Conception et mise en oeuvre d'outils efficaces pour le partitionnement et la distribution parallèles de problèmes numériques de très grande taille

Cédric Chevalier

▶ **To cite this version:**

Cédric Chevalier. Conception et mise en oeuvre d'outils efficaces pour le partitionnement et la distribution parallèles de problèmes numériques de très grande taille. Modélisation et simulation. Université Bordeaux I, 2007. Français. NNT : . tel-00410402

HAL Id: tel-00410402
https://theses.hal.science/tel-00410402

Submitted on 20 Aug 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : 3434

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Cédric CHEVALIER**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Conception et mise en œuvre d'outils efficaces pour le partitionnement et la distribution parallèles de problèmes numériques de très grande taille

---

Soutenue le : 28 septembre 2007

Après avis des rapporteurs :

Erik G. Boman	Scientist, SANDIA	
Stéphane Lanteri	Directeur de recherche, INRIA	

Devant la commission d'examen composée de :

Olivier Coulaud	Directeur de recherche, INRIA .	Président & Rapporteur du Jury
François Pellegrini ..	Maître de conférence, ENSEIRB	Directeur de Thèse
Jean Roman	Professeur, ENSEIRB	Directeur de Thèse
Jean-Christophe Weill	Chercheur, CEA	Examinateur

– 2007 –

Remerciements

Tout d'abord, j'aimerais signaler que ces remerciements me tiennent vraiment à cœur, je regrette simplement qu'il me soit impossible d'être exhaustif sur le papier.

Comme il s'agit quand-même de mon manuscrit de thèse, je voudrais commencer par remercier ceux qui ont accepté de le rapporter, Erik Boman et Stéphane Lanteri. Je les remercie aussi de m'avoir fait part de remarques judicieuses concernant la pré-version de ce document, et également d'avoir été présents à ma soutenance, en effectuant un voyage de près de 9000 kilomètres pour Erik. Je remercie également Jean-Christophe Weill et Olivier Coulaud d'avoir accepté d'être présents dans mon jury de thèse.

Ensuite, et je crois que je suis son premier thésard à pouvoir le dire :), je remercie Jean Roman pour avoir laissé son fameux stylo rouge au repos pour ma thèse. Ce qui ne l'a pas empêché, rassurez-vous, de m'indiquer des corrections. Je le remercie également pour plein d'autres choses, notamment ses conseils, qui sont d'ailleurs à l'origine de mon choix d'effectuer cette thèse.

Vient maintenant le tour de remercier le dernier membre du jury, François Pellegrini, qui m'a encadré tout au long de cette thèse et même avant puisque cela avait commencé pour le DEA et aussi en cours à l'ENSEIRB. Il m'a réellement fait entrer dans le monde de la recherche, que ce soit avec l'aide de stylos (parfois rouge, et oui, il en faut quand même un) ou lors des voyages et des conférences. Sa disponibilité à des horaires tardifs a aussi permis de nombreuses fois que la recherche avance de manière (terriblement) efficace aux bons moments :).

Pour continuer dans l'environnement de travail, je tiens à remercier les thésards qui m'ont chaleureusement accueilli au sein de l'équipe ScAlApplix lors de ma première année au Haut-Carré : Gaple, Pierre, Guillaume et mon homonyme Cédric. Le déménagement au LaBRI a permis d'intégrer d'autres personnes de qualité dans ce groupe « super sérieux », je pense notamment à Nico, Benji, Guilhem mais aussi à Martin. Je pense également aux petits jeunes qui m'ont aidé à garder le moral lors de la rédaction : Jérémie, Mathieu (×2), Adam, Robin et les divers stagiaires qui ont souffert pour la science ;) (d'ailleurs on me fait signe qu'il reste des matches de catch à faire). Je n'oublie pas non plus les conseils avisés d'Orel, ni les discussions avec Abdou (sur la moto et `O_DIRECT` notamment !).

Je tiens également à remercier très vivement le CNRS pour son financement (avec la région Aquitaine) et surtout, le plus important, son restaurant de Talence (qui hélas a tiré sa révérence en même temps que moi) pour les succulents repas dont j'ai pu profiter ces deux dernières années, et qui sont étonnamment à l'origine de pas mal d'idées.

Merci aussi à mes amis qui m'ont tous permis de passer de bons moments, souvent malgré la distance. C'est aussi ce qui me fait penser que mon départ pour le Nouveau-Mexique va se dérouler sans accrocs (comme dirait un gouverneur californien) ; par contre, c'est pas une raison pour ne pas venir me voir là-bas ! Je garderai de très bons souvenirs de ces dernières années de vie étudiante mais je rejoins confiant mes anciens compagnons ENSEIRBiens dans le monde « actif », en délaissant à regret mes voisin(e)s qui continuent leurs études.

Je ne peux pas non plus parler des amis sans citer mes copains de lycée : Bob(o), Cédric (encore un !) et Vincent. De soirées grand Cinéma en restos en passant par des vacances à Royan, je leur dois de bons instants de détente et de bonnes rigolades, et ceci depuis maintenant plus d'une dizaine d'années. D'ailleurs, j'espère que nous ne cesserons jamais d'écouter et d'appliquer le conseil plein de sagesse du fabuleux Joe de Killer Crocodile : « On ne refuse jamais une bière. ».

Maintenant vient le tour de ceux qui me supportent depuis le début (et même avant ?), ma famille. Merci à mes parents de m'avoir toujours laissé libre dans mes choix tout en me soutenant

et me conseillant constamment. Merci à ma grand-mère pour les week-ends (nourrissants !) de ressource dans la campagne limousine. Une pensée aussi à mes autres grands-parents qui n'auront pas pu assister à la fin de mes études. Maintenant, je passe le relais des études à ma sœur Sandrine (bonne thèse !) et mes « petites » cousines Amandine et Laure.

Cette page fait certes un peu liste de personnes, mais n'ayant pas les talents artistiques de Zlad ou des Fatals Picards, j'ai préféré ne pas me laisser tenter par des envolées lyriques que je ne maîtriserais pas. Pour finir, je remercie aussi toutes les personnes qui n'ont pas été citées mais qui auraient mérité de l'être.

Bonne lecture,

Cédric.

Conception et mise en œuvre d'outils efficaces pour le partitionnement et la distribution parallèles de problèmes numériques de très grande taille

Résumé :

Cette thèse porte sur le partitionnement parallèle de graphes et essentiellement sur son application à la renumérotation de matrices creuses.

Nous utilisons pour résoudre ce problème un schéma multi-niveaux dont nous avons parallélisé les phases de contraction et d'expansion.

Nous avons ainsi introduit pour la phase de contraction un nouvel algorithme de gestion des conflits d'appariements distants, tout en améliorant les algorithmes déjà existants en leur associant une phase de sélection des communications les plus utiles.

Concernant la phase de d'expansion, nous avons introduit la notion de graphe bande qui permet de diminuer de manière très conséquente la taille du problème à traiter par les algorithmes de raffinement. Nous avons généralisé l'utilisation de ce graphe bande aux implantations séquentielles et parallèles de notre outil de partitionnement Scotch.

Grâce à la présence du graphe bande, nous avons proposé une utilisation nouvelle des algorithmes génétiques dans le cadre de l'expansion en les utilisant comme heuristiques parallèles de raffinement de la partition.

Mots clés :

Parallélisme, partitionnement, renumérotation, matrice creuse, dissections emboîtées, multi-niveaux, heuristiques, contraction, expansion, optimisation locale, mémoire distribuée

Discipline :

Informatique

LaBRI (UMR CNRS 5800) et Projet INRIA Futurs ScAlApplix[1],
Université Bordeaux 1,
351, cours de la libération
33405 Talence Cedex, FRANCE

[1] ScAlApplix : Schémas et Algorithmes Hautes Performances pour les Applications Scientifiques Complexes, `http://www.labri.fr/scalapplix` .

Design and implementation of efficient tools for parallel partitioning and distribution of very large numerical problems

Abstract :

This thesis deals with parallel graph partitioning and, more specifically, focuses on its application to sparse matrix ordering.

To solve this problem, we use a multi-level scheme, of which we have parallelized the coarsening and uncoarsening phases.

We have developed, for the coarsening phase, a new synchronization algorithm to handle conflicts in remote matchings. We have also improved over existing algorithms by adding to them a selection step which aims at keeping only the most useful communications.

Regarding the uncoarsening phase, we have introduced the concept of band graph, which allows us to dramatically decrease problem size for refinement algorithms. We have generalized the use of band graphs to the sequential and parallel implementations of our Scotch partitioning tool.

Basing on band graphs, we have proposed a new application of genetic algorithms to the uncoarsening phase, using them as parallel refinement algorithms.

Keywords :

Parallelism, partitioning, ordering, sparse matrix, nested dissection, multi-level, heuristics, coarsening, uncoarsening, local optimization, distributed memory

Discipline :

Computer science

LaBRI (UMR CNRS 5800) et Projet INRIA Futurs ScAlApplix[2],
Université Bordeaux 1,
351, cours de la libération
33405 Talence Cedex, FRANCE

[2]ScAlApplix : Schémas et Algorithmes Hautes Performances pour les Applications Scientifiques Complexes, `http://www.labri.fr/scalapplix` .

Table des matières

Liste des algorithmes

Liste des tableaux

Table des figures

Liste des définitions

Introduction générale

De plus en plus de domaines nécessitent l'utilisation de simulations numériques pour résoudre leurs problèmes. Par exemple, on peut citer le secteur des prévisions météorologiques, celui de l'automobile et plus généralement tous les domaines de l'industrie lourde ainsi que tous les secteurs de la recherche (en physique, biologie mais aussi en économie). Cette liste est loin d'être exhaustive mais elle montre déjà la variété des personnes utilisant le calcul scientifique.

Les nombreux problèmes simulés peuvent très souvent se réduire à un problème d'algèbre linéaire creuse, c'est-à-dire à un système d'équations linéaires dont les coefficients des inconnues sont presque tous nuls. Cependant, les problèmes actuels comptent plusieurs dizaines de millions d'inconnues, ce qui rend la résolution numérique des systèmes associés impossible avec les moyens de calculs actuels si l'on ne tient pas compte du creux (le coût de la résolution serait alors en $O(n^3)$, où n est le nombre d'inconnues ou d'équations). Ces problèmes linéaires peuvent se ramener à des équations matricielles de type $Ax = b$ où A est une matrice creuse. Des méthodes spécifiques de résolution ont été conçues pour résoudre ces problèmes, qui nécessitent généralement une renumérotation de la matrice A.

Une approche très efficace pour résoudre ce problème consiste à utiliser des techniques de partitionnement de graphes, appliquées aux graphes d'adjacence qui modélisent la matrice à renuméroter. Si, à l'heure actuelle, il existe des techniques séquentielles efficaces de partitionnement, le partitionnement parallèle, rendu lui aussi indispensable du fait de l'augmentation de la taille des problèmes (le graphe modélisant le problème n'est plus stockable sur une seule machine), est loin d'être aussi performant. Les différentes tentatives menées à ce jour ne sont pas réellement concluantes, car trop fortement basées sur des adaptations des algorithmes séquentiels existants et qui se parallélisent mal.

L'objectif principal de cette thèse est donc de rechercher des méthodes intrinsèquement parallèles efficaces de partitonnement de grands graphes irréguliers, et de les valider au sein d'outils facilement intégrables au sein des grands codes numériques parallèles traitant de challenges numériques actuels (problèmes issus de collaborations avec le CEA/CESTA et le CEA-Île-de-France comme par exemple l'endommagement laser, *etc.*).

Le but d'obtenir un partitionneur parallèle de graphe fournissant des résultats d'une qualité comparable à celle des meilleurs outils séquentiels disponibles va conduire à effectuer des tâches de plusieurs natures :

- **algorithmique**, car il faut étudier les faiblesses des approches existantes et y remédier en adoptant de nouvelles méthodes ;
- **implémentation haute performance**, puisque le partitionneur doit s'insérer dans une suite logicielle industrielle complexe et optimisée dont il ne faut pas qu'il devienne l'élément limitant au niveau des performances en temps et en mémoire ;
- **parallélisation et distribution**, car nous souhaitons profiter au mieux de toutes les ressources de la machine, tout en maintenant un bon niveau d'efficacité qualitative.

Pour ce faire, nous allons tout d'abord préciser la nature du problème et rappeler les différents

algorithmes et méthodes séquentiels qui sont couramment utilisés pour le résoudre.

Ensuite, nous présenterons les différentes étapes de la parallélisation et notre choix d'exploiter une méthode multi-niveaux. Nous aborderons aussi le processus de validation expérimentale que nous avons mis en place.

Le chapitre 2 présentera la parallélisation de l'étape de contraction, que nous avons effectuée d'une part en améliorant un algorithme existant et d'autre part en introduisant un nouvel algorithme parallèle de résolution des conflits. Des résultats expérimentaux relatifs à la contraction valideront nos choix.

Au cours du chapitre 3, nous aborderons la description de la parallélisation de la phase d'expansion. Pour celle-ci, nous avons tout d'abord défini la notion de graphe bande qui permet de considérablement réduire la taille du problème à optimiser localement au cours du raffinement. Nous présenterons ce que cette technique, lorsqu'elle est couplée avec des heuristiques d'optimisation, a apporté dans le cas séquentiel, avant de proposer un système parallèle de raffinement reposant sur la méthode d'extraction du graphe bande. Nous validerons les travaux effectués sur l'expansion et sur tout le schéma multi-niveaux en confrontant les résultats que nous obtenons sur différents graphes issus du monde académique et industriel à ceux obtenus avec des outils séquentiels quand cela sera possible, ainsi qu'à ceux obtenus par le renumeroteur parallèle le plus utilisé, PARMETIS.

Le chapitre 4 étend l'utilisation du graphe bande pour le raffinement à un couplage avec un algorithme génétique. Une caractéristique des algorithmes génétiques étant leur capacité à résoudre des problèmes d'optimisation et à se paralléliser facilement, cette voie nous a paru naturelle à explorer. Ce chapitre montrera les différents choix que nous avons effectués pour implanter les algorithmes génétiques sur un prototype multi-threadé afin de valider leur utilisation.

Nous conclurons en décrivant les résultats que nous pouvons obtenir et comment étendre les méthodes que nous avons introduites au partitionnement k-aire de graphes.

Cependant, commençons tout d'abord par rappeler les notations qui nous seront nécessaires tout au long de cette thèse, ainsi que par définir précisément la problématique de la renumérotation des matrices creuses et les manières usuelles de résoudre ce problème.

Chapitre 1

Présentation du problème, état de l'art et positionnement de notre approche

Sommaire

Ce chapitre se veut une présentation rapide des connaissances et procédés actuellement employés pour le partitionnement de graphes ainsi que sur l'utilisation de ce dernier pour la renumérotation de matrices creuses.

1.1 Définitions et notations générales

Cette section rappelle brièvement les notions de graphes et de partitionnement de graphes. Commençons tout d'abord par préciser certains termes qui seront employés tout au long de ce document.

1.1.1 Vocabulaire et notations

Nous utiliserons les notations ensemblistes classiques, ainsi que le vocabulaire français qui leur est habituellement associé. Afin d'éviter toute ambiguïté, nous rappelons les principaux termes qui seront utilisés et leur signification.

- L'expression « supérieur à » (respectivement « inférieur à ») signifie en fait « supérieur ou égal à » (respectivement « inférieur ou égal à ») ; dans le cas contraire, on dit « strictement supérieur à ». De même, un nombre sera dit « positif » si et seulement si il est supérieur (donc éventuellement égal) à 0 (équivalent au terme anglais « *non-negative* »).
- $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$ et $\mathbb{C}$ désigneront respectivement l'ensemble des nombres entiers naturels (positifs), entiers relatifs, des nombres réels et des nombres complexes.
- Pour tout nombre réel x, la notation $\lceil x \rceil$ (respectivement $\lfloor x \rfloor$) désignera la partie entière supérieure (respectivement inférieure de x).
- La notation E^*, par exemple $\mathbb{N}^*$, signifiera que l'ensemble E est privé de l'élément 0.
- $|E|$ désignera le cardinal de l'ensemble E, c'est-à-dire son nombre d'éléments s'il s'agit d'un ensemble fini, $+\infty$ sinon.
- Pour un ensemble $X = \{x_i\}_{i \in [\![1;N]\!]}$, et une fonction $f : X \to \mathbb{R}$, la notation $\bar{f}$ désigne la moyenne arithmétique de f sur X :

$$\bar{f} = \frac{1}{N} \sum_{i=1}^{N} f(x_i) = \operatorname*{moy}_{x \in X} f(x) \ . \tag{1.1}$$

Pour évaluer, ou indiquer la complexité des algorithmes présentés nous utiliserons la notation de Landau au voisinage de $+\infty$ avec $\mathcal{O}$, Θ ou Ω.

Définition 1 (Notation de Landau)
Soient f et g deux fonctions de $\mathbb{R}$ dans $\mathbb{R}$. Le fait que f soit dominée (respectivement minorée) par g au voisinage de $+\infty$ est noté $f = \mathcal{O}(g)$ (respectivement $f = \Omega(g)$), tandis que le fait que f soit asymptotiquement équivalente à g en $+\infty$ est noté $f = \Theta(g)$.

$$f = O(g) \quad \Leftrightarrow \quad \exists k \in \mathbb{R}_+^*, \exists X \in \mathbb{R}, \forall x \in \mathbb{R}, (x > X \Rightarrow |f(x)| \leq k|g(x)|) \ . \tag{1.2}$$

$$f = \Omega(g) \quad \Leftrightarrow \quad \exists k \in \mathbb{R}_+^*, \exists X \in \mathbb{R}, \forall x \in \mathbb{R}, (x > X \Rightarrow |f(x)| \geq k|g(x)|) \ . \tag{1.3}$$

$$f = \Theta(g) \quad \Leftrightarrow \quad f = \mathcal{O}(g) \text{ et } f = \Omega(g) \ . \tag{1.4}$$

Nous serons aussi amenés à parler de matrices et de notions associées. Dans la suite de cet ouvrage nous entendrons par matrice, s'il n'y a pas d'autres précisions, une matrice d'éléments de $\mathbb{R}$. Une matrice de taille n correspondra à la matrice carrée de taille $n \times n$.

Concernant le vocabulaire informatique, nous utiliserons les conventions usuelles concernant les tailles des objets, c'est-à-dire, par exemple, 1 ko = 1024 octets. Lorsque nous parlerons de

processeur, nous évoquerons une unité de calcul, qui pourra correspondre à un cœur ou à un processeur logique dans le cas de l'utilisation d'une architecture permettant le multi-threading.

Nous allons maintenant définir les objets sur lesquels nous allons principalement travailler, c'est-à-dire les graphes.

1.1.2 Définitions sur les graphes

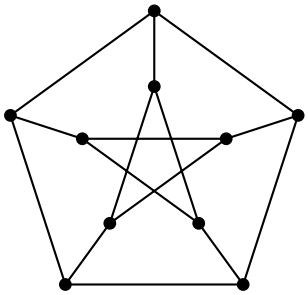
Définition 2 (Graphe non-orienté)
Un graphe non-orienté $G = (V, E)$ est une structure composée d'un ensemble V d'éléments, appelés sommets, et d'une collection E de paires de sommets, appelées arêtes. $V(G)$ et $E(G)$ désigneront respectivement l'ensemble des sommets et la collection des arêtes de G.

Dans la suite, nous noterons $n = |V|$ le nombre de sommets et $m = |E|$ le nombre d'arêtes de G.

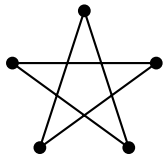
Une arête $\{u, v\}$ est dite incidente à u et à v. u et v sont les extrémités de $\{u, v\}$ et sont dits adjacents. Une arête de la forme $\{u, u\}$ est appelée boucle. Une arête existant en plusieurs exemplaires dans la collection des arêtes est une arête multiple.

Définition 3 (Graphe simple)
Un graphe simple est un graphe sans boucles ni arêtes multiples.

Dans la suite, nous ne considérerons, sauf mention contraire, que des graphes simples non vides, c'est-à-dire comportant au moins un sommet, et nous parlerons donc d'ensemble d'arêtes pour $E(G)$.



(a) Un exemple de graphe simple : le graphe de Petersen.



(b) Un sous-graphe du graphe de Petersen.

FIG. 1.1: Exemples de graphes.

Définition 4 (Degré d'un sommet)
Soit u un sommet du graphe G ; le degré de u, noté $\delta(u)$, est le nombre d'arêtes de $E(G)$ incidentes à u.

