leur régime permanent, en optimisant le débit d'une série d'opérations de communications, en vue d'obtenir un ordonnancement asymptotiquement optimal du point de vue du temps d'exécution total. Nous présentons un cadre général pour l'étude de ces opérations, qui nous permet d'établir que la complexité du problème est polynomiale pour la plupart des opérations de communications collectives. Pour un modèle de communication particulier, le modèle un-port bidirectionnel, nous proposons une méthode de résolution pratique que nous appliquons à la diffusion, la distribution et la réduction. Elle permet de trouver une solution optimale en utilisant un programme linéaire de taille polynomiale en nombre rationnels. Nous montrons cependant que certains problèmes de communication demeurent NP-complets pour l'optimisation du régime permanent, comme la diffusion partielle (multicast) et le calcul de prefixes parallèles. Ces résultats sont illustrés à l'aide de simulations et d'expérimentations sur la grille Grid5000. Par la suite, nous avons orienté notre étude du régime permanent vers l'ordonnancement d'applications sur la grille. Nous avons proposé des ordonnanceurs centralisés et décentralisés pour des applications consistant en un ou plusieurs ensembles de tâches indépendantes sur une plate-forme maître-esclaves organisée en arbre. Nous avons également proposé une modélisation réaliste de grilles de calcul, que nous avons utilisé pour concevoir des stratégies d'ordonnancement pour des applications divisibles.
Mots-clés :
Algorithmique parallèle, ordonnancement, hétérogénéité, régime permanent, communications collectives, optimisation combinatoire.

Abstract:
The results presented in this document concern scheduling for large-scale heterogeneous platforms. We mainly focus on collective communications taking place during the execution of distributed applications: broadcast, scatter or reduction of data for instance. We study the steady-state operation of these communications and we aim at maximizing the throughput of a series of similar communications. The goal is also to obtain an asymptotically optimal schedule for makespan minimization. We present a general framework to study these communications, which we use to assess that the complexity of the problem is polynomial for most of the collective communications. For a particular model of communication, the bidirectional one-port model, we propose a practical method for solving the problem, which is applied to the broadcast, scatter and reduce operations. This method allows us to build an optimal solution by the resolution of a linear program of size polynomial in rational numbers. However, we prove that the optimization problem remains NP-hard for some communication schemes, like the multicast. These results are illustrated by simulations and real experiments conducted on the Grid5000 platform. Then, we extend our study of the steady-state operation to the scheduling of Grid applications. We first study how to execute multiple applications with centralized and decentralized schedulers on tree-shaped master-slave platforms. We also propose a new realistic model for hierarchical grid platforms, which we use to schedule divisible applications.
Keywords:
Parallel algorithms, scheduling, heterogeneity, steady-state, collective communications, combinatorial optimization.