Le degré minimal (respectivement maximal) de G, noté $\delta(G)$ (respectivement $\Delta(G)$) est le minimum (respectivement maximum) des degrés de tous les sommets de G. Le degré moyen de G, noté $\bar{\delta}(G)$, est la moyenne arithmétique des degrés de tous les sommets appartenant à $V(G)$.

$$\delta(G) \stackrel{\text{def}}{=} \min_{u \in V(G)} \delta(u) \ . \tag{1.5}$$

$$\Delta(G) \stackrel{\text{def}}{=} \max_{u \in V(G)} \delta(u) \ . \tag{1.6}$$

$$\bar{\delta}(G) \stackrel{\text{def}}{=} \operatorname*{moy}_{u \in V(G)} \delta(u) \ . \tag{1.7}$$

Définition 5 (Chemin)
Un chemin entre deux sommets u et v est une suite $\{\{w_1, w_2\}, \{w_2, w_3\}, \ldots, \{w_{k-1}, w_k\}\}$ d'arêtes de $E(G)$ telle que $u = w_1$ et $v = w_k$. Le nombre d'arêtes de la suite est appelé longueur du chemin.

Nous noterons $\mathcal{C}_G(u, v)$ l'ensemble des chemins de G entre u et v.

Définition 6 (Connexité)
Un graphe G est dit connexe lorsqu'il existe un chemin entre tout couple de sommets.

Un ensemble C de sommets tel qu'il soit maximal au sens de l'inclusion et tel que chaque couple de sommets soit relié par au moins un chemin est appelé composante connexe du graphe G.

Définition 7 (Arbre)
Un arbre est un graphe T connexe et muni de $|V| - 1$ arêtes.

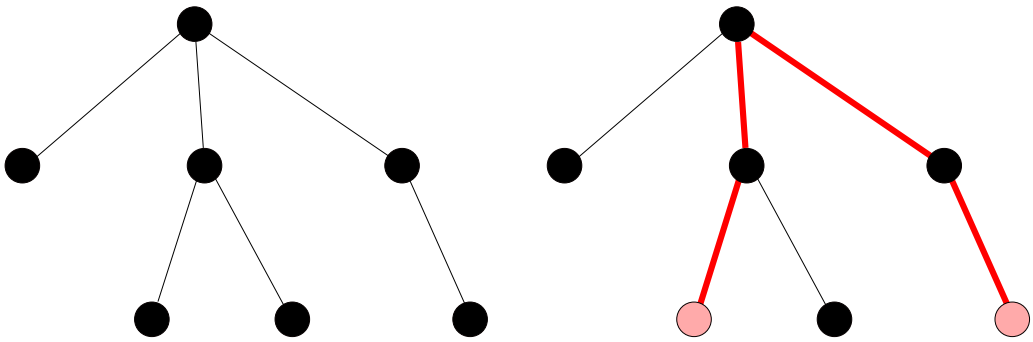
Un arbre à n sommets est donc le plus petit, au sens du nombre d'arêtes, graphe connexe de n sommets. La figure 1.2a représente un arbre.

Définition 8 (Distance dans un graphe)
La distance entre deux sommets u et v est la longueur du plus court chemin entre u et v, s'il en existe un, ou $+\infty$ sinon. Elle est notée $d(u, v)$.

Définition 9 (Diamètre d'un graphe)
Le diamètre d'un graphe G, noté $d(G)$ est égal au maximum de la distance entre deux sommets de G.

$$d(G) \stackrel{\text{def}}{=} \max_{(u,v)\in V^2} d(u, v) \ . \tag{1.8}$$

L'arbre présenté précédemment a donc un diamètre égal à 4, comme illustré en figure 1.2b.



(a) Un exemple d'arbre.

(b) Un chemin de longueur maximale (4) dans cet arbre. Ses extrémités sont en clair et les arêtes empruntées sont en gras.

FIG. 1.2: Un exemple d'arbre de diamètre 4.

Définition 10 (Sous-graphe)
Un sous-graphe $H(U, F)$ de $G(V, E)$ est un graphe tel que :
- *U est un sous-ensemble V ($U \subseteq V$) ;*

– *F est la restriction de E aux couples d'éléments de $U \times U$* $(F = E \cap (U \times U))$.

On dit que H est le sous-graphe de G induit par U et on le note $G|U$. Un exemple de sous-graphe est donné en figure 1.1b de la page 5.

Définition 11 (Graphe pondéré)
Un graphe $G(V, E)$ est dit pondéré lorsqu'il satisfait l'une au moins des deux conditions suivantes :

- *il existe une fonction $w_V : V \to \mathbb{R}$ associant à chaque sommet u son poids $w_V(u)$; on dit dans ce cas que les sommets de G sont valués ;*
- *il existe une fonction $w_E : E \to \mathbb{R}$ associant à chaque arête $\{u, v\}$ son poids $w_E(\{u, v\})$; on dit alors que les arêtes de G sont valuées.*

Dans la suite, nous ne considérerons que des graphes pondérés, en utilisant les fonctions constantes égales à 1 lorsque les graphes ne sont pas à sommets ou arêtes valués.

Définition 12 (Matrice d'adjacence)
La matrice d'adjacence d'un graphe G à n sommets est la matrice $A = (a_{ij})_{(i,j)\in[\![1;n]\!]^2}$ définie par :

$$\forall (i, j) \in [\![1; n]\!]^2, a_{ij} = \begin{cases} 0 & si\ \{i, j\} \notin E(G)\ ; \\ 1 & si\ \{i, j\} \in E(G)\ . \end{cases} \tag{1.9}$$

Nous aurons aussi besoin d'évoquer la coloration des graphes.

Définition 13 (Coloration d'un graphe)
La k-coloration d'un graphe $G(V, E)$ est la définition d'une fonction $f : V \to [\![1; k]\!]$ telle que $\forall (u, v) \in V^2, f(u) = f(v) \Rightarrow \{u, v\} \notin E$.

La valeur $f(u)$ est appelée couleur du sommet u ; un graphe admettant une k-coloration est dit k-coloriable. On peut remarquer qu'un graphe de taille n est forcément n-coloriable.

La notion de graphe quotient nous sera aussi utile dans la suite de cet ouvrage.

Définition 14 (Graphe quotient)
Soient $G(V, E)$ un graphe et $\mathcal{R}$ une relation d'équivalence pour les sommets de $V(G)$. On appelle graphe quotient le graphe $G_{|\mathcal{R}}$ défini de la façon suivante :

1. *les sommets de $G_{|\mathcal{R}}$ sont les classes d'équivalence sur $V(G)$. Si s est la surjection canonique de $V(G)$ dans $V(G)/\mathcal{R} = V(G_{|\mathcal{R}})$, alors on a :*

$$v' \in V(G_{|\mathcal{R}}) \iff \exists v \in V, s(v) = v'\ ; \tag{1.10}$$

2. *son ensemble d'arêtes est décrit par la relation :*

$$\forall \{u, v\} \in E(G), \{s(u), s(v)\} \in E(G_{|\mathcal{R}}) \Leftrightarrow \neg u\mathcal{R}v\ . \tag{1.11}$$

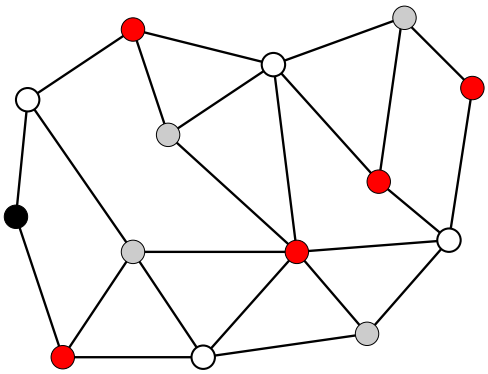
Un exemple de graphe quotient est visible en figure 1.3b. Pour ce dernier, la relation d'équivalence est « possède la même couleur », dans le cadre du coloriage présenté en figure 1.3a.

Comme nous évoquerons aussi quelquefois l'extension de notre problème de partitionnement de graphes aux hypergraphes, il est utile de préciser leur nature.

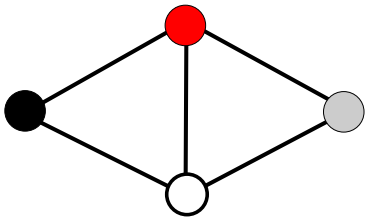
Définition 15 (Hypergraphe)
Un hypergraphe $H = (V, \mathcal{E})$ est une structure composée d'un ensemble V d'éléments appelés sommets, et d'un ensemble $\mathcal{E}$ de sous-ensembles de V appelés hyper-arêtes.

$V(G)$ et $\mathcal{E}(G)$ désigneront respectivement l'ensemble des sommets et l'ensemble des hyper-arêtes de G.

Un graphe est donc un hypergraphe dont chaque hyper-arête ne comporte que deux sommets.

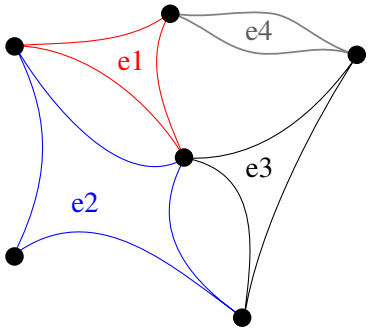


(a) Exemple d'une 4-coloration d'un graphe.

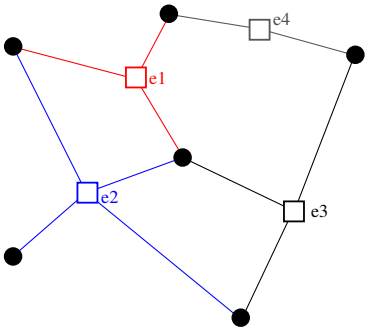


(b) Graphe quotient du graphe précédent, muni de la relation d'équivalence « possède la même couleur ».

FIG. 1.3: Exemple de coloriage d'un graphe et du graphe quotient associé à cette coloration.



(a) Un exemple d'hypergraphe.



(b) Une autre représentation du même hypergraphe.

FIG. 1.4: Deux représentations d'un hypergraphe.

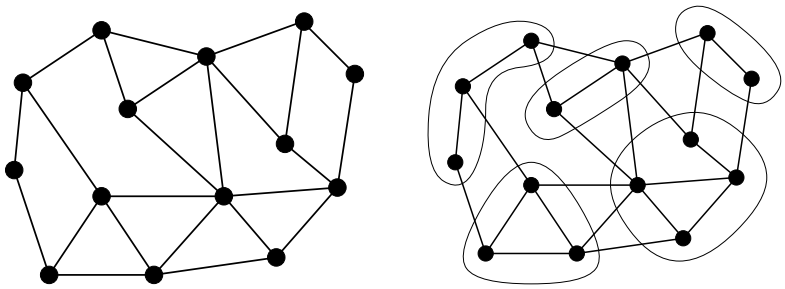
1.1.3 Définitions du partitionnement de graphes

Définition 16 (Partitions de graphes)
Une partition Π *de* V *est une famille* $(V_i)_{i\in[\![1;k]\!]}$ *de* k *sous-ensembles non vides disjoints de* V *telle que l'union de toutes ces parties soit* V*.*

Nous pouvons noter qu'une k-coloration du graphe G peut être vue comme une k-partition du graphe G avec des conditions particulières sur les relations entre les parties.

Nous noterons, pour tout sommet v de G, $\pi(v)$ la partie de Π contenant v et $\mathcal{P}(G)$ l'ensemble de toutes les partitions de $V(G)$. Un exemple de partition d'un graphe G est fourni en figure 1.5 de la page suivante. On peut remarquer que le nombre $|\mathcal{P}(G)|$ de k-partitions du graphe G est asymptotiquement minoré par $\Omega(k^n)$. Trouver la meilleure partition satisfaisant un certain critère en parcourant l'ensemble des partitions de G est donc matériellement impossible pour la plupart des graphes.

Le poids d'une partie π d'une partition Π du graphe $G(V, E)$ est égal à la somme des poids des sommets appartenant à cette partie π. On le note $w_V(\pi)$.



(a) Un graphe G. (b) Une partition Π de G.

FIG. 1.5: Partition d'un graphe.

Soit $\Pi \in \mathcal{P}(G) = (V_i)_{i\in[\![1;k]\!]}$ une partition de G. Parmi les arêtes de G, on distingue, relativement à une partie V_i, trois ensembles distincts :

1. $E_I(V_i)$, l'ensemble des arêtes internes associées à V_i, c'est-à-dire l'ensemble des arêtes ayant leurs deux extrémités dans V_i (*i.e.*l'ensemble des arêtes du sous-graphe $G|V_i$) ;
2. $E_F(V_i)$, l'ensemble des arêtes frontières associées à V_i, c'est-à-dire l'ensemble des arêtes ayant exactement une extrémité dans V_i ;
3. $E_E(V_i)$ l'ensemble des arêtes externes associées à V_i, c'est-à-dire l'ensemble des arêtes n'ayant aucune extrémité dans V_i.

Pour toute partie V_i, la famille $(E_I(V_i), E_F(V_i), E_E(V_i))$ est une partition de l'ensemble des arêtes $E(G)$. Un exemple illustratif de ces ensembles est visible à la figure 1.6a de la page suivante.

Par extension, on définit :

$$E_I(\Pi) \stackrel{\text{def}}{=} \bigcup_{\pi\in\Pi} E_I(\pi)\,; \tag{1.12}$$

$$E_E(\Pi) \stackrel{\text{def}}{=} E(G) - E_I(\Pi) = \bigcup_{\pi\in\Pi} E_F(\pi)\,; \tag{1.13}$$

les ensembles qui représentent respectivement les arêtes internes et externes à l'ensemble des parties de la partition. L'ensemble des arêtes externes $E_E(\Pi)$ est très souvent noté $S(\Pi)$ et est appelé séparateur du graphe G pour la partition Π. Ces deux ensembles sont illustrés en figure 1.6b de la page suivante.

Dans la pratique, un tel partitionnement est appelé partitionnement arête car les interfaces entre les différents ensembles de sommets correspondent à des ensembles d'arêtes.

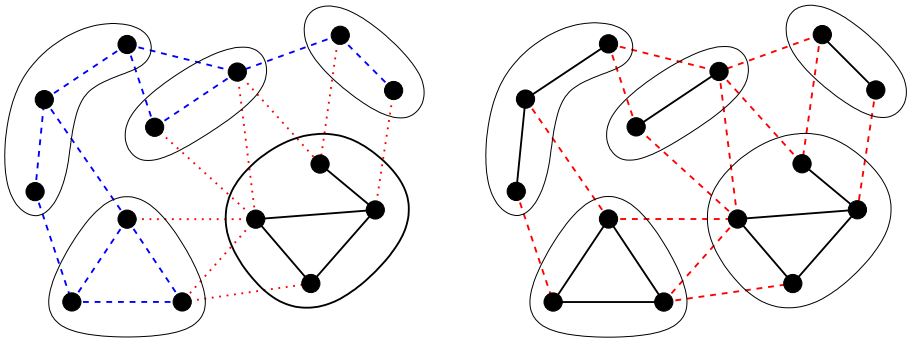
Il existe aussi un k-partitionnement sommet du graphe G qui consiste à partitionner $V(G)$ en une famille $(V_i)_{i\in[\![1;k]\!]}$ d'ensembles de sommets d'une part, et un ensemble S de sommets d'autre part, tel qu'il n'existe pas d'arête $\{u, v\}$ reliant un sommet u de V_i à un sommet v de V_j lorsque $i \neq j$. Pour toute partie V_i, toute arête $\{u, v\}$ de $E_F(V_i)$ a donc une extrémité dans S. La partie S est appelée séparateur du graphe G pour le partitionnement Π.

Le problème de k-partitionnement du graphe $G(V, E)$ correspond généralement à trouver la partition $\Pi \in P_V(G)$ telle que :

– le poids de chaque partie est identique, dans la mesure du possible ;
– le poids de l'interface entre les parties est le plus petit possible.

La première contrainte est une contrainte d'équilibrage, tandis que la seconde est une contrainte sur la taille des interfaces.

On obtient donc la définition suivante.



(a) L'ensemble des arêtes internes E_I (en traits pleins), frontières E_F (en pointillés) et externes E_E (en tirets) de la partie π de G.

(b) Extension à la partition : le séparateur S apparaît en tirets, les arêtes internes en traits pleins.

FIG. 1.6: Les trois différents types d'arêtes induits par une partition.

Définition 17 (Problème du *k*-partitionnement de graphes)
Le problème de k-partitionnement d'un graphe non orienté $G = (V, E)$ à arêtes et sommets valués dans $\mathbb{R}$ où $w_V : V \to \mathbb{R}$ et $w_E : E \to \mathbb{R}$ désignent les applications qui à chaque sommet ou arête associent son poids, s'énonce maintenant de la manière suivante :

Trouver un couple $(S, (V_i)_{1\leq i\leq k})$ où S est un séparateur et $(V_i)_{1\leq i\leq k}$ une famille de sous-ensembles de V tels que :

1. *le poids de chaque partie V_i soit le même, c'est-à-dire qu'il faut minimiser la différence des poids entre chaque couple de parties :*

$$\forall (i,j) \in [\![1;k]\!]^2, |w_v(\pi_i) - w_v(\pi_j)| \text{ minimal};$$

2. *la coupe $\sum_{x\in S} w(x)$ soit minimale (avec $w = w_V$ dans le cas d'un séparateur sommet et $w = w_E$ dans le cas d'un séparateur arête).*

Un 2-partitionnement est appelé bipartitionnement.

1.2 Algorithmes utilisés pour le partitionnement de graphes

Notre but est de résoudre le problème de partitionnement présenté dans la définition 17. Ce problème est un problème de recherche d'un optimum parmi l'ensemble des partitionnements du graphe G, qui a été démontré comme étant NP-Complet [25, 62]. Trouver la meilleure solution dans le cas d'un graphe G quelconque a donc une complexité proportionnelle au nombre de partitions de G, c'est-à-dire en $\Omega(k^n)$. De plus, on remarquera que vérifier que l'on dispose du meilleur partitionnement est généralement tout aussi difficile. Dans la suite, nous serons donc la plupart du temps obligés de nous contenter de résultats non-optimaux, produits par des heuristiques.

Les approches actuelles reposent ainsi sur des heuristiques fournissant un résultat approximatif de plus ou moins bonne qualité relative, la qualité réelle n'étant généralement pas mesurable puisqu'on ne connaît pas l'optimum. Il existe plusieurs familles d'heuristiques utilisées. Une présentation plus complète et étendue aux hypergraphes est disponible dans la thèse d'Aleksandar Trifunović [79].

Nous allons voir dans la suite de ce chapitre les principales classes d'heuristiques utilisées le plus couramment. Les algorithmes que nous présenterons sont des algorithmes de partitionnement arête, mais des adaptations au partitionnement sommet existent également.

1.2.1 L'approche spectrale

Définition 18 (Matrice des degrés)
La matrice D des degrés associée au graphe G est la matrice de taille $|V| = n$ suivante :

$$\forall (i,j) \in [\![1;n]\!]^2, a_{ij} = \begin{cases} \delta(v_i) & si\ i = j \\ 0 & sinon \end{cases} \tag{1.14}$$

La matrice D est donc la matrice diagonale ayant pour termes diagonaux les degrés des sommets.

Définition 19 (Matrice de Laplace)
La matrice de Laplace Q du graphe G est définie par $Q = D - A$ où A est la matrice d'adjacence de G et D la matrice des degrés.

L'approche spectrale du partitionnement de graphe consiste à rechercher les valeurs propres de la matrice de Laplace Q associée au graphe G. La matrice Q est semi-définie positive, donc les valeurs propres de Q peuvent être ordonnées de la façon suivante : $\lambda_1 = 0 \leq \lambda_2 \leq \ldots \leq \lambda_n$. Il est démontré que la multiplicité de la première valeur propre, égale à 0, correspond au nombre de composantes connexes du graphe G. Si G est connexe, la deuxième plus petite valeur propre λ_2 est strictement positive. Le vecteur propre X_2 associé à la valeur propre λ_2, souvent appelé *vecteur de Fiedler*, a été intensivement étudié par Fiedler [21, 22] et possède des propriétés intéressantes concernant le placement des sommets de G sur un segment [35].

Nous pouvons ordonner les sommets de G sur la droite des réels en associant à chaque sommet v_i une position x_i correspondant à sa composante (la $i^{\text{ème}}$) du vecteur de Fiedler X_2. Le résultat de Hall [35], qui montre que la solution optimale du placement des sommets sur une droite est donné par cette deuxième plus petite valeur propre λ_2, implique que si deux sommets v_i et v_j sont connectés par une arête de $E(G)$, la distance $|x_i - x_j|$ est petite. Les sommets fortement connectés sont donc proches dans l'ordonnancement des sommets. On peut donc déduire une partition $\Pi = (P_0, P_1)$ du graphe G en choisissant un réel ρ et en posant $P_0 = \{v_i | x_i \leq \rho\}$ et $P_1 = \{v_i | x_i > \rho\}$.

Par exemple, dans [69], Pothen *et al.* utilisent la valeur médiane x_m comme valeur de ρ pour effectuer le bipartitionnement récursif d'un graphe.

Il a été montré que cette méthode permettait d'obtenir un extremum global avec certains graphes ; cependant, il a aussi été mis en évidence que cette méthode est très coûteuse en terme de calculs [5].

1.2.2 L'approche combinatoire

Dans cette approche, nous travaillons directement sur la structure du graphe. L'idée générale consiste à effectuer un parcours de proche en proche d'une partie de l'ensemble $\mathcal{P}(G)$ afin d'y trouver le meilleur candidat qui résolve notre problème.

Un algorithme de ce type nécessite, au plus, les deux données suivantes :

1. la définition d'un voisinage dans $\mathcal{P}(G)$;
2. l'historique des optimisations précédentes.

Le voisinage permet de définir comment l'algorithme va progresser en *perturbant* la solution courante S : par exemple, on peut définir le voisinage de S comme correspondant à un seul déplacement de sommet par l'ensemble des éléments de $\mathcal{P}(G)$ qui peuvent être obtenus en changeant de partie un seul sommet de S.

1.2.2.1 Les algorithmes itératifs d'optimisation

Les algorithmes itératifs d'optimisation fonctionnent en partant d'une partition $\Pi_0 \in \mathcal{P}(G)$ de G, valide et bien équilibrée, et se déplacent dans l'espace des solutions en sélectionnant le voisin le plus à même de réduire la coupe de la partition. L'algorithme s'arrête lorsqu'aucun des voisins n'est statisfaisant. Ces algorithmes convergent donc vers l'optimum local accessible depuis la partition initiale Π_0.

Bien que la convergence ne soit que locale et dépendante du point de départ dans $\mathcal{P}(G)$, ces algorithmes sont très populaires, les résultats produits pouvant être améliorés en effectuant plusieurs exécutions à partir de partitions initiales différentes ou en utilisant le schéma multi-niveaux qui sera présenté plus loin. Parmi les algorithmes de cette catégorie, nous pouvons citer l'algorithme de Kernighan-Lin (KL), ainsi que l'algorithme de Fiduccia et Mattheyses (FM).

L'algorithme de Kernighan-Lin Cet algorithme a été présenté par Kernighan et Lin [47]. Il utilise des échanges de paires de sommets entre les différentes parties, c'est-à-dire que ce voisinage de $\mathcal{P}(G)$ implique que n'importe quelle paire de sommets $(u, v) \in V^2$, telle que $\Pi(u) \neq \Pi(v)$, puisse être échangée.

L'algorithme de Kernighan-Lin, que nous noterons dorénavant KL, fonctionne par *passes*, c'est-à-dire qu'il effectue plusieurs itérations, comme on peut le voir à la ligne 2 de l'algorithme 1 de la page ci-contre. Le *gain* associé à un échange est la différence entre la valeur de la coupe avant l'échange et celle obtenue après l'échange. Il représente la variation de la qualité du partitionnement due à l'échange de sommets associé, une valeur positive du gain correspondant à une amélioration du partitionnement.

L'aspect itératif de la méthode est dû à la boucle externe (ligne 2). Le cœur de l'algorithme est d'effectuer le meilleur échange possible parmi ceux qui sont disponibles, puis de marquer les deux sommets déplacés comme dorénavant non échangeables. On calcule la coupe de notre solution et on note le mouvement qui vient d'être effectué. Lorsqu'il n'y a plus de mouvements possibles, on recherche dans l'historique des valeurs de coupe la meilleure valeur, c'est-à-dire la valeur maximale de la somme des gains, et on revient dans cette configuration. On itère ce processus, en débloquant tous les sommets, jusqu'à ce qu'on ne puisse plus améliorer la qualité.

On remarque que l'on effectue tous les mouvements possibles, dans l'ordre donné par les gains ; cela peut permettre de sortir des extrema locaux, car même les mouvements dégradant temporairement la qualité sont pris en compte si ce sont les seuls disponibles (à la ligne 6 de l'algorithme, le gain peut être négatif). Néanmoins, le choix du mouvement à effectuer lorsque plusieurs mouvements de même gain sont disponibles peut avoir d'importantes conséquences sur la solution finale obtenue. C'est pourquoi on effectue en général plusieurs exécutions de l'algorithme de Kernighan-Lin, en conservant seulement la meilleure solution.

En utilisant une liste triée selon les gains, l'étape de sélection du meilleur échange possible peut s'effectuer en $\Theta(n \log n)$; comme la boucle est parcourue au plus n fois, la complexité en temps de la boucle interne peut être ramenée à $\Theta(n^2 \log n)$. Le nombre de passes, c'est-à-dire le nombre d'itérations de la boucle externe est quant à lui borné par le nombre m d'arêtes dans le cas d'un graphe non pondéré.

Algorithme 1 Algorithme de Kernighan-Lin

```
1: Procédure KL(G : graphe, Π : partition de G)
   i : rang de l'itération interne
   S_i : somme des gains jusqu'au rang i
2:     Répéter                                   ▷ On itère sur le graphe G
3:         i ← 0                                 ▷ Initialisation des variables pour une itération
4:         S_i ← 0
   Boucle interne de l'algorithme KL
5:         Tant que (il existe un échange de deux sommets non bloqués) Faire
6:             Faire le meilleur échange possible
7:             Bloquer les deux sommets déplacés
8:             Enregistrer le gain g_i de l'échange
9:             S_{i+1} ← S_i + g_i
10:            i ← i + 1
11:        Fin de Tant que
12:        Trouver x telle que la somme partielle S_x des gains soit maximale
13:        Si S_x < 0 alors
14:            Annuler tous les échanges effectués
15:        Sinon
16:            Annuler les mouvements de x à i
17:        Fin de Si
18:    Jusqu'à ce que S_x < 0
19: Fin de Procédure
```

L'algorithme de Fiduccia-Mattheyses L'algorithme de Fiduccia-Mattheyses (FM) [20] correspond à une amélioration en temps quasi-linéaire de l'algorithme de Kernighan-Lin présenté précédemment.

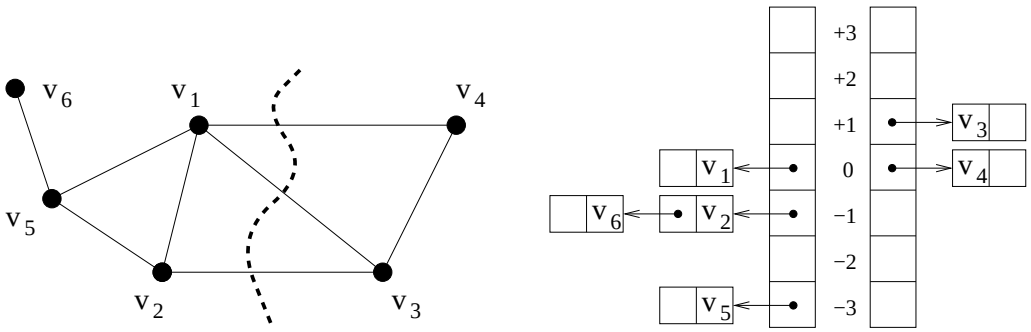
Contrairement à l'algorithme KL, FM réalise des mouvements de sommets d'une partie vers une autre et non des échanges. Il peut donc déséquilibrer la partition, et c'est pour éviter cela que seuls les mouvements qui permettent de rester dans une tolérance prédéfinie pour l'équilibre sont réalisables.

L'algorithme maintient pour chaque sommet une valeur de gain, qui représente la variation de la valeur de coupe si le sommet est migré vers l'autre partie. Les sommets sont classés selon leur gain et un tableau de listes chaînées de sommets, indexé par les gains, est utilisé, chaque liste contenant les sommets de gain correspondant à l'indice. Ce tableau est borné par une valeur maximale et une valeur minimale du gain (souvent l'opposée de la maximale). En pratique, un tableau de gain est utilisé pour chaque partie comme cela est illustré par la figure 1.7 de la page suivante.

L'algorithme est présenté dans l'algorithme 2 de la page 15. Il consiste à sélectionner un sommet v associé au meilleur gain possible, puis à effectuer le mouvement, à marquer v comme étant déplacé et à mettre à jour les gains de ses voisins qui n'ont pas déjà été déplacés. On réordonne ces voisins en mettant à jour leur position dans la table des gains et on réitère le procédé. On constate que la principale différence avec l'algorithme KL tient à la capacité de FM de trouver le sommet de plus grand gain en temps quasi-constant. Une étude plus complète de cette caractéristique peut être trouvée dans la thèse de François Pellegrini [65, page 91]. Le temps quasi-constant est obtenu grâce à la structure d'ordonnancement des gains, qui associe à chaque valeur autorisée pour les gains la liste des sommets correspondants à ce gain. Le temps

nécessaire pour sélectionner un sommet de meilleur gain est donc proportionnel au nombre de valeurs autorisées, qui est une constante lors du déroulement de l'algorithme. La mise à jour de cette structure étant effectuée uniquement pour les sommets non marqués, elle devient de moins en moins coûteuse lorsque le nombre de boucles effectuées augmente. La complexité en espace de l'algorithme est en $\Theta(n+m)$ et celle en temps est aussi en $\Theta(n+m)$ [65, pages 92–93].

Comme pour l'algorithme de Kernighan-Lin, on notera qu'en général plusieurs sommets correspondent au gain maximal et que le choix d'un sommet peut grandement influencer la solution obtenue au final par l'algorithme [33, 50]. Il est difficile de sélectionner le *bon* sommet et c'est pour cela que, comme pour l'algorithme de Kernighan-Lin, de nombreuses implantations de l'algorithme de Fiduccia-Mattheyses effectuent plusieurs exécutions et gardent seulement le meilleur résultat obtenu.



(a) Bipartition courante d'un graphe non pondéré.

(b) Listes et tableaux de gains correspondant à cette bipartition.

FIG. 1.7: Structures de données utilisées par l'heuristique de Fiduccia-Mattheyses.

Le principal défaut de ces approches itératives concerne leur vision strictement locale du problème, ce qui implique que le résultat ne peut être au mieux qu'un optimum local, fortement dépendant de la solution initiale. D'autres approches d'algorithmes d'optimisation ayant une vision plus globale peuvent être utilisées.

Les algorithmes à bulles Les algorithmes à bulles, qui font partie de la classe des algorithmes de diffusion, sont généralement employés de manière récursive. Contrairement aux algorithmes d'optimisation locale présentés précédemment, comme Kernighan-Lin ou Fiduccia-Mattheyses, ces algorithmes ne nécessitent pas forcément la connaissance d'une partition de départ, celle-ci pouvant simplement être aléatoire, la seule donnée importante étant l'emplacement du centre de chaque bulle.

Le fonctionnement de la méthode des bulles est présenté dans l'algorithme 3 de la page suivante. L'étape principale se situe à la ligne 4 et consiste à faire grossir les bulles depuis leur centre, c'est-à-dire les points d'où la diffusion est initiée, jusqu'à ce qu'elles se neutralisent en recouvrant tous les sommets du graphe. La nouvelle partition est définie en prenant les bulles comme parties. On réitère le procédé jusqu'à ce qu'une certaine qualité soit obtenue, ou que le résultat obtenu n'évolue plus.

Les résultats obtenus par cette approche sont encourageants en terme de qualité [58], mais l'algorithme reste lent, essentiellement à cause de la difficulté de trouver le centre des parties.

On notera aussi que les algorithmes de bulles n'optimisent pas la coupe directement, et sont donc réservés la plupart du temps aux problèmes dépendant fortement de la forme des parties,

Algorithme 2 Algorithme de Fiduccia-Mattheyses

Procédure FM(G : graphe,Π : bipartition de G)
  Répéter ▷ On itère sur le graphe G
    Calculer les gains des sommets
    Ordonner les gains des sommets
  Boucle interne de l'algorithme FM
    Tant que Il existe un sommet à déplacer **Faire**
      Sélectionner le sommet v correspondant au meilleur mouvement possible
  $\overline{\pi(v)}$ *est le complémentaire de* $\pi(v)$ *dans* Π
      Déplacer v de $\pi(v)$ vers $\overline{\pi(v)}$
      Marquer v
      Pour tous les voisins non marqués u de v dans G **Faire**
        Calculer le gain de u
      Fin de Pour
      Ordonner les gains des sommets non marqués
      Calculer le gain g_i du déplacement
      $S_{i+1} \leftarrow S_i + g_i$
      $i \leftarrow i + 1$
    Fin de Tant que
    Calculer la meilleure somme partielle S_x des gains
    Si $S_x < 0$ **alors**
      Annuler tous les déplacements effectués
    Sinon
      Annuler les mouvements de x à i
    Fin de Si
  Jusqu'à ce que $S_x < 0$
Fin de Procédure

Algorithme 3 Algorithme à bulles

Fonction Bulles(G : graphe)
  Π : bipartition de G
  $\mathcal{C}$: ensemble des centres des bulles
  $\mathcal{C} \leftarrow k$ sommets choisis aléatoirement
  Répéter
    Faire grossir les bulles depuis leur centre $\mathcal{C}$: obtention de Π
    Calculer les centres $\mathcal{C}$ des parties de Π
  Jusqu'à ce que (la coupe soit inférieure à un certain seuil)
  Retourner Π
Fin de Fonction

comme certaines méthodes itératives de résolution de systèmes linéaires.

Le problème de partitionnement de graphes étant un problème d'optimisation, les résultats de problèmes d'optimisation plus généraux ont aussi été utilisés, afin d'obtenir des solutions représentant un minimum global pour la coupe.

1.2.2.2 Les algorithmes génériques d'optimisation

La plupart des méta-heuristiques d'optimisation ont aussi été utilisées pour résoudre le problème de partitionnement de graphes. Nous nous contenterons d'évoquer les différentes approches.

Les algorithmes évolutionnistes Les algorithmes évolutionnistes, parmi lesquels on trouve les algorithmes génétiques, ont fait l'objet de plusieurs approches [11, 16, 74, CP06a] totalement différentes qui seront décrites dans la section 4.3 de la page 102. Le concept de base de ces algorithmes est de tenter d'imiter la nature et sa faculté d'adaptation à un problème donné grâce à la sélection naturelle décrite par la théorie de l'évolution.

Dans le cas des algorithmes génétiques, le principe est d'explorer l'espace des solutions à l'aide d'une population d'individus qui vont échanger entre eux certaines de leurs caractéristiques lors d'une étape de reproduction, ceci dans le but d'obtenir de nouveaux individus combinant les qualités de leurs parents tout en essayant d'en diminuer les défauts. Tous les individus ne se reproduisent pas et il existe plusieurs stratégies pour sélectionner la partie reproductrice de la population. Les principales difficultés d'application de ces algorithmes tiennent d'une part à la multitude de paramètres qu'il faut régler en fonction de la nature du problème, d'autre part à leurs coûts en mémoire et en temps : la taille de la population est liée à celle du problème et le temps est dépendant du nombre de générations qui est lié à la taille de la population. Différentes variantes des algorithmes génétiques seront elles aussi décrites dans le chapitre 4.

Le recuit simulé Le recuit simulé est une meta-heuristique introduite comme une approche générale aux problèmes d'optimisation [13, 48] et est basé sur la procédure *Metropolis* [57] de la mécanique statistique. Il consiste ici à optimiser une partition initiale en la modifiant légèrement de manière aléatoire [7]. Le schéma de fonctionnement du recuit simulé est présenté dans l'algorithme 4 de la page suivante. À la ligne 5, on choisit un voisin, puis on le sauvegarde s'il s'agit du meilleur élément trouvé jusqu'à présent. Cet individu devient notre nouveau point de départ avec la probabilité $Prob(C, C', \text{temperature}(i, i_{max}))$ (ligne 10). Cette probabilité est donnée par $\exp\left(\frac{-\delta\mathcal{E}}{T}\right)$ où $\delta\mathcal{E}$ est la variation d'énergie du système et T la température. Dans notre cas, la variation d'énergie $\delta\mathcal{E}$ correspond à la différence de qualité $C' - C$ et la température est une fonction $\text{temperature}(i, i_{max})$ des nombres d'itérations effectuées et totales. La probabilité de changer de point de départ pour notre voisin couramment sélectionné (ligne 10) est donc : $Prob(C, C', \text{temperature}(i, i_{max})) = \exp\left(\frac{C-C'}{\text{temperature}(i,i_{max})}\right)$.

Il a été démontré par Hajek [34] que l'algorithme du recuit simulé converge vers l'optimum global si et seulement si le déroulement du recuit permet à la température T de tendre vers 0 suffisamment lentement. Ceci est, en partie, responsable de la lenteur de cette approche qui n'a donc pas été, pour l'instant, utilisée avec succès dans le cas du partitionnement de graphes, contrairement à de nombreux autres domaines. Une autre difficulté importante concerne le choix de la définition du voisinage ainsi que de la fonction de température, ces deux paramètres étant responsables du bon fonctionnement de l'algorithme.

Algorithme 4 Algorithme du recuit simulé

Procédure RECUIT SIMULÉ(G : graphe, Π : bipartition de G)
Π' : partition courante de G
Π_m : meilleure partition
C : énergie (coût) de la partition Π
C' : énergie de la partition Π'
C_m : Coût de la meilleure partition
i : nombre d'itérations
Paramètres i_{max} et C_{max} : nombre maximal et énergie maximale permis
    $i \leftarrow 0$; $C \leftarrow$ Coût(Π) ▷ Initialisation
    $(\Pi_m, C_m) \leftarrow (\Pi, C)$ ▷ Meilleure solution connue
    Tant que $i < i_{max}$ et $C > C_{max}$ **Faire** ▷ Il reste du temps et le but n'est pas atteint
        $\Pi' \leftarrow$ voisin(Π) ▷ On choisit aléatoirement un voisin
        $C' \leftarrow$ Coût(Π')
        Si $C' < C_m$ **alors**
            $(\Pi_m, C_m) \leftarrow (\Pi', C')$ ▷ On stocke une meilleure solution
        Fin de Si
        Si random() $< Prob(C, C', \text{temperature}(i, i_{max}))$ **alors**
            $(\Pi, C) \leftarrow (\Pi', C')$ ▷ Changer aléatoirement l'état
        Fin de Si
        $i \leftarrow i + 1$
    Fin de Tant que
    $\Pi \leftarrow \Pi_m$ ▷ On garde le meilleur résultat obtenu
Fin de Procédure

Autres approches D'autres méta-heuristiques comme la recherche tabou [6], les algorithmes de gradients aléatoires adaptatifs (GRASP) [4], les méthodes de diffusion stochastique [77, 83] ou encore les méthodes de colonies de fourmis [49, 51] ont été appliquées avec plus ou moins de succès au partitionnement de graphe.

La recherche tabou, introduite par Glover [27], consiste à se déplacer dans le voisinage de la solution courante, mais en gardant en mémoire les solutions précédemment sélectionnées, afin d'éviter de parcourir des cycles dans l'exploration.

Les méthodes de colonies de fourmis consistent à imiter le comportement de nombreux insectes, comme les fourmis, les fourmis volantes ou les abeilles, qui exploitent une méthode de résolution collective. En effet, les fourmis utilisent des phéromones pour marquer les différents chemins empruntés et la concentration de ces marqueurs chimiques est d'autant plus importante que la fréquence d'utilisation est importante. Pour le k-partitionnement l'idée est donc d'utiliser k colonies de fourmis dont le but est d'accumuler la nourriture qui est distribuée sur les sommets du graphe. Ces méthodes sont différentes des méthodes de diffusion ou de bulles présentées précédemment car l'exploration ne se fait pas de manière uniforme mais est guidée par la répartition des colonies, de la nourriture et par les décisions des fourmis.

Toutes ces méthodes semblent pouvoir conduire à des partitions de bonne qualité mais leurs utilisations restent marginales à cause de leurs durées et de la difficulté pour étalonner leurs paramètres correctement ; on leur préfère souvent l'utilisation d'optimisations locales itératives dans un contexte multi-niveaux, que nous allons maintenant présenter.

1.3 L'approche multi-niveaux

L'efficacité des méthodes combinatoires est étroitement liée à la taille de l'espace de recherche, c'est-à-dire à la taille de $\mathcal{P}(G)$. De plus, l'optimisation réalisée n'est généralement que locale, sans prise en compte de la topologie globale du graphe G. Au contraire, les méthodes spectrales exploitent une vision globale du graphe G mais deviennent inefficaces lorsque la taille du graphe devient importante.

La technique du multi-niveaux, introduite par [5, 81, 36], permet de réduire la taille du graphe ainsi que celle de l'espace de recherche tout en procurant l'accès à une vision globale pour les algorithmes combinatoires. L'idée directrice est de travailler sur un graphe réduit ayant les mêmes propriétés topologiques que le graphe initial.

Pour ce faire, le schéma multi-niveaux comporte trois étapes :

1. l'étape de contraction, durant laquelle on applique récursivement une fonction de contraction afin de diminuer la taille du graphe ;
2. l'étape de partitionnement initial, où l'on applique une heuristique sur le plus petit graphe ;
3. l'étape d'expansion, durant laquelle la partition initiale est projetée et raffinée sur les graphes de plus en plus gros jusqu'au graphe initial.

Cette procédure est illustrée par la figure 1.8 de la page ci-contre. Détaillons maintenant ces différentes phases.

1.3.1 La phase de contraction

Le but de cette phase est d'obtenir une suite (G_λ) de graphes issus de G tels que leurs topologies soient proches de celle de G et que leurs nombres de sommets soient strictement décroissants.

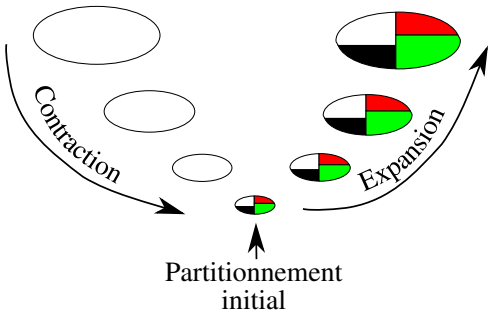


FIG. 1.8: Représentation schématique d'un 4-partitionnement multi-niveaux.

On appelle rapport de réduction ou de contraction r le quotient de $|V_\lambda|$ et de $|V_{\lambda+1}|$.

$$r = \frac{|V_\lambda|}{|V_{\lambda+1}|} \ . \tag{1.15}$$

Généralement, les sommets de G_λ sont appariés, c'est-à-dire regroupés par deux, pour former les sommets de $G_{\lambda+1}$. Le rapport de réduction r vaut donc dans ce cas au maximum deux. Pour apparier les sommets, différentes stratégies sont utilisées :

- un choix aléatoire du voisin avec lequel le sommet va s'apparier ; c'est l'approche qui était utilisée dans les premières implantations du schéma multi-niveaux [9, 36] ;
- la méthode *Heavy Edge Matching* (H.E.M.) [43], qui consiste à choisir le voisin qui est l'extrémité de l'arête dont le poids est le plus élevé.

La technique H.E.M. est privilégiée dans les outils actuels [46, 82, 68] parce qu'elle semble mieux préserver la topologie du graphe. On parle alors de « contraction dimensionnelle ».

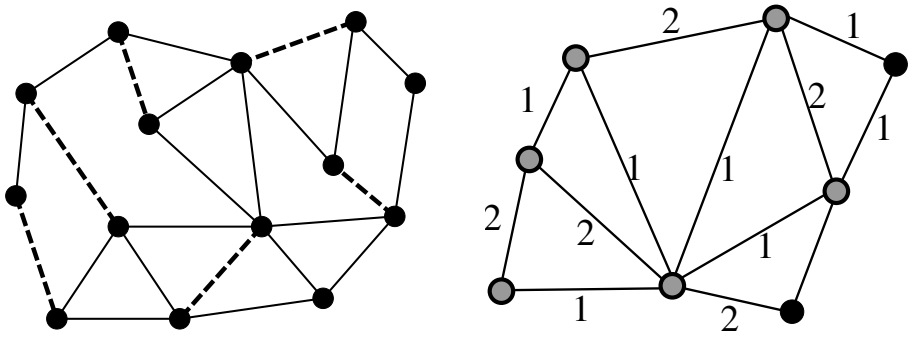
Tout sommet s résultant de la contraction a pour caractéristiques d'être pondéré par la somme des poids des deux sommets u et v dont il est issu et d'être l'extrémité de toutes les arêtes dont les deux sommets u et v étaient extrémités, mise à part l'arête qui les reliait, qui est supprimée. On a donc, en notant $E_{\lambda+1}(x)$ l'ensemble des arêtes ayant pour extrémité x :

$$s \text{ sommet contracté depuis } u \text{ et } v \Rightarrow \begin{cases} w_V(s) = w_V(u) + w_V(v) \, ; \\ E_{\lambda+1}(s) = E_{\lambda+1}(u) \cup E_{\lambda+1}(v) \backslash \{u, v\} \ . \end{cases} \tag{1.16}$$

Les arêtes incidentes à u ou v sont alors modifiées pour les déclarer maintenant incidentes à s et dans le cas où des arêtes multiples apparaissent, seule une représentante est conservée, son poids valant la somme des arêtes concernées. Un exemple de contraction est donné en figure 1.9 de la page suivante.

Il est important de noter que la phase de contraction préserve le poids total des sommets du graphe, ce qui permettra d'avoir un partitionnement qui respectera l'équilibrage des parties, à tous les niveaux.

Nous noterons qu'il existe d'autres techniques de contraction qui ne forment pas nécessairement des appariements mais regroupent les sommets par ensembles. Les taux de contraction ainsi obtenus peuvent être supérieurs à 2, ce qui permet de limiter le nombre d'étapes dans le schéma multi-niveau, mais la conservation de la topologie est en général plus difficile. Pour améliorer la conservation topologique dans ce cas, il existe une technique de contraction *probabiliste* [71], c'est-à-dire qu'un sommet peut appartenir à plusieurs ensembles contractés, avec une certaine probabilité. L'intérêt de cette méthode est de combiner un bon ratio de contraction et



(a) Le graphe original. Les arêtes en tirets sont les arêtes qui vont être contractées.

(b) Le graphe contracté. Les nombres sur les arêtes correspondent au poids de celles-ci.

FIG. 1.9: Exemple de contraction de graphe par appariement d'arêtes.

une bonne proximité topologique entre les niveaux. Cependant, elle n'est que peu utilisée, car elle est difficile à mettre en œuvre.

Plus de détails sur la contraction seront donnés dans le chapitre 2.

1.3.2 Le partitionnement initial

Lorsque le nombre de sommets du graphe contracté G_λ est suffisamment petit, on applique une heuristique de partitionnement sur celui-ci. Dans la version actuelle de SCOTCH, par exemple, ce seuil est fixé à une centaine de sommets. Nous notons G_{λ_m} le graphe le plus contracté dans le schéma multi-niveaux.

Les types d'heuristiques qui sont appliquées à ce graphe G_{λ_m} sont très variables selon les utilitaires : spectral pour [5, 36], itératif glouton pour [44, 65], utilisant des algorithmes génétiques pour [4], *etc.* ...

La taille du graphe G_{λ_m} étant petite, le schéma multi-niveaux tend à ce que la partition trouvée corresponde à un optimum le plus global possible[3] pour notre problème, et c'est cette globalité que la dernière phase, l'expansion, va tenter de conserver.

1.3.3 La phase d'expansion

Le but de cette phase est de projeter sur le graphe G initial l'optimum global calculé sur le plus petit graphe G_{λ_m}.

L'idée consiste à procéder par étapes en projetant la partition du niveau $\lambda + 1$ sur le niveau λ et en raffinant la partition obtenue grâce à une heuristique d'optimisation locale, comme, par exemple, KL ou FM Le fait que les graphes de deux niveaux consécutifs λ et $\lambda+1$ soient proches topologiquement permet d'espérer que l'optimum local vers lequel va converger l'algorithme de raffinement sur G_λ ne soit pas très éloigné de la projection de celui de $G_{\lambda+1}$; donc, par induction, si le partitionnement obtenu au niveau le plus contracté G_{λ_m} correspond à l'optimum global, l'optimum de chaque niveau G_λ correspond aussi à l'optimum global sur G_λ.

On voit que la contrainte de la proximité topologique entre deux niveaux est la clé de la réussite du schéma ; la qualité de la phase d'appariement des sommets est donc critique. Pour assurer une meilleure qualité, Karypis et Kumar [42] ont proposé d'augmenter le nombre de

[3] L'optimum global étant en général inaccessible de manière sûre.

niveaux et donc de réduire le taux de contraction r pour le conserver dans l'intervalle $[1.5, 1.8]$ au lieu de le faire tendre vers 2 pour accélérer les calculs.

Nous reviendrons plus précisément sur la phase de raffinement dans le chapitre 3 de la page 67.

1.4 Mise en œuvre parallèle

La taille des graphes à partitionner étant de plus en plus importante, la mémoire des ordinateurs séquentiels actuels se révèle insuffisante pour exécuter les algorithmes de partitionnement. D'autre part, pour de nombreuses applications, comme la renumérotation de matrices creuses par exemple, le partitionnement de graphes ne correspond qu'à un pré-traitement, le traitement principal étant pour sa part effectué en parallèle. Le temps de pré-traitement devient alors trop important par rapport à celui de la tâche principale.

Pour ces deux raisons, des tentatives de parallélisation des algorithmes de partitionnement de graphes ont vu le jour. On peut citer ParMeTiS [46], ParJostle [82], ou encore notre projet PT-Scotch [68] pour ce qui concerne les partitionneurs de graphes ; Zoltan [8] ou encore Par*k*way [80] pour le partitionnement d'hypergraphes.

Tous ces outils utilisent un schéma multi-niveaux parallèle, mais les résultats fournis ne sont pas parfaits. Nous allons tenter d'expliquer pourquoi et de remédier à ces imperfections. Tout d'abord, nous rappellerons les notions de parallélisme qui seront utilisées dans la suite de cette thèse.

1.4.1 Vocabulaire du parallélisme

Dans la suite, nous considérerons que l'on dispose de p processeurs.

Définition 20 (Temps effectif parallèle, accélération)
Le temps effectif parallèle est la durée que l'utilisateur attend pour que le programme parallèle termine. Nous le noterons t_p dans la suite de ce document.

Le temps séquentiel est le temps mis pour résoudre le même problème avec le meilleur algorithme séquentiel connu. Nous le noterons t_s.

L'accélération (« speed-up » en anglais) a_p correspond au rapport t_p sur t_s :

$$a_p = \frac{t_p}{t_s} \ . \tag{1.17}$$

Sur des machines classiques (non quantiques) utilisant des algorithmes déterministes, l'accélération a_p est majorée par p. En effet, la capacité de calcul avec p processeurs étant p fois plus élevée que celle avec un seul processeur, le temps d'exécution peut être au plus divisé par p. Cette remarque nous permet de définir la notion d'efficacité.

Définition 21 (Efficacité, rendement)
L'efficacité (ou rendement) η_p est définie comme étant le quotient entre a_p et p. Elle est donc majorée par 1.

$$0 \leq \eta_p = \frac{a_p}{p} \leq 1 \ . \tag{1.18}$$

Nous dirons qu'un programme est *scalable* en temps quand son efficacité tend vers 1 lorsque le nombre de processeurs p tend vers $+\infty$. Par extension, nous dirons qu'un programme est scalable lorsque son temps d'exécution avec p processeurs t_p est divisé par un facteur p par rapport au

temps séquentiel de référence t_s. De même, nous dirons qu'un programme est scalable en mémoire lorsque l'occupation en mémoire par processeur lors d'une exécution avec p processeurs est p fois inférieure à celle obtenue en effectuant le calcul sur un seul processeur.

1.4.2 Formulation parallèle

Maintenant que nous avons défini des notions d'évaluation de la qualité d'un algorithme parallèle, nous pouvons définir plus précisément ce que signifie l'expression *partitionnement parallèle de graphes*.

Le problème du partitionnement reste celui posé à la définition 17 de la page 10, auquel s'ajoute une contrainte concernant l'utilisation des p processeurs disponibles : le partitionneur doit être *scalable* en mémoire et en temps.

Les surcoûts en mémoire et en temps dus aux communications entre les processeurs empêchent d'avoir une scalabilité parfaite, mais le but visé par toutes les méthodes proposées dans la littérature est de les minimiser autant que possible, tout en conservant un objectif qualitatif pour le résultat. Il est en effet souhaitable que la qualité du partitionnement produit par l'outil parallèle soit invariante par rapport au nombre de processeurs utilisés, et soit identique à celle obtenue en séquentiel.

Le besoin de scalabilité mémoire impose de travailler avec des graphes qui seront répartis sur l'ensemble des processeurs. Dans le cas où la machine parallèle utilisée ne dispose pas d'une mémoire partagée, nous sommes donc dans le cadre d'un algorithme parallèle distribué. Sauf mention contraire, nous nous placerons dans ce cadre d'utilisation et nous utiliserons des architectures employant des systèmes d'échanges de messages, tel MPI [61]. Dans ce cas, il sera très souvent possible d'identifier le nombre de processus au nombre de processeurs.

1.4.3 Résultats actuels en parallèle

L'outil de référence pour le partitionnement parallèle de graphes est ParMeTiS [46]. Ses résultats concernant la scalabilité en temps sont excellents mais la qualité du partitionnement se dégrade de manière très significative lorsque le nombre de processus augmente [CP06b].

Ce phénomène est dû à la difficulté d'obtenir une bonne parallélisation du schéma multi-niveaux. En effet, paralléliser les phases de contraction et d'expansion pose de nombreux problèmes, les principaux algorithmes usuels étant intrinsèquement séquentiels, qu'il s'agisse de l'algorithme de contraction ou des algorithmes de raffinement utilisés lors de l'expansion, comme Fiduccia-Mattheyses notamment. De plus, un relâchement de certaines contraintes durant la phase de contraction peut être à l'origine d'une importante perte de qualité, car l'hypothèse de proximité topologique entre les niveaux n'est alors pas forcément respectée.

L'optique de conserver en parallèle la même qualité que les meilleures approches séquentielles conduit néanmoins à conserver un tel schéma multi-niveaux. Des compromis scalabilité/qualité devront donc être étudiés, du moins lors des tentatives de parallélisation directe de l'existant.

Le chapitre 1.6 présentera comment nous avons parallélisé le schéma multi-niveaux, les chapitres 2 et 3 se concentrant pour leur part sur la phase de contraction et le raffinement durant l'expansion.

1.5 Un sous-problème : la renumérotation de matrices creuses

Cette section a pour but de présenter le problème de renumérotation de matrices creuses. Ce problème est présenté maintenant car c'est lui qui était notre but pratique lors de la réalisation

de PT-Scotch.

1.5.1 Notions d'algèbre linéaire creuse

Tout d'abord, il faut préciser que notre problème fait partie d'une problématique importante de l'algèbre linéaire creuse.

L'algèbre linéaire creuse est une restriction de l'algèbre linéaire classique qui consiste essentiellement en la résolution de grands systèmes d'équations, mais dont les inconnues n'ont que peu de dépendances entre elles (on dit aussi qu'elles sont faiblement couplées). Les matrices associées à de telles équations possèdent donc un grand nombre de termes nuls, appelés *zéros*. Le but de l'algèbre linéaire creuse est d'exploiter le plus efficacement possible ces zéros.

Nous nous plaçons dans le cas où notre but est de résoudre l'équation suivante, avec A une matrice $n \times n$ ($n \in \mathbb{N}^*$) à coefficients dans $\mathbb{R}$ ou $\mathbb{C}$, b un vecteur de $\mathbb{R}^n$ ou $\mathbb{C}^n$ et x le vecteur inconnu à trouver :

$$Ax = b \ . \tag{1.19}$$

Nous nous plaçons dans le cas où cette équation a une solution unique, quel que soit le second membre b, donc dans le cas où la matrice A est non-singulière, c'est-à-dire inversible.

Définition 22 (Inverse d'une matrice)
La matrice A est inversible si et seulement si il existe une matrice A^{-1} telle que $AA^{-1} = A^{-1}A = I$, où I est la matrice identité de taille n.

Nous nous intéressons maintenant aux *méthodes directes* de résolution de l'équation (1.19), c'est-à-dire aux méthodes qui se ramènent à calculer explicitement l'inverse A^{-1} de la matrice A, ce qui nous permet d'obtenir le résultat suivant :

$$x = A^{-1}b \ . \tag{1.20}$$

Définition 23 (Matrice triangulaire)
Une matrice carrée M est dite triangulaire à diagonale unitaire si et seulement si :

- *d'une part, elle est triangulaire, $\forall (i,j) \in [\![1;n]\!]^2, (i > j \Rightarrow m_{ij} = 0)$ ou $(i < j \Rightarrow m_{ij} = 0)$. Dans le premier cas, M est dite triangulaire supérieure, dans le second, triangulaire inférieure ;*
- *d'autre part, tous les termes de sa diagonale sont égaux à 1 : $\forall i \in [\![1;n]\!], m_{ii} = 1$.*

Théorème 1 (Condition nécessaire pour l'existance d'une factorisation *LU*)
Si A est une matrice non singulière, alors il existe une matrice de permutation P, une matrice L triangulaire inférieure à diagonale unitaire et une matrice U triangulaire supérieure telle que $A = PLU$.

Souvent, la matrice P peut être la matrice identité, dans ce cas, l'équation (1.19) devient donc :

$$LUx = b \Leftrightarrow Ly = b \text{ et } Ux = y \ . \tag{1.21}$$

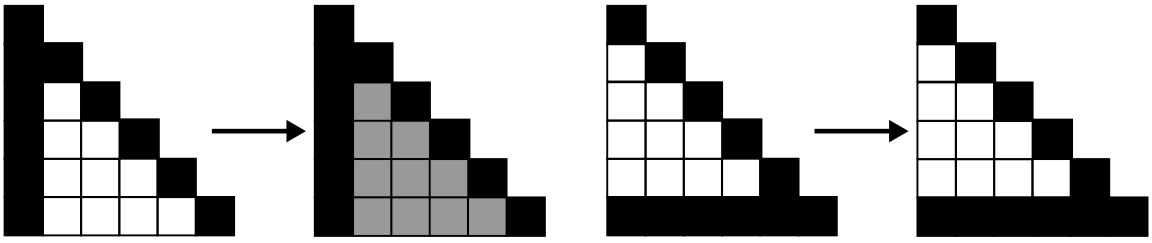
La complexité en temps de la factorisation LU de A est cubique (en $\Theta(\frac{n^3}{3})$), celle des résolutions d'équations de premier membre U et L est quadratique (en $\Theta(n^2)$). Le problème peut donc se résoudre de manière cubique par rapport au temps, à savoir en $\Theta(\frac{n^3}{3})$.

Définition 24 (Matrice définie positive)
Une matrice A est dite définie positive si et seulement si toutes ses valeurs propres sont strictement positives.

Théorème 2 (Condition nécessaire pour l'existance d'une factorisation de Cholesky) *Si A est une matrice symétrique définie positive, alors elle admet une factorisation de Cholesky, c'est-à-dire qu'il existe une unique matrice L triangulaire inférieure dont les termes diagonaux sont tous strictement positifs et telle que $A = LL^t$.*

En pratique, la taille n de la matrice est très grande (plusieurs millions), donc la méthode n'est pas exploitable si l'on ne tient pas compte du creux, c'est-à-dire des zéros, de la matrice A. D'autre part, beaucoup de problèmes réels comportent des symétries qui font que la matrice A est symétrique et très souvent définie positive, ce qui permet d'effectuer une factorisation de Cholesky $A = LL^t$ en $\Theta(\frac{n^3}{6})$.

Cependant, la place mémoire nécessaire pour stocker la matrice L peut fortement différer de celle nécessaire pour stocker la matrice A lorsqu'on tient compte du creux. Ce phénomène s'appelle le *remplissage* et dépend de l'ordre des inconnues ; la figure 1.10 permet de le visualiser.



(a) Remplissage total. (b) Absence de remplissage.

FIG. 1.10: Influence de la numérotation des inconnues dans le remplissage lors de la factorisation de Cholesky $A = LL^t$. Les termes non nuls de la matrice A sont représentés en noir, les termes de remplissage, c'est-à-dire les termes non nuls de L n'apparaissant pas dans A, en gris.

Il n'est pas toujours possible de rendre inexistant le remplissage de A, mais des techniques ont été mises en œuvre pour limiter son importance au maximum. En effet, le remplissage fait perdre la nature creuse du problème et nous ramène à résoudre un problème d'algèbre linéaire dense, qui va vite saturer les capacités de calcul et de stockage dont nous disposons.

De manière à calculer le plus rapidement possible les solutions, les algorithmes de résolution doivent tenir compte de l'indépendance des calculs afin de permettre une résolution plus rapide grâce à l'emploi de machines parallèles (calculateurs parallèles, grappes de stations, ...).

La méthodologie de résolution directe des systèmes creux se décompose en cinq phases :

1. Renumérotation de la matrice A : on cherche une matrice de permutation P telle que la factorisation de $P^{-1}AP$ en LL^t crée le moins de termes non nuls possibles (conservation du creux) et maximise l'indépendance des calculs (parallélisme). Dans les étapes suivantes, nous désignerons par A la matrice PAP^t.
2. Factorisation symbolique : on ne veut pas gérer dynamiquement les termes créés lors du calcul de L. On veut donc une structure statique connue à l'avance. La factorisation symbolique consiste à calculer cette structure creuse de L en fonction de celle de A.
3. Distribution et ordonnancement : on calcule une distribution des données et un ordonnancement des calculs à partir de la structure de L, dans le but d'optimiser le temps de calcul de la factorisation parallèle à venir.
4. Factorisation parallèle de la matrice A.

5. Résolution finale du système grâce à une descente-remontée[4] parallèle.

1.5.2 Optimisation du creux

Un objectif prioritaire de notre logiciel PT-Scotch était de fournir un renumeroteur parallèle performant et produisant des résultats de bonne qualité. Nous nous intéressons donc à la première phase de la méthodologie de résolution présentée précédemment.

Nous cherchons à améliorer la résolution du système de l'équation (1.19) dans le cadre d'une résolution directe par factorisation de Cholesky en optimisant la taille mémoire nécessaire et aussi le temps de la factorisation. Pour cela, nous devons trouver une renumérotation de la matrice A qui minimise le remplissage lors de la factorisation, c'est-à-dire que nous cherchons une matrice de permutation P telle que la factorisation de Cholesky de la matrice $P^{-1}AP$ nécessite le moins de place et soit la plus rapide possible.

Pour ce faire, nous travaillons généralement sur le graphe d'adjacence associé à la matrice A. La définition du graphe d'adjacence d'une matrice A est en quelque sorte le symétrique de celle de matrice d'adjacence, donnée à la définition 12. On obtient donc la définition suivante :

Définition 25 (Graphe d'adjacence)
La matrice A peut être modélisée par son graphe d'adjacence $G(A) = (V, E, \phi)$ tel que :
- *il y a autant de sommets dans V que de colonnes dans A, donc $|V| = n$;*
- *il existe une bijection $\phi : V \to [\![1; n]\!]$ qui associe à chaque sommet de V un numéro dans l'intervalle $[\![1; n]\!]$;*
- $\forall (u, v) \in V^2, \{u, v\} \in E \iff a_{\phi(u)\phi(v)} \neq 0$.

La recherche de la matrice de permutation P pour la matrice A correspond à la recherche d'une permutation p de $[\![1; n]\!]$ sur la numérotation ϕ des sommets du graphe.

Le graphe d'élimination est le graphe $G(L)$ associé à la matrice factorisée. $G(L) = G^* = (V, E^*, p)$ avec $E^* = E \cup R$ (R désigne l'ensemble des arêtes de remplissage associées aux termes créés).

L'algorithme de construction de G^* à partir de $G(A)$, obtenu par le déroulement l'algorithme de factorisation, est décrit par l'algorithme 5.

Algorithme 5 Calcul de G^* à partir de $G(A)$

Pour $k = 1$ à $n + 1$ **Faire**

Déterminer l'ensemble $Madj(n_k)$ des sommets x_i, $i > k$, adjacents à x_k (monotones adjacents).

Rajouter des arêtes pour transformer le graphe induit par $Madj(n_k)$ en graphe clique (arêtes de remplissage).

Fin de Pour

Maintenant que les notations sont définies, nous allons pouvoir étudier le problème du remplissage minimum de A, c'est-à-dire comment limiter la création d'arêtes de remplissage.

1.5.3 Problème de la minimisation du remplissage

L'étude de l'algorithme de construction de G^* à partir de $G(A)$ nous permet de remarquer qu'il y a création d'une arête entre x_i et x_j à l'étape k si x_i et x_j sont adjacents à x_k et que $k < i$ et $k < j$. On peut même généraliser en disant qu'il y création d'une arête de remplissage entre

[4] Descente : résolution de $Ly = b$; remontée : résolution de $L^t x = y$.

x_i et x_j lors de la construction du graphe d'élimination s'il existe un chemin, dans le graphe d'adjacence de A, allant du sommet x_i au sommet x_j qui ne passe que par des sommets d'indices plus petits que i et j (relation 1.22) :

$$\exists C \text{ chemin dans } G(A) \text{ entre } x_i \text{ et } x_j \text{ tel que } \forall x_k \in C \backslash \{x_i, x_j\}, k < i \text{ et } k < j$$
$$\Updownarrow$$
$$\text{arête de remplissage entre } x_i \text{ et } x_j \ . \quad (1.22)$$

La figure 1.11 permet de visualiser le remplissage d'une matrice et l'apparition des arêtes de remplissage dans son graphe d'élimination.

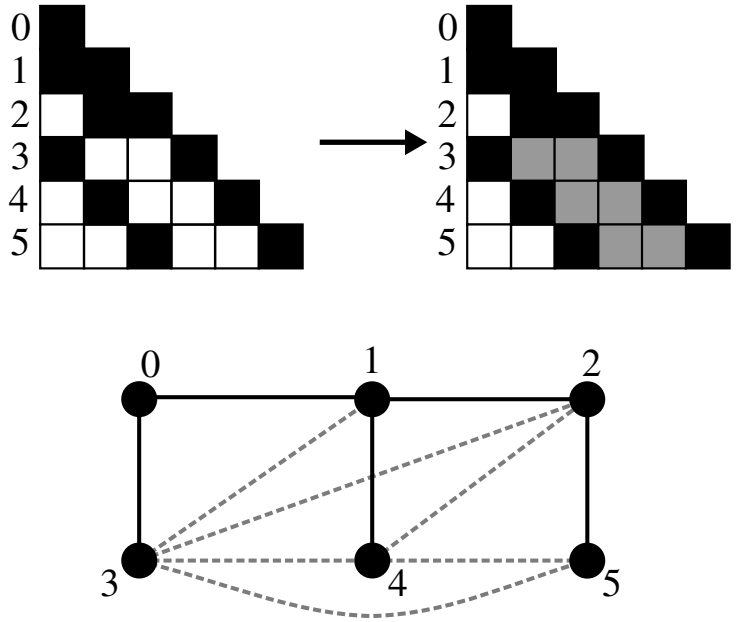


FIG. 1.11: Exemple de remplissage pour une matrice et visualisation de son graphe d'élimination (les arêtes de remplissage sont en tirets).

Le problème du remplissage de la matrice A se ramène donc à chercher une renumérotation du graphe d'adjacence $G(A)$ qui minimise le nombre d'arêtes de remplissage créées lors de la construction de G^*.

Grâce à la relation (1.22), le problème de minimisation du remplissage de A, devient sur le graphe d'adjacence $G(A)$:

Définition 26 (Problème du remplissage minimum)
Pour une matrice creuse A et son graphe d'adjacence $G(A) = (V, E, \phi)$, si $\mathcal{R}_p$ désigne l'ensemble des arêtes de remplissage associé à la permutation, le problème du remplissage consiste à la recherche d'une fonction de permutation p_{opt} telle que le nombre d'éléments de $\mathcal{R}_{p_{opt}}$ soit minimal.

$$\forall p, \forall \{u, v\} \in E, \{u, v\} \in \mathcal{R}_p \Longleftrightarrow \exists C \in \mathcal{C}_G(u, v), \forall w \in C \setminus \{u, v\}, p(\phi(w)) < p(\phi(u))$$
$$\text{et } p(\phi(w)) < p(\phi(v)) \ . \quad (1.23)$$

Cependant, il est aussi nécessaire de connaître l'ordre dans lequel les calculs de factorisation doivent être effectués. Celui-ci dépend des relations entre les inconnues qui peuvent être explicitées par l'arbre d'élimination dont la définition est fournie ci-après.

Définition 27 (Arbre d'élimination)
L'arbre d'élimination T associé à une renumérotation de la matrice A est défini comme étant l'arbre ayant n sommets reliés de telle sorte que :

$$\forall (i,j) \in [\![1;n]\!], i<j, \{\phi^{-1}(i),\phi^{-1}(j)\} \in E(T) \Longleftrightarrow j = \min_{k \ in [\![i+1,k]\!]} (k | \{\phi(k),\phi(i)\} \in E(G^*)) \ . \tag{1.24}$$

La présence d'une arête $\{i,j\}$, où $i<j$, dans l'arbre d'élimination signifie que le calcul de la j-ème colonne de la matrice nécessite que celui de la i-ème colonne soit déjà effectué. La connaissance de ces relations de dépendances entre les inconnues est cruciale pour le déroulement de la factorisation.

1.5.4 Solutions approchées pour le problème de minimisation du remplissage

Le remplissage minimum étant un problème NP-Complet [25], tous les programmes utilisés pour renuméroter les matrices utilisent des heuristiques afin de le résoudre. Les deux méthodes les plus fréquemment utilisées reposent sur les deux constats suivants :

- pour éviter le remplissage, on peut numéroter avec les plus grands numéros les sommets qui contribuent au plus de chemins possibles, pour les « casser », c'est-à-dire qu'ils ne puissent pas contribuer au remplissage au sens de la relation (1.22).
- le nombre de chemins passant par un sommet dépend du degré de celui-ci, mais les degrés des sommets peuvent augmenter lors du déroulement des heuristiques ; c'est pourquoi il faut respecter l'ordre d'élimination des sommets/inconnues. La méthode consiste alors à numéroter avec les plus petits numéros les sommets qui contribuent le moins possible aux chemins de remplissage.

1.5.4.1 Méthode du degré minimum

Cette méthode [78] consiste à sélectionner à chaque étape le sommet de plus petit degré puis de l'éliminer du graphe en formant une clique avec ses voisins. Ce sommet est renuméroté avec le plus petit numéro encore disponible. Le but de ce procédé est de placer dans les premières colonnes de la matrice les éléments qui ont potentiellement le moins de contributions au remplissage.

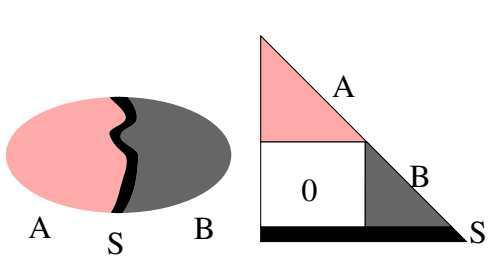
Cependant, on constate qu'au cours du déroulement de l'algorithme le degré de ses voisins peut augmenter. Ce phénomène rend cette méthode très séquentielle, puisqu'on ne peut pas sélectionner à l'avance les sommets suivants.

Cette heuristique est aussi un algorithme local puisqu'on ne s'intéresse qu'à un sommet et à ses voisins. Les arbres d'élimination ainsi produits comportent de nombreuse chaînes et ne sont donc que peu équilibrés et il en résulte généralement un mauvais parallélisme, ce qui peut être pénalisant lors de la phase de factorisation numérique.

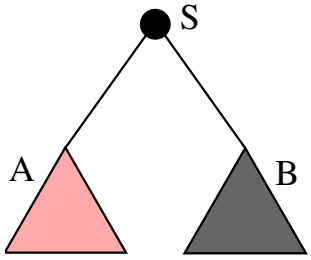
Ces deux défauts justifient la recherche d'autres solutions au problème du remplissage de la matrice.

1.5.4.2 Méthode des dissections emboîtées

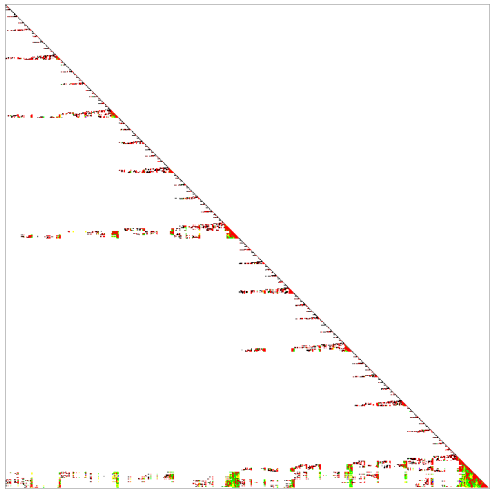
Cette méthode consiste à découper à chaque étape le graphe en deux parties principales A et B, où l'on réitérera le procédé, séparées par un ensemble de sommets S appelé séparateur. Ce dernier est numéroté avec les nombres les plus élevés, afin qu'il ne puisse pas y avoir de chemins de remplissage entre deux sommets n'appartenant pas à la même partie. Le séparateur est choisi de façon à ce que les deux parties ainsi délimitées soient de tailles équivalentes et que la taille du séparateur soit la plus petite possible, car il correspond à une zone a priori dense dans la matrice factorisée. On obtient donc une structure de matrice caractéristique de cette approche, comme le montre la figure 1.12a.



(a) Le principe de la dissection emboîtée.



(b) Arbre d'élimination associé à la dissection précédente.



(c) Exemple de structure de matrice obtenue à l'aide de dissections emboîtées récursives.

FIG. 1.12: Conséquences de la renumérotation à l'aide des dissections emboîtées.

Cet algorithme est de type « diviser pour résoudre » (« *divide & conquer* ») et, les sous problèmes étant indépendants, peut donc être parallélisé assez facilement. De plus, il s'agit d'un algorithme global qui nécessite la connaissance de tout le graphe pour pouvoir fonctionner. Il produit des arbres d'élimination équilibrés, ce qui facilitera l'exécution parallèle de la factorisation numérique de la matrice renumérotée. En effet, comme cela est illustré en figure 1.12b, les calculs de la factorisations peuvent être effectués en parallèle dans les deux sous-arbres associés à A et B.

La méthode des dissections emboîtées correspond donc à des bipartitionnements récursifs

par des séparateurs sommets du graphe d'adjacence G associé à la matrice A.

La taille des séparateurs va influencer directement le remplissage de la matrice A, tandis que la contrainte d'équilibre va surtout influencer la concurrence possible des calculs lors de la factorisation. Pour mesurer ces deux effets, nous utiliserons comme métrique le nombre d'opérations effectuées (additions, multiplications, soustractions et divisions) lors de la factorisation de Cholesky de A, et ce nombre sera noté OPC (« *OPeration Count* »).

1.6 Positionnement, mise en œuvre et protocole expérimental

L'objectif essentiel de notre travail était d'obtenir un outil parallèle efficace pour la renumérotation de matrices creuses. Nous avons écrit dans la section précédente que nous utilisions essentiellement la technique des dissections emboîtées pour renuméroter les matrices.

1.6.1 Approches classiques et limitations actuelles

Notre choix d'utiliser essentiellement la méthode des dissections emboîtées provient du fait que cette technique permet, d'une part, plus de parallélisme dans la factorisation numérique que celles utilisant des techniques de degré minimum [15] ou de remplissage minimum [1], mais surtout, elle produit souvent de meilleurs résultats. En pratique, la stratégie par défaut de Scotch est de coupler ces deux techniques [67], les heuristiques de type degré minimum semblant plus performantes sur les petits graphes (de taille inférieure à 120 sommets par défaut) que les dissections emboîtées, et le parallélisme ayant déjà été extrait lors des premiers niveaux de dissections.

De nombreux partitionneurs séquentiels de graphes proposent une fonctionnalité de renumérotation de matrices creuses. Nous pouvons citer, parmi les plus connus MeTiS [46], Jostle [82] ou encore Scotch [68] qui est devéloppé dans l'équipe ScAlApplix de l'INRIA Futurs de Bordeaux et au sein de laquelle j'ai effectué ma thèse.

Cependant, il est actuellement assez difficile de renuméroter des graphes de plus d'une dizaine de millions de sommets car la mémoire requise est trop importante. Or, les autres phases nécessaires à la résolution des systèmes étant, de nos jours, généralement parallèles et distribuées, l'étape de renumérotation devient le goulet d'étranglement de la chaîne de résolution.

Il est donc nécessaire de paralléliser cette étape, en utilisant des partitionneurs parallèles de graphes. Cependant, si les partitionneurs parallèles de graphes ou d'hypergraphes sont relativement nombreux, seul ParMeTiS [46] dispose réellement d'un module de renumérotation parallèle de graphes. Ce dernier repose aussi sur une technique de dissections emboîtées, mais la qualité des résultats qu'il fournit à tendance à décroître rapidement lorsque le nombre de processeurs utilisés augmente. L'exemple fourni en figure 1.13 montre les conséquences en nombre d'opérations nécessaire à la factorisation séquentielle de la renumérotation de la matrice `bmw32` avec le logiciel ParMeTiS. On observe que la factorisation utilisant la permutation produite avec 64 processeurs est 248.42% plus coûteuse que celle utilisant la renumérotation calculée sur un processeur.

Une autre limitation de ParMeTiS est que son module de renumérotation ne peut fonctionner que sur un nombre de processeurs qui est une puissance de deux. Ce phénomène peut s'avérer très pénalisant, le fait d'avoir une machine à 768 processeurs n'apportant par exemple aucun avantage par rapport au fait de posséder une machine à 512 processeurs.

Nous allons néanmoins adopter une méthodologie de renumérotation assez semblable à celle de ParMeTiS en utilisant la technique des dissections emboîtées associée à un schéma multi-

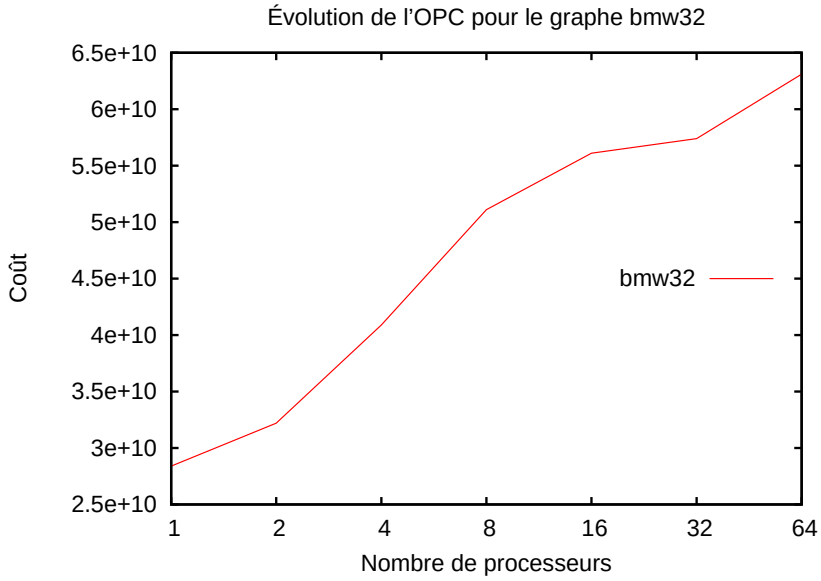


FIG. 1.13: Nombre d'opérations induites par la renumérotation parallèle de la matrice `bmw32` avec PARMETIS.

niveaux, cette stratégie étant déjà celle implantée dans la version séquentielle de SCOTCH.

Nous allons maintenant observer quels mécanismes peuvent être utilisés pour paralléliser cette approche de type dissections emboîtées.

1.6.2 Parallélisme intrinsèque des dissections emboîtées

L'aspect intrinsèquement parallèle de l'algorithme des dissections emboîtées apparaît dans le nom même de la technique, au travers de l'adjectif « emboîté ».

Observons ce qui se passe lors de l'exécution de l'algorithme. Après le premier bipartitionnement du graphe en sous-graphes A, B et S, pour reprendre les notations de la page 28, les sous-graphes A et B vont être eux aussi bipartitionnés, de manière indépendante. L'aspect récursif des dissections emboîtées est donc à l'origine d'un parallélisme de haut niveau.

Pour exploiter celui-ci, nous calculons effectivement les partitionnements des graphes A et B en parallèle, en essayant d'optimiser au mieux les calculs. Les sous-graphes A et B ont une taille environ deux fois plus petite que celle du graphe original, mais il est très probable qu'ils soient eux aussi distribués sur l'ensemble des p processeurs. Notre idée est d'augmenter la localité des données en effectuant un repliement de ces deux sous-graphes sur une moitié des processeurs chacun. Cela permettra une localisation et une réduction des volumes des communications, et diminuera donc la congestion du réseau.

Le repliement permet ainsi de séparer les deux calculs non seulement logiquement mais aussi physiquement. Par exemple, dans le cas où nous disposons de deux nœuds SMP, chaque sous-graphe pourra être bipartitionné sur un nœud, sans perturber les communications inter-processeurs de l'autre.

Le repliement proprement dit est disponible en deux versions au sein de PT-SCOTCH : une première version threadée, c'est-à-dire que les reconstructions des deux sous-graphes se font simultanément, et une seconde version séquentielle, pour les systèmes dont la bibliothèque MPI

n'est pas « *thread-safe* ». L'intérêt de ce parallélisme utilisant les threads est qu'il permet de recouvrir les communications nécessaires à la construction des deux sous-graphes. La figure 1.14 illustre le processus de repliement et de redistribution nécessaire à l'exécution du deuxième niveau de dissection.

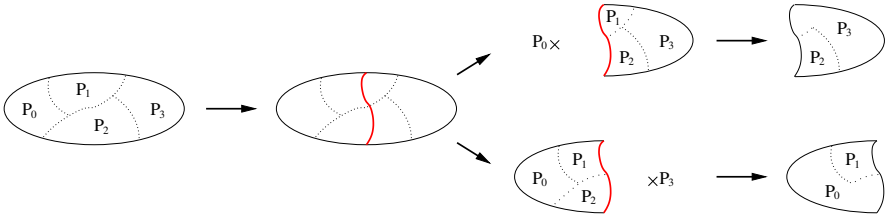


FIG. 1.14: Redistribution des sous-graphes lors du deuxième appel au bipartitionnement, dans le cadre des dissections emboîtées.

Cette phase de repliement et de redistribution se reproduit jusqu'à ce que chaque sous-graphe ne soit plus porté que par un seul processeur, moment à partir duquel l'exécution de la méthode des dissections emboîtées s'effectue au moyen de la version séquentielle des routines de calcul des dissections.

Le parallélisme induit par cette phase est idéal à exploiter, les calculs d'un même niveau de dissection étant totalement indépendants. De plus la localité apportée par le repliement permet de basculer le plus tôt possible dans un schéma séquentiel, plus performant et pouvant profiter des heuristiques purement séquentielles dans le cas où celles-ci sont plus efficaces.

Il faut aussi noter que notre algorithme de repliement gère aussi le cas où le nombre de processeurs n'est pas une puissance de deux. Ce cas n'est, en effet, pas géré par PARMETIS qui nécessite de toujours obtenir un nombre pair de processeurs lors de la récursion. PT-SCOTCH peut faire un repliement sur un nombre impair p de processeurs, le premier sous-graphe étant redistribué sur $\lceil \frac{p}{2} \rceil$ processeurs, et le second sur les $\lfloor \frac{p}{2} \rfloor$ restants.

Nous avons ainsi un parallélisme parfait entre les différents membres d'un même niveau de dissection. Cependant, comme le calcul du bipartitionnement pour un graphe du niveau k s'effectue sur environ $\frac{p}{2^k}$ processeurs, il est également nécessaire de paralléliser l'algorithme de bipartitionnement.

Afin de préserver une bonne qualité de partitionnement, nous avons choisi de conserver une approche multi-niveaux [10], que nous devons donc paralléliser. Comme nous l'avons déjà évoqué dans la section 1.3 de la page 18, le schéma multi-niveaux pour le partitionnement de graphes se décompose en trois phases distinctes.

Seule une de ces trois étapes ne nécessite pas d'être parallélisée : l'étape de partitionnement initial. En effet, celle-ci concerne uniquement le graphe de plus petite taille obtenu lors de l'exécution de l'algorithme, et son nombre de sommets, de l'ordre de la centaine, fait qu'il est totalement inutile d'envisager sa parallélisation.

Les étapes de contraction et d'expansion nécessitent, en revanche, d'être parallélisées. Les détails concernant cette mise en œuvre seront respectivement donnés dans les chapitres 2 et 3. Concernant l'expansion, nous nous sommes aussi également intéressés à la recherche d'algorithmes hautement parallèles et c'est pourquoi nous présenterons dans le chapitre 4 une étude sur l'utilisation des algorithmes génétiques dans ce contexte.

1.6.3 Structure de graphe distribué

Une étape indispensable avant la conception des algorithmes parallèles est la définition des structures de données qui seront manipulées.

Nos algorithmes de partitionnement parallèle de graphes manipulant des graphes distribués sur un ensemble de p processeurs, nous allons présenter ici la façon dont leurs données sont distribuées au sein de PT-Scotch.

De la nature et de la distribution des données dépendent grandement les performances des algorithmes que nous voulons développer.

Principe de base du graphe distribué

Le principe de la distribution que nous utilisons est simple : chaque processeur p_i possède un sous-ensemble $V_{|p_i}$ de sommets et le sous-ensemble $E_{|p_i}$ des arêtes incidentes aux sommets de $V_{|p_i}$. Les sommets $V_{|p_i}$ sont les sommets locaux au processeur p_i, les sommets $\bigcup_{p_j \neq p_i} V_{|p_j}$ sont les sommets distants. La famille $\{V_{|p_i}\}_{p_i \in [\![1;p]\!]}$ est une partition de l'ensemble des sommets V. Les arêtes reliant des sommets situés sur deux processeurs différents sont donc dupliquées sur ces deux processeurs. Dans le cadre de Scotch, les arêtes étaient déjà, en séquentiel, stockées comme des listes d'adjacence pour chaque sommet, ce qui revient à représenter une arête comme deux arcs de sens opposé. La duplication nécessaire à la distribution était donc déjà présente au sein de la version séquentielle et ne provoque donc pas de surcoût mémoire.

Afin de pouvoir effectuer la plupart des calculs sur le graphe au niveau local, on utilise le couple $(V_{|p_i}, E_{|p_i})$ comme un sous-graphe de G. Pour éviter des communications avec les sommets distants, ceux-ci sont répliqués en local sous le nom de « sommets fantômes ». La gestion du sous-graphe est proche de celle de la version séquentielle de Scotch, y ajoutant simplement la notion d'un halo de sommets fantômes, un peu à la manière de ce qui avait déjà été réalisé pour le couplage des méthodes de dissections emboîtées et du degré minimum [67].

La structure séquentielle de graphe dans Scotch comporte essentiellement deux tableaux associés aux sommets et aux arêtes [64, pages 47-48]. Le premier tableau est indicé par les sommets et permet d'accéder au sous-tableau d'arcs associé à chaque sommet, grâce à l'indice du premier arc. Les sommets locaux, appartenant à $V_{|p_i}$ sont les seuls à disposer de la connaissances des arcs, les sommets fantômes n'étant présents que pour permettre de qualifier certains états propres aux sommets, comme un drapeau indiquant l'état du sommet ou un entier indiquant à quelle partie appartient le sommet. Les sommets fantômes ne participent donc pas à une extension de la connaissance topologique du sous-graphe, mais servent de zone de recouvrement pour les différents sous-graphes $(V_{|pi}, E_{|p_i})$, à la manière de ce qui se fait fréquemment lors des décompositions de domaines.

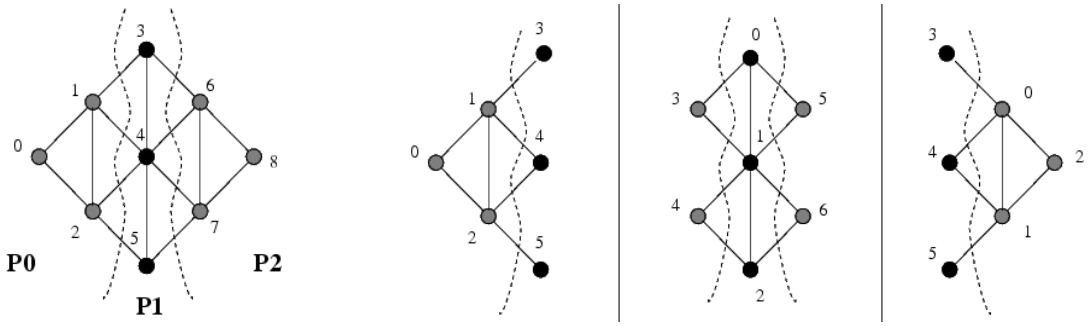
Afin de pouvoir facilement identifier un tel sous-graphe à un graphe Scotch centralisé sur le processeur p_i, nous utilisons un système de double numérotation des sommets. Les sommets portent, d'une part, un numéro global, qui permet de les identifier dans l'ensemble du graphe distribué ; d'autre part, ils possèdent aussi une numérotation locale, valide seulement sur le processeur courant, et qui est compatible avec sa numérotation globale, ainsi qu'avec la numérotation standard des graphes centralisés Scotch. En effet, l'indice de référence pour le parcours des tableaux de la structure de graphe centralisé peut prendre les valeurs 0 ou 1, selon la valeur de départ de la numérotation du graphe ou selon le contexte d'utilisation de la bibliothèque, les tableaux commençant à 0 en C, mais à 1 en Fortran. La compatibilité de la numérotation globale signifie que l'ordre sur la numérotation des sommets est conservé pour les sommets locaux.

En fait, la numérotation globale se déduit de la numérotation locale en ajoutant à celle-ci un décalage propre au processeur, celui-ci nécessitant seulement d'être strictement supérieur au numéro le plus grand parmi les sommets appartenant aux processeurs de rangs inférieurs. Dans le cas où le décalage est l'entier immédiatement suivant, on a une numérotation globale continue, sinon on obtient une numérotation à trous. Le numéro global d'un sommet est donc croissant selon le numéro du processeur concerné.

Un tableau, commun à tous les processeurs, permet de connaître l'intervalle des numéros de sommets valides pour chaque processeur. La stricte croissance de la numérotation rend possible l'utilisation d'une dichotomie pour connaître à quel processeur appartient un sommet identifié par son numéro global.

Les sommets fantômes, dont l'existence n'est que locale, sont numérotés après les sommets locaux et sont classés selon leur numéro global. Les sommets fantômes sont donc regroupés par appartenance à un processeur.

Un exemple de distribution est donné en figure 1.15. Elle représente la distribution du graphe illustré figure 1.15a sur trois processeurs, p_0, p_1 et p_2. La figure 1.15b permet de visualiser quelles sont les données possédées par chaque processeur. On peut remarquer que le coût mémoire des sommets fantômes est loin d'être négligeable dans notre cas, et qu'il est très dépendant de la manière de répartir les sommets entre les différents processeurs. Ce surcoût explique que nous ne pourrons pas atteindre une scalabilité mémoire parfaite.



(a) Le graphe que l'on souhaite distribuer, avec sa numérotation globale.

(b) Données possédées par chaque processeur, avec leurs numérotations locales.

FIG. 1.15: Exemple de structure de graphe distribué sur trois processeurs.

La figure 1.16 montre les données locales, propres à un processeur, et les données globales, dupliquées sur tous les processeurs d'un graphe distribué.

Les champs dupliqués contiennent la base d'indice des tableaux (`baseval`), le nombre de sommets et d'arêtes du graphe distribué (`vertglbnbr` et `edgeglbnbr`), le nombre de processeurs sur lesquels est distribué le graphe (`procglbnbr`), le nombre de sommets que possèdent les processeurs (`proccnttab`), les intervalles de numéros de sommets qui leur sont associés (`procdsptab`) pour pouvoir gérer les graphes avec une numérotation à trous dans la renumérotation globale.

Les champs locaux sont très semblables à ceux de la structure `Graph` de la bibliothèque SCOTCH, qui correspond à la représentation d'un graphe centralisé sur un processeur. On peut notamment citer les champs relatifs aux nombres de sommets et d'arêtes possédées localement (`vertlocnbr` et `edgelocnbr`), le nombre de sommets locaux et fantômes (`vertgstnbr`), deux tableaux de taille `vertlocnbr` permettant de connaître l'indice du premier et du dernier arc relatif à un sommet et de fournir ainsi les indices pour les tableaux d'arêtes qui contiennent le

numéro local du voisin (`edgegsttab`) et son numéro global (`edgeloctab`).

Type abstrait graphe distribué

Le graphe distribué est donc le type abstrait de données sur lequel la version parallèle de Scotch repose. Nous le nommerons `Dgraph` (pour « *distributed graph* ») son nom dans l'implantation PT-Scotch. Nous n'allons pas ici détailler sa signature complète, mais simplement passer en revue ses principales fonctionnalités qui nécessitent une description :

- `dgraphLoad` : permet de charger un graphe distribué ;
- `dgraphScatter` : permet de distribuer un graphe centralisé sur une structure `Dgraph` ;
- `dgraphGather` : permet de centraliser un graphe distribué ;
- `dgraphHalo` : permet de diffuser des informations relatives aux sommets sur tous les processeurs du graphe distribué, y compris aux sommets fantômes.

La routine `dgraphLoad` crée un graphe distribué à partir de sa description contenue dans un ou plusieurs fichiers. La distribution se fait par tranches de l'ensemble des sommets. Les intervalles de de sommets distribués à chaque processeur dépendent de la numérotation du graphe fournie par l'utilisateur, ce qui peut avoir d'importantes conséquences sur l'efficacité de la distribution.

La routine `dgraphGather` est nécessaire notamment pour pouvoir exploiter les routines séquentielles de Scotch, qui travaillent sur un graphe centralisé. En effet, `dgraphGather` permet de construire ce graphe afin de pouvoir appeler la routine séquentielle.

La fonction `dgraphHalo` permet d'associer des informations relatives aux sommets à tous les sommets fantômes, c'est-à-dire qu'elle met à jour les informations des sommets fantômes à partir de celles des sommets réels qui leurs sont associés, situés sur d'autres processeurs. Par exemple, en se basant sur la figure 1.15, si l'état associé au sommet de numéro global 6 est 0, l'exécution de `dgraphHalo` va aussi associer l'état du sommet fantôme 5 du processeur p_1 à 0.

Les algorithmes développés pour le partitionnement des graphes centralisés reposent essentiellement sur ces fonctionnalités.

1.6.4 Protocole expérimental d'évaluation dans le contexte de la renumérotation

Avant de détailler, dans les chapitres suivants, comment nous avons parallélisé la phase de bipartitionnement, nous allons brièvement décrire comment nous avons évalué les améliorations apportées à nos algorithmes.

Les divers progrès pouvant être amenés par nos algorithmes ne sont pas tous observables de la même façon : l'accélération temporelle est assez simple à déceler, mais ce n'est pas forcément le cas des améliorations qualitatives.

Métrique de comparaison

Comme nous l'avons énoncé à la fin du chapitre 1, nous avons choisi d'utiliser le nombre OPC d'opérations flottantes effectuées lors de la factorisation numérique séquentielle de la matrice numérotée.

L'intérêt immédiat de ce choix est qu'il permet de comparer facilement la conséquence de deux numérotations d'une même matrice, tout en étant quasi-indépendant du *solveur*[5] utilisé.

[5]Solveur :logiciel de résolution de systèmes d'équations linéaires.

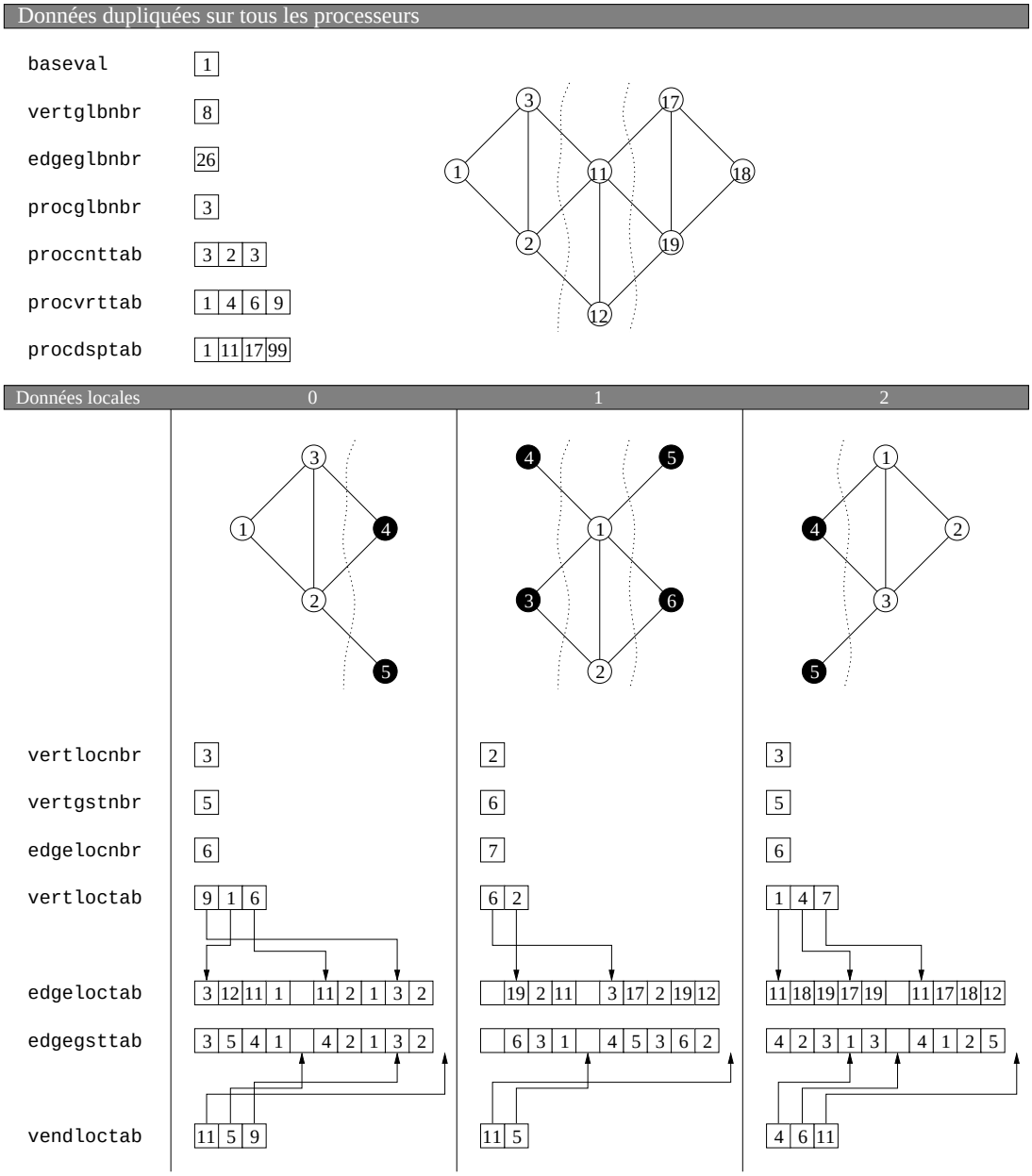


FIG. 1.16: Exemple de distribution des données sur 3 processeurs dans le cas d'un graphe avec une numérotation à trous.

Deux solveurs basés sur le même algorithme vont effectivement effectuer le même nombre d'opérations.

Cependant, malgré cet aspect pratique, il présente des défauts qui peuvent se montrer rédhibitoires dans certains cas. Le principal défaut vient du constat qu'il n'y a pas de corrélation forte entre la quantité de calculs nécessaires à la factorisation et le temps nécessaire pour la réaliser. Effectivement, les calculs peuvent être quelquefois effectués par blocs et il devient alors possible pour les solveurs d'exploiter les capacités superscalaires des processeurs modernes, par l'utilisation des bibliothèques optimisées de type *BLAS*[6] [19, 52].

Le cas des solveurs parallèles est plus complexe encore, ceux-ci exploitant le parallélisme de l'arbre d'élimination des inconnues dans le but d'accélérer les calculs. Deux numérotations ayant le même OPC peuvent donc donner deux arbres d'éliminations différents et donc des temps de résolution des systèmes eux aussi très dissemblables. La métrique OPC ne permet donc pas d'évaluer le degré de parallélisme induit par la renumérotation.

Un autre point négatif concernant l'utilisation de cette métrique est que nous avons surtout travaillé l'aspect dissections emboîtées en nous préoccupant essentiellement de la taille des séparateurs obtenus et de l'équilibre des partitions, mais dans le cadre global de la renumérotation de graphes d'autres paramètres peuvent aussi entrer en compte, comme la façon de numéroter les séparateurs et aussi les effets de couplage avec certaines approches locales de type degré minimum. Une amélioration légère des résultats produits par l'algorithme de partitionnement n'est donc pas forcément visible avec le seul OPC de la renumérotation.

Nous avons néanmoins retenu cette métrique car elle nous permet d'être indépendante du solveur utilisé et quantifie somme toute assez correctement la qualité d'une renumérotation. Il faudra, cependant, garder à l'esprit que de faibles écarts en OPC ne sont pas forcément significatifs.

Matrices et graphes utilisés

Nous avons testé nos algorithmes de renumérotation sur des graphes de test générés automatiquement, comme des grilles 2D ou 3D, mais aussi principalement sur des graphes provenant de diverses applications industrielles.

Le tableau 1.1 présente certains des graphes que nous avons utilisés comme cas tests dans la suite de ce document. Les nombres de sommets et d'arêtes quantifient la taille du graphe, le degré moyen permettant, pour sa part, d'estimer assez souvent la topologie du graphe ; la valeur de l'OPC fournit, en mesurant le coût de la factorisation, une estimation de la difficulté de la factorisation. La valeur de l'OPC indiquée par le tableau est celle obtenue en utilisant la stratégie par défaut de Scotch.

La plupart de ces graphes proviennent du domaine public et sont donc accessibles à toute personne voulant comparer ses résultats à ceux de PT-Scotch, fournis dans ce document. Les graphes dont la provenance est indiquée « Florida » proviennent de la collection de matrices de l'université de Floride [18], celles appelées « Parasol » sont des cas tests collectés dans le cadre de l'ancien projet européen Parasol [63].

Moyens utilisés

Scotch est une bibliothèque pouvant être utilisée sur tout système respectant la norme POSIX [41]. Elle réclame pour sa compilation, en plus des outils standards (compilateurs C, gestionnaire de compilation, commandes de manipulation de fichiers, système de base), Lex et

[6] BLAS : Basic Linear Algebra Subprograms.

Graphe	**Taille** ($\times 10^3$)		$\bar{\delta}$ **: degré moyen**	**OPC**	**Provenance**
	$\|V\|$	$\|E\|$			
altr4	26	163	12.50	3.46e+8	Maillage, CEA-Cesta
conesphere1m	1 055	8 023	15.21	1.81e+12	Maillage, CEA-Cesta
coupole8000	1 768	41 657	47.12	7.46e+10	Maillage, CEA-Cesta
audikw1	944	38 354	81.28	5.28e+12	Maillage 3D, Parasol
bmw32	227	5 531	48.65	3.21e+10	Matrice de rigidité 3D
bcsstk32	45	985	44.16	1.30e+9	Matrice de rigidité
oilpan	74	1 762	47.77	2.97e+10	Matrice de rigidité 3D
thread	30	2 220	149.32	3.85e+10	Parasol
bone010	986	35 339	35.82	4.58e+12	Structure 3D d'os, Florida
G3_circuit	1 585	3 037	3.84	5.54e+10	Simulation électronique, Florida
cage14	1 505	12 812	17.02	2.50e+15	Électrophorèse ADN, Florida
cage15	5 154	47 022	18.24	4.06e+16	Électrophorèse ADN, Florida
thermal2	1 228	3 676	5.98	1.57e+10	Simulation thermique, Florida
Qimonda07	8 613	29 143	6.76		Circuit électronique
brgm	3 699	151 940	82.14	2.44e+13	Géophysique
23millions	23 114	175 686	7.60	1.29e+14	CEA/CESTA

TAB. 1.1: Présentation de quelques graphes utilisés pour nos tests. La colonne OPC fournit le nombre d'opérations de la factorisation séquentielle de la matrice obtenue après renumérotation par SCOTCH séquentiel.

YACC (ou leurs équivalents modernes FLEX et BISON) lors de la compilation, pour son module d'analyse des chaînes de stratégies de partitionnement et de renumérotation.

PT-SCOTCH réclame en plus de cela la présence d'une bibliothèque MPI. Celle-ci doit au moins supporter le standard MPI-1.2, mais le support du dernier standard MPI-2.0 [61] est un plus, qui permet de pouvoir utiliser les threads POSIX (« *pthreads* ») lors de l'exécution.

Nous avons effectué des tests sur plusieurs machines d'architectures matérielles et logicielles différentes, mais dans ce document nous présenterons uniquement les temps obtenus sur la machine *decrypthon* du pôle M3PEC de l'université de Bordeaux I [55]. En effet, c'est une des seules machines facilement disponibles qui soit de taille suffisante pour conduire des expérimentations jusqu'à une centaine de processeurs. De plus, elle dispose d'un système de traitement par lots qui permet une certaine fiabilité dans la prise de temps, en empêchant les perturbations par les autres utilisateurs. Néanmoins, une limitation de ce gestionnaire de ressources concerne la stratégie d'allocation mémoire : il faut préciser pour chaque processus une limite mémoire, que les processeurs ne pourront pas dépasser. Le fait de ne pas pouvoir fixer globalement une limite mémoire est contraignant pour des applications de type partitionnement de graphes, car il est alors nécessaire de gérer au mieux la répartition mémoire sur l'ensemble des processeurs.

La machine *decrypthon* est une machine de type IBM P575, comportant 128 processeurs répartis en 16 nœuds octoprocesseurs, interconnectés par un réseau FEDERATION à 12 Gb/s, chaque nœud disposant de deux cartes réseau. Les processeurs sont des IBM POWER5 bi-cœurs fonctionnant à 1.5 GHz et se partageant 32 Go de mémoire vive par nœud. La bibliothèque MPI utilisée sur cette machine est l'implantation d'IBM qui est compatible MPI-1.2, mais qui fournit aussi une grande partie des fonctionnalités MPI-2. Elle est de plus compatible avec le niveau de thread `MPI_THREAD_MULTIPLE`, qui permet une utilisation concurrente et indépendante de la

bibliothèque MPI par plusieurs threads POSIX.

Les autres machines que nous avons utilisées étaient des PC GNU/Linux compatibles x86, ainsi que des clusters d'AMD Opteron compatibles x86-64. Nous avons employé sur ces machines la bibliothèque MPI MPICH-2 [32], qui supporte l'utilisation concurrente des threads POSIX. Nous avons aussi effectué quelques tests avec d'autres implantations libres comme OpenMPI [23], dont le support des threads n'est hélas pas encore officiel.

La portabilité du code a donc elle aussi été testée, dans la mesure de la disponibilité de machines différentes. Aucun problème majeur de portabilité n'a été relevé à cette occasion, que l'architecture soit 32 ou 64 bits.

Nous allons maintenant détailler la conception et la mise en œuvre de la phase de contraction parallèle.

Chapitre 2

La phase de contraction

Sommaire

Ce chapitre présente nos contributions quant à la parallélisation de la première phase du schéma multi-niveaux : la contraction. Nous allons tout d'abord commencer par décrire cette phase, brièvement présentée à la section 1.3.1 de la page 18.

2.1 Principe de fonctionnement et problèmes en parallèle

2.1.1 Principe de la contraction

Le principe du schéma multi-niveaux est de permettre d'obtenir un optimum le plus global possible de la qualité des partitionnements obtenus tout en utilisant l'algorithme d'optimisation globale seulement sur un graphe réduit où il sera moins coûteux de le faire fonctionner. Le résultat trouvé sur ce graphe réduit sera alors projeté de proche en proche lors de la phase d'expansion, afin qu'il corresponde à un optimum lui aussi le plus global possible pour notre graphe de départ. La figure 2.1 illustre le déroulement de cette phase de contraction.

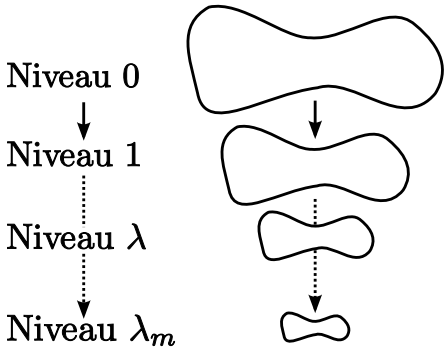


FIG. 2.1: Schéma de la phase de contraction. Le graphe initial G, dit de niveau 0, est contracté en un graphe plus petit de niveau 1, qui est à son tour contracté, jusqu'au niveau λ_m. Les éléments de la suite $(G_k)_{k\in[\![0,l]\!]}$ doivent rester topologiquement proches du graphe G.

Nous avons indiqué plusieurs fois lors de la présentation de la méthode multi-niveaux (dans la section 1.3.3 notamment), que l'efficacité de celle-ci reposait surtout sur l'hypothèse de proximité topologique entre les niveaux.

2.1.2 Problèmes soulevés par la parallélisation de la contraction

La procédure habituellement utilisée en séquentiel consiste à choisir un sommet au hasard et à l'apparier avec le voisin libre auquel il est relié par l'arête de plus grand poids. Cette technique est appelée *Heavy Edge Matching*. La version séquentielle de cet algorithme est donnée par l'algorithme 6 de la page suivante. En modifiant la condition de la ligne 11, on peut adopter un autre critère pour la sélection du sommet avec lequel s'apparier, comme par exemple la sélection aléatoire d'un voisin, sans se soucier du poids des arêtes.

Nous pouvons remarquer que, le parcours des sommets étant aléatoire, il ne devrait pas poser de problème lors du passage en parallèle : il ne sera pas nécessaire d'ajouter de synchronisation. Cependant, le résultat obtenu dépend de l'ordre de visite, et la connaissance des voisins libres au moment du choix de l'appariement est indispensable, ce qui implique en revanche de réaliser des communications pour connaître l'état de tous les sommets.

Plusieurs implantations parallèles se restreignent à faire uniquement, ou du moins à privilégier, des appariements locaux, c'est-à-dire entre sommets appartenant au même processeur, ceci

Algorithme 6 Algorithme séquentiel d'appariement.

```
1: Fonction Appariement(G : graphe)
   L : liste des paires de sommets pour la contraction
2:     L ← ∅
3:     Marquer tous les sommets comme « disponibles »
4:     Tant que (il existe un sommet « disponible ») Faire
5:         Choisir aléatoirement un sommet u
6:         Si (tous les voisins de u sont « appariés ») alors
7:             L ← {L, (u, u)}
8:             Marquer u comme « apparié »
9:             Continuer
10:        Fin de Si
11:        Choisir aléatoirement un voisin v disponible parmi ceux dont l'arête est de plus grand
   poids
12:        L ← {L, (u, v)}
13:        Marquer u et v comme « appariés »
14:    Fin de Tant que
15: Fin de Fonction
```

dans le but d'éviter les communications inter-processus pour s'informer de l'état des sommets voisins appartenant à un autre processeur. On observe alors une perte de qualité dans l'appariement, celui-ci étant biaisé et ne conservant pas aussi bien la topologie du graphe. D'autre part, les performances de tels algorithmes sont assez faibles en ce qui concerne les taux de contraction obtenus : dans le cas extrême où chaque sommet ne possède pas de voisin local, cette stratégie ne permet aucun appariement.

Il paraît donc nécessaire de conserver la liberté quant au choix du voisin avec lequel le sommet va s'apparier. Nous allons maintenant présenter comment nous avons effectué cette parallélisation et comment nous l'avons mise en œuvre afin de tenter de recouvrir les temps de communications.

2.2 Parallélisation de l'appariement

La méthode synchrone d'appariement que nous avons développée est décrite dans l'algorithme 7. Celui-ci est, sur le principe, assez proche de la version séquentielle de l'algorithme 6. La principale différence tient en l'apparition d'un nouvel état « réservé » pour le marquage des sommets qui sont potentiellement appariés avec un sommet situé sur un processeur distant, et surtout à l'existence d'une phase de communication collective à la ligne 23. La condition de terminaison de la boucle de l'algorithme (ligne 7), qui correspond à la boucle issue de la version séquentielle, porte sur l'existence d'un sommet « appariable » et non d'un sommet « libre ». En effet, un sommet dont tous les voisins sont marqués comme « non-libres », mais dont il existe parmi ceux-ci au moins un sommet marqué « réservé », pourra encore être apparié lors de l'itération suivante, si l'appariement inter-processus du sommet réservé échoue.

La fonction Faire_Appariements_Distants a pour but de valider les appariements entre sommets appartenant à des processeurs différents. C'est elle qui est responsable de la synchronisation des différents processus avant de réitérer la boucle de recherche de candidats pour l'appariement.

Algorithme 7 Algorithme parallèle d'appariement.

```
1: Fonction Appariement(G : graphe distribué sur les p processeurs)
   L_c : liste des paires de sommets pour la contraction sur le processeur courant
   L_échange : liste des paires de sommets d'appariements potentiels non locaux
2:     L_c ← ∅
3:     Marquer tous les sommets locaux comme disponibles
4:     Tant que (des appariements peuvent être réalisés) Faire
5:         Marquer les sommets réservés comme disponibles
6:         L_échange ← ∅
7:         Tant que (il existe un sommet local « appariable ») Faire
8:             Choisir aléatoirement un sommet u
9:             Si Tous les voisins de u sont appariés alors
10:                L_c ← {L_c, (u, u)}
11:                Marquer u comme apparié
12:                Continuer
13:            Fin de Si
14:            Choisir aléatoirement un voisin v disponible parmi ceux dont l'arête est de plus
   grand poids
15:            Si (v n'appartient pas au processeur courant) alors
16:                L_échange ← {L_échange, (u, v)}
17:                Marquer u et v comme réservés
18:            Sinon
19:                L_c ← {L_c, (u, v)}
20:                Marquer u et v comme appariés
21:            Fin de Si
22:        Fin de Tant que
23:        Faire_Appariements_Distants(G, L_c, L_échange)
24:        Si (le taux de contraction est suffisant) alors
25:            Sortir du TantQue
26:        Fin de Si
27:    Fin de Tant que
28: Fin de Fonction
```

Nous avons étudié et implanté deux versions différentes de la fonction de synchronisation qui vont maintenant être décrites.

2.2.1 Un premier algorithme de synchronisation

Ce premier algorithme est équivalent à celui utilisé dans ParMeTiS [45], et c'est pourquoi nous le dénommerons algorithme de synchronisation P. Il est décrit par l'algorithme 8 de la page suivante, et se décompose en trois phases :

1. une phase de diffusion des requêtes entre les différents processeurs (lignes 2 – 7) ;
2. une phase d'analyse et de validation des requêtes précédemment reçues (lignes 8 – 21) ;
3. une phase de communication des réponses relatives aux requêtes postées (lignes 22 – 29).

Les deux étapes de communication (1 et 3) correspondent à des communications point-à-point entre toutes les paires de processeurs voisins. La deuxième phase est celle qui va permettre de valider les appariements demandés par les processeurs voisins. Elle consiste en une boucle sur les processeurs voisins (ligne 8), parcourus dans un ordre aléatoire. Toutes les requêtes issues du processeur p_i sont alors traitées et l'appariement n'est permis que si le sommet local souhaité est libre ou alors réservé pour le même sommet que celui effectuant la demande d'appariement dans la requête analysée (ligne 18). Dans ce dernier cas, on définit une priorité entre les processeurs, afin que l'appariement ne soit validé qu'une seule fois. Il est à noter qu'aucune communication n'est effectuée lors de cette phase.

Le parcours aléatoire des différents processeurs voisins permet de conserver un certain niveau d'aléa dans l'appariement, l'ordre de parcours ayant une influence sur la position des sommets « libres ».

Le principal défaut de cette méthode d'analyse et de validation des requêtes est que le critère permettant l'appariement est très, voire trop, contraignant. Ce fait sera d'ailleurs plus amplement étudié dans la section 2.3.2, relative à l'analyse des algorithmes de synchronisation.

Afin de diminuer globalement le nombre de refus, l'algorithme contenu dans ParMeTiS utilise en plus une phase de coloration du graphe G, à l'aide l'algorithme de Luby [54], et effectue l'étape de synchronisation en bouclant sur les couleurs et en autorisant seulement les sommets de la couleur courante à émettre des demandes d'appariement. Ce procédé ne modifie pas l'efficacité de l'appariement mais peut permettre de diminuer le volume total de communications ; cela sera discuté en section 2.3.2.

2.2.2 Un second algorithme de synchronisation : utilisation d'une coloration

Afin d'optimiser le taux de réussite d'un appariement distant, nous avons donc introduit un autre algorithme de synchronisation. Celui-ci repose sur un coloriage de Luby du graphe de voisinage des processeurs et non du graphe G. Dans la suite du document, nous ferons référence à cette méthode sous le nom d'algorithme de synchronisation L.

Définition 28 (Graphe de voisinage des processeurs)
Le graphe de voisinage G_p des processeurs d'un graphe G distribué sur p processeurs est le graphe quotient de G en prenant comme relation d'équivalence le fait pour les sommets d'être situés sur le même processeur.

Soit $V_{\lfloor p_i}$ l'ensemble des sommets de G localisés sur le processeur p_i.

Algorithme 8 Algorithme P réalisant les appariements distants.

1: **Procédure** FAIRE_APPARIEMENTS_DISTANTS(G : Graphe distribué sur les p processeurs, L_c : Liste locale des paires de sommets dont l'appariement est sûr, $L_{\text{échange}}$: Liste locale des appariements potentiels inter-processeurs)
$L_{\text{accepte}_{p_i}}$: liste des appariements acceptés entre un sommet du processeur courant et un sommet du processeur p_i
$L_{\text{requête}_{p_i}}$: liste des appariements souhaités avec un sommet du processeur courant par un sommet du processeur p_i

Première phase de communication : émission et réception des requêtes
2: **Pour** (tous les processeurs voisins p_i) **Faire**
3: Communiquer au processeur p_i la sous-liste $L_{\text{échange}_{p_i}}$ comportant les éléments de $L_{\text{échange}}$ correspondant à un appariement avec un sommet appartenant à p_i
4: **Fin de Pour**
5: **Pour** (tous les processeurs voisins p_i) **Faire**
6: Recevoir du processeur p_i la sous-liste $L_{\text{requête}_{p_i}}$ comportant les appariements souhaités par le processeur p_i avec le processeur courant
7: **Fin de Pour**

Phase d'analyse des requêtes
8: **Pour** (p_i un processeur voisin choisi aléatoirement) **Faire**
9: $L_{\text{accepte}_{p_i}} \leftarrow \emptyset$
10: **Pour** (tous les éléments *paire* de $L_{\text{requête}_{p_i}}$) **Faire**
11: **Si** (le sommet local de *paire* est marqué « apparié ») **alors**
12: **Continuer**
13: **Sinon Si** (le sommet local de *paire* est marqué « réservé » avec le sommet distant de p_i émettant le message) **alors**
14: **Si** (le processeur p_i est prioritaire sur le processeur courant) **alors**
15: **Continuer**
16: **Fin de Si**
17: **Fin de Si**
18: $L_{\text{accepte}_{p_i}} \leftarrow \{L_{\text{accepte}_{p_i}}, \textit{paire}\}$ ▷ Dans ce cas on accepte l'appariement
19: Marquer le sommet local de *paire* comme apparié
20: **Fin de Pour**
21: **Fin de Pour**

Seconde phase de communication : émission et réception des réponses
22: **Pour** (tous les processeurs voisins p_i) **Faire**
23: Communiquer au processeur p_i la liste $L_{\text{accepte}_{p_i}}$
24: **Fin de Pour**
25: **Pour** (tous les processeurs voisins p_i) **Faire**
26: Réceptionner les paires de sommets appariés à distance
27: Marquer tous les sommets locaux appariés sur le processeur p_i comme appariés
28: Marquer les sommets dont les appariements ont échoué comme « libres »
29: **Fin de Pour**
30: **Fin de Procédure**

Dans le graphe G_p, il existe une arête entre deux processeurs p_i et p_j si et seulement si il existe au moins une arête $\{u, v\}$ dans G telle que u est placé sur p_i et v est placé sur p_j :

$$\forall (p_i, p_j) \in [\![1; p]\!]^2, \{u, v\} \in E(G_p) \iff \exists \{u, v\} \in E(G), u \in V_{|p_i}, v \in V_{|p_j} \ . \tag{2.1}$$

Ce graphe représente les dépendances entre les processeurs ; ainsi si deux processeurs ne sont pas reliés, il ne pourra pas exister d'appariements entre eux deux. Un exemple de graphe de voisinage des processeurs est visible à la figure 2.2a de la page 47.

Un coloriage appliqué au graphe G_p permet de former des ensembles indépendants de processeurs. Les processeurs d'une même couleur n'étant pas adjacents, il ne peut y avoir d'appariements de sommets entre eux. C'est cette propriété qui est utilisée dans l'algorithme 9 de la page suivante.

Celui-ci nécessite la connaissance d'un coloriage du graphe de voisinage des processeurs, dans notre cas calculé par l'algorithme de coloration de Luby. Il consiste en une boucle sur les couleurs, les processeurs de la couleur courante étant les seuls autorisés à émettre leurs requêtes d'appariement. Tous les processeurs traitent ensuite les demandes, en accordant celles qui concernent un sommet libre ou bien réservé pour une requête vers le processeur émetteur. Les résultats d'appariements sont ensuite renvoyés aux émetteurs des requêtes, puis les listes d'appariement et les marquages des sommets sont mis à jour sur ceux-ci.

Un exemple de déroulement de cet algorithme est fourni en figure 2.2b de la page 47. Le graphe de voisinage des processeurs est dans ce cas un graphe coloriable avec deux couleurs, que nous noterons 0 et 1. On constate que notre algorithme nécessite quatre phases de communication :

1. les processeurs de la couleur 0 effectuent les demandes d'appariements à leurs voisins (phase 0.1) ;
2. les processeurs de la couleur 0 reçoivent les réponses à leurs requêtes (phase 0.2) ;
3. les processeurs de la couleur 1 demandent à leur tour à effectuer des appariements (phase 1.1) ;
4. les processeurs de la couleur 1 reçoivent les réponses concernant leurs souhaits d'appariements (phase 1.2).

Cet exemple illustre bien l'existence d'une mise en série des communications, mais il faut aussi constater qu'avec plus de processeurs toujours répartis sur une chaîne, le graphe de voisinage des processeurs reste bi-coloriable et les communications s'effectuent toujours en au plus quatre phases.

Cependant, on constate que le déroulement de l'algorithme est très dépendant de la partition initiale : dans le cas où celle-ci est mauvaise, le graphe de voisinage des processeurs G_p est un graphe complet, et il n'est donc au mieux que p-coloriable. Dans ce cas, la synchronisation est séquentielle. Nous verrons dans la section 2.3.2, consacrée à l'étude de l'efficacité des algorithmes de synchronisation, que l'augmentation du nombre de couleurs peut néanmoins avoir un intérêt en facilitant les appariements distants.

2.3 Analyse de la parallélisation de l'appariement

2.3.1 Analyse quantitative de l'appariement

L'aspect itératif de l'algorithme d'appariement rend difficile à étudier sa complexité en temps ainsi que le volume de communications nécessaires, le taux d'acceptation des appariements distants étant très dépendant de la topologie locale du graphe.

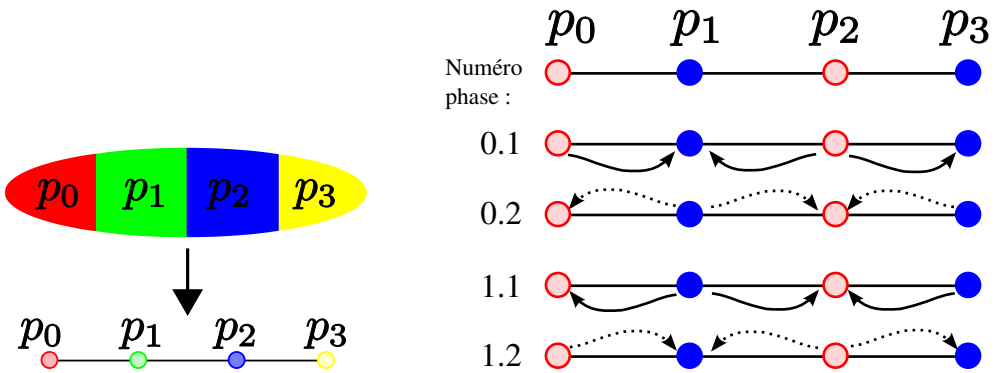
Algorithme 9 Algorithme L réalisant les appariements distants : utilisation d'une coloration

1: **Procédure** FAIRE_APPARIEMENTS_DISTANTS(G : Graphe distribué sur les p processeurs, L_c : Liste locale des paires de sommets dont l'appariement est sûr, $L_{\text{échange}}$: Liste locale des appariements potentiels inter-processeurs)

Requiert Une coloration du graphe des processeurs
$L_{\text{accepte}_{p_i}}$: liste des appariements acceptés entre un sommet du processeur courant et un sommet du processeur p_i
$L_{\text{requête}_{p_i}}$: liste des appariements souhaités avec un sommet du processeur courant par un sommet du processeur p_i

2: **Pour** (toutes les couleurs i) **Faire**
3: **Si** (le processeur local est de la couleur i) **alors**
Communication de requêtes et attente des réponses
4: **Pour** (tous les processeurs voisins p_j) **Faire**
5: Communiquer au processeur p_j la sous liste $L_{\text{échange}_{p_j}}$ comportant les éléments de $L_{\text{échange}}$ correspondant à un appariement avec un sommet appartenant à p_j
6: **Fin de Pour**
7: **Pour** (tous les processeurs voisins p_j) **Faire**
8: Recevoir du processeur p_j la sous-liste $L_{\text{accepte}_{p_j}}$
9: Marquer tous les sommets locaux de $L_{\text{accepte}_{p_j}}$ comme « appariés »
10: Marquer les sommets dont l'appariement a échoué comme « libres »
11: **Fin de Pour**

12: **Sinon** ▷ Processeur d'une autre couleur
Réception des requêtes
13: **Pour** (tous les processeurs voisins p_j de la couleur i) **Faire**
14: Recevoir du processeur p_k la sous liste $L_{\text{requête}_{p_j}}$
15: **Fin de Pour**
Analyse des requêtes
16: **Pour** (tous les processeurs voisins p_j de la couleur i) **Faire**
17: $L_{\text{accepte}_{p_j}} \leftarrow \emptyset$
18: **Pour** (tous les éléments *paire* de $L_{\text{requête}_{p_j}}$) **Faire**
19: **Si** (le sommet local de *paire* est marqué « apparié ») **alors**
20: **Continuer**
21: **Sinon Si** (le sommet local de *paire* n'est pas marqué « réservé » avec un sommet distant de p_j) **alors**
22: **Continuer**
23: **Fin de Si**
24: $L_{\text{accepte}_j} \leftarrow \{L_{\text{accepte}_j}, paire\}$ ▷ Dans ce cas on accepte l'appariement
25: Marquer le sommet local de *paire* comme apparié
26: **Fin de Pour**
27: **Fin de Pour**
Émission des réponses
28: **Pour** (tous les processeurs voisins p_j de la couleur i) **Faire**
29: Communiquer au processeur p_j la liste $L_{\text{accepte}_{p_j}}$
30: **Fin de Pour**
31: **Fin de Si**
32: **Fin de Pour**
33: **Fin de Procédure**



(a) Un graphe distribué et le graphe de voisinage des processeurs associé.

(b) Déroulement du second algorithme : les différentes étapes de communication. La couleur 0 est représentée en clair, la 1 en foncé. Les flèches en traits pleins représentent les requêtes, celles en pointillés les réponses.

FIG. 2.2: Exemple de déroulement du second algorithme de synchronisation.

Nous allons seulement étudier ici la première itération, qui est la plus coûteuse, puisque le nombre de sommets à analyser est maximal, aucun sommet n'ayant encore été traité. Le nombre de souhaits d'appariements distants est au plus n, dans le cas où tous les sommets possèdent au moins un voisin non local et choisissent celui-ci. Ce nombre est majoré par la taille des interfaces.

Dans l'algorithme de synchronisation P, toutes les requêtes sont émises dès le début et les réponses relatives associées à la fin et en même nombre. Avec le second algorithme présenté, L, les requêtes sont émises par couleur, en fonction des souhaits courants. Ceux-ci tiennent donc compte des réponses obtenues lors des échanges avec les couleurs précédentes, et sont donc moins nombreux que les souhaits initiaux. Le volume des communications est donc plus faible pour ce deuxième algorithme que pour le premier. Cependant, les communications effectuées par ce second algorithme sont plus difficiles à recouvrir et sont plus nombreuses, ce qui peut s'avérer coûteux sur un réseau dont la latence est élevée.

Si l'on note $inter(G)$ l'interface de G, c'est-à-dire l'ensemble des sommets qui sont adjacents à des sommets non locaux, on obtient que le volume total des communications nécessaires à la première itération est majoré par $\mathcal{O}(|inter(G)|)$. Ainsi, la distribution initiale de G a un impact sur le volume et donc la durée des communications, ce qui a conduit certains développeurs, comme ceux de PARMETIS notamment, à effectuer une redistribution des données avant d'effectuer ces calculs. Le choix d'une telle redistribution dans PT-SCOTCH sera discuté plus loin.

Nous allons maintenant nous intéresser à l'efficacité des deux algorithmes de synchronisation, et par conséquent aux nombres d'itérations qui seront nécessaires pour mener à bien l'appariement.

2.3.2 Comparaison de l'efficacité de l'appariement

Afin d'étudier la qualité des deux algorithmes de synchronisation, il nous faut évaluer leurs capacités à permettre et à réaliser des appariements distants.

Nous allons calculer la probabilité de succès d'un appariement distant lorsque celui-ci est demandé. Cette probabilité sera évaluée en moyenne. Nous nous plaçons dans le cas d'un graphe

aléatoirement distribué, c'est-à-dire que les sommets sont répartis au hasard, avec une probabilité uniforme, sur les différents processeurs.

Soient deux sommets u et v, u étant sur le processeur p_i et v sur le processeur p_j ; on note δ_u et δ_v leurs degrés respectifs et on notre $\delta_{u|p_l}$ le nombre de voisins de u appartenant au processeur p_l. Nous souhaitons calculer la probabilité de succès d'appariement entre u et un sommet v non local. On note $\mathcal{S}_{G_p}(p_l)$ l'ensemble des processeurs voisins[7] de p_l et $\bar{p}$ le nombre moyen de voisins pour un processeur.

La figure 2.3 illustre un exemple de voisinage du sommet u. Dans ce cas, on observe la présence de trois processeurs voisins, chacun ayant deux connexions avec u, donc dans ce cas $\delta_{u|p_l} = 2$, quelle que soit la valeur de p_l. Nous allons continuer notre raisonnement en nous plaçant dans une situation similaire à celle représentée, si ce n'est que le nombre de voisins n'est pas fixé, tout comme le degré des sommets.

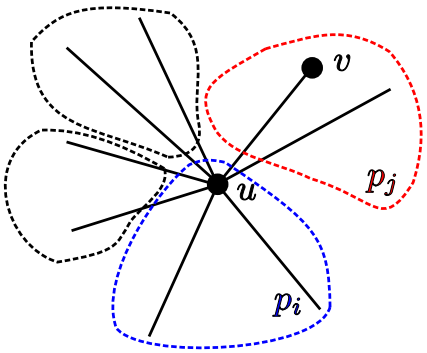


FIG. 2.3: Un voisinage possible du sommet u.

2.3.2.1 Calculs préliminaires

Nous allons évaluer pour chacun des algorithmes la probabilité $P_{\text{algo }\nu}$ qu'un sommet u du processeur p_i soit apparié avec un processeur distant lors de la première itération de l'algorithme ν. Nous noterons $P(u \to v)$ la probabilité que le sommet u fasse une requête pour s'apparier avec le sommet v.

La valeur de $P(u \to v)$ correspond en fait à la conjonction des événements suivants :

- le sommet u n'a pas été apparié lorsqu'il est parcouru par le processus d'établissement des demandes ;
- le sommet u fait le choix de s'apparier avec le sommet v.

Grâce au parcours aléatoire des sommets lors de la phase de décision des appariements souhaités pour les sommets, le sommet u est, en moyenne, parcouru lorsque la moitié des sommets locaux l'ont été. Comme la distribution du graphe est aléatoire et uniforme sur l'ensemble des processeurs, on peut dire qu'en moyenne, lorsque u est parcouru par le processus de décision, la moitié de ses voisins l'ont été également. La probabilité qu'un voisin de u ne choisisse pas u comme partenaire d'appariement est $\dfrac{\delta_w - 1}{\delta_w}$, soit en moyenne $\dfrac{\bar{\delta} - 1}{\bar{\delta}}$. La probabilité que u soit encore « libre » lorsque son tour arrive dans l'algorithme de décision est donc, en moyenne, égale à la fonction $f : (\bar{\delta}, \bar{p}) \mapsto \left(\dfrac{\bar{\delta} - 1}{\bar{\delta}}\right)^{\frac{\bar{\delta}}{2\cdot(\bar{p}+1)}}$. Pour u, le choix du partenaire se fait parmi les voisins

[7] En fait, $\mathcal{S}_{G_p}(p_l)$ représente la sphère de centre p_l et de rayon un dans le graphe G_p de voisinage des processeurs.

non encore visités, qui sont, en moyenne, au nombre de $\bar{\delta} - \frac{\delta_{v|j}}{2}$.

On obtient donc l'expression suivante :

$$\begin{aligned} P(u \to v) &= P\left(u \in \mathcal{L}_{\frac{1}{2}}\right) \cdot \frac{1}{\bar{\delta} - \frac{\delta_{v|j}}{2}} \ , \\ &= \frac{1}{\bar{\delta} - \frac{\bar{\delta}}{2 \cdot \bar{p}+1}} \cdot f(\bar{\delta}, \bar{p}) \ , \\ &= \frac{2 \cdot (\bar{p}+1)}{\bar{\delta} \cdot (2\bar{p}+1)} \cdot f(\bar{\delta}, \bar{p}) \ . \end{aligned} \tag{2.2}$$

Nous allons maintenant évaluer la probabilité que le sommet v soit dans l'état « libre » lors de l'analyse par le k-ième processeur voisin. Soit $\mathcal{L}_k$ l'ensemble des sommets « libres » pendant l'analyse par le k-ième processeur voisin. Nous pouvons dire que des éléments sont ajoutés à cet ensemble seulement lors de la phase de synchronisation, nous savons donc que la probabilité $P(v \in \mathcal{L}_1)$ que v soit libre lors de la tentative de synchronisation avec le premier processeur voisin est égale à zéro.

Nous allons maintenant calculer la probabilité $P(v \in \mathcal{L}_k)$ que possède le sommet v d'être « libre » après l'analyse par $k-1$ processeurs.

Le sommet v est « libre » pour le k-ième processeur voisin si et seulement si les conditions suivantes sont réunies :

- v n'est pas apparié localement, alors qu'il a une probabilité $P_l(v)$ de l'être ;
- v n'a pas été apparié avec un sommet distant lors des $k-1$ précédentes études, alors qu'il a une probabilité $P_d(v, k-1)$ de l'être ;
- v ne souhaite pas s'apparier avec un sommet du processeur p_λ, avec $\lambda > k$.

On obtient donc :

$$P(v \in \mathcal{L}_k) = 1 - P_l(v) - P_d(v, k-1) - (\bar{p} - k) \cdot P\left(v \to v | (v \in V_{p_\lambda}, \lambda > k)\right) \ .$$

Les sommets étant répartis aléatoirement et uniformément sur les processeurs, on a :

$$P(v \in \mathcal{L}_k) = 1 - P_l(v) - P_d(v, k-1) - (\bar{p} - k) \cdot \frac{\bar{\delta}}{\bar{p}+1} \cdot P(v \to u) \ . \tag{2.3}$$

La probabilité que v soit apparié localement est égale à :

$$\begin{aligned} P_l(u) &= \frac{1}{2} \cdot \sum_{\substack{v \in V_i \\ \{u,v\} \in E}} P(u \to v) + \frac{1}{2} \cdot \sum_{\substack{v \in V_i \\ \{u,v\} \in E}} P(v \to u) \ , \\ P_l(u) &= \frac{\bar{\delta}}{\bar{p}+1} \cdot P(u \to v) \ . \end{aligned} \tag{2.4}$$

En remplaçant $P_l(v)$ dans l'équation (2.3) par le résultat précédent, on obtient :

$$P(v \in \mathcal{L}_k) = 1 - P_d(v, l-1) - (p - k + 1) \cdot \frac{\bar{\delta}}{\bar{p}+1} \cdot P(u \to v) \ . \tag{2.5}$$

Il peut être intéressant de noter que la fonction $k \to P(v \in \mathcal{L}_k)$ est croissante.

Maintenant que ces probabilités sont exprimées, nous allons nous intéresser, pour chacun des deux algorithmes, à évaluer la probabilité $P_d(u, \bar{p})$ pour un sommet u d'être apparié avec un sommet distant après l'analyse des $\bar{p}$ processeurs voisins.

2.3.2.2 Probabilité des appariements distants avec l'algorithme P

Nous allons étudier la probabilité $P_d(u,k)$ pour un sommet u du processeur p_i d'être apparié avec un sommet k après l'analyse du k-ième processeur voisin. Nous notons $\nu(j)$ le nombre de processeurs voisins de p_i dont les réponses aux requêtes d'appariement sont déjà parvenues et ont déjà été traitées par p_i avant l'analyse du j-ième processeur. Ce nombre nous est utile car notre implantation introduit une petite variante par rapport à l'algorithme P présenté en page 44. Effectivement, afin de recouvrir les communications, les messages contenant des réponses aux requêtes d'appariements sont traités aussitôt qu'ils peuvent l'être, et donc potentiellement avant que les demandes de tous les voisins nous soient parvenues ou aient été étudiées. Le même phénomène se produit dans l'algorithme original de ParMeTiS puisque celui-ci parcourt les requêtes selon la couleur des sommets, et donc met à jour l'état des sommets avant de passer à la couleur suivante.

On a donc :

$$P_d(u,k) = \sum_{\substack{p_j \in \mathcal{S}_{G_p}(p_i) \\ j \in [\![1;k]\!]}} \sum_{\substack{v \in V_{|p_j} \\ \{u,v\} \in E}} (P(u \to v) \cdot (P(v \to u) + P(v \in \mathcal{L}_1)) + P(v \to u) \cdot P(u \in \mathcal{L}_{\nu(j)})) \ .$$

Comme il y a en moyenne $\frac{\bar{\delta}}{\bar{p}+1}$ sommets de p_j voisins de u, on obtient :

$$P_d(u,k) = k \cdot \frac{\bar{\delta}}{\bar{p}+1} \cdot P^2(u \to v) + \frac{\bar{\delta}}{\bar{p}+1} \cdot P(u \to v) \cdot \sum_{\substack{p_j \in \mathcal{S}_{G_p}(p_i) \\ j \in [\![1;k]\!]}} P(u \in \mathcal{L}_{\nu(j)}) \ . \tag{2.6}$$

Nous allons proposer un encadrement pour la probabilité $P_d(u,k)$ en nous plaçant dans les deux cas extrêmes suivants : soit aucun des voisins n'a encore répondu, soit $\nu(j) = j$ voisins ont répondu. Dans le premier cas, sachant que $P(u \in \mathcal{L}_j) = 0$, on obtient :

$$P_d(u,k) \geq k \cdot \frac{\bar{\delta}}{\bar{p}+1} \cdot P^2(u \to v) \ , \tag{2.7}$$

et en vertu de l'équation (2.2), on a :

$$P_d(u,k) \geq \frac{4k \cdot (\bar{p}+1)}{\bar{\delta} \cdot (2\bar{p}+1)^2} \cdot f^2(\bar{\delta}, \bar{p}) \ .$$

Une majoration nous est donnée par le cas $\nu(j) = j$:

$$\begin{aligned}
P_d(u,k) &\leq k \cdot \frac{\bar{\delta}}{\bar{p}+1} \cdot P^2(u \to v) \\
&\quad + \frac{\bar{\delta}}{\bar{p}+1} \cdot (P(u \to v) \cdot \sum_{\substack{p_j \in \mathcal{S}_{G_p}(p_i) \\ j \in [\![1;k]\!]}} \left(1 - P_d(u, j-1) - (p-j+2) \cdot \frac{\bar{\delta}}{\bar{p}+1} \cdot P(u \to v)\right) , \\
P_d(u,k) &\leq \frac{k \cdot \bar{\delta}}{\bar{p}+1} \cdot \left(1 - \bar{\delta} + \frac{\bar{\delta} \cdot (k+1)}{2(\bar{p}+1)}\right) \cdot P^2(u \to v) + \frac{k \cdot \bar{\delta}}{\bar{p}+1} \cdot P(u \to v) \\
&\quad - \frac{\bar{\delta}}{\bar{p}+1} \cdot P(u \to v) \cdot \sum_{\substack{p_j \in \mathcal{S}_{G_p}(p_i) \\ j \in [\![1;k]\!]}} P_d(u, j-1) \ .
\end{aligned} \tag{2.8}$$

Il en découle l'encadrement suivant :

$$\begin{aligned}
\bar{p} \cdot \frac{\bar{\delta}}{\bar{p}+1} \cdot P^2(u \to v) \leq P_{\text{algo 1}} \leq \frac{\bar{p} \cdot \bar{\delta}}{\bar{p}+1} \cdot \left(1 - \frac{\bar{\delta}}{2}\right) \cdot P^2(u \to v) + \\
\frac{\bar{p} \cdot \bar{\delta}}{\bar{p}+1} \cdot P(u \to v) - \frac{\bar{\delta}}{\bar{p}+1} \cdot P(u \to v) \cdot \sum_{\substack{p_j \in \mathcal{S}_{G_p}(p_i) \\ j \in [\![1;\bar{p}]\!]}} P_d(u, j-1) \ .
\end{aligned} \tag{2.9}$$

Le graphique 2.4 montre l'encadrement que nous avons obtenu pour la probabilité d'appariement avec l'algorithme P et les valeurs que nous avons mesurées en instrumentant notre implantation. Afin de nous rapprocher le plus possible du cadre théorique tel que nous l'avons décrit, nous avons procédé à ces tests sur des graphes permutés aléatoirement et distribués sur les processeurs par intervalles selon cette permutation des sommets.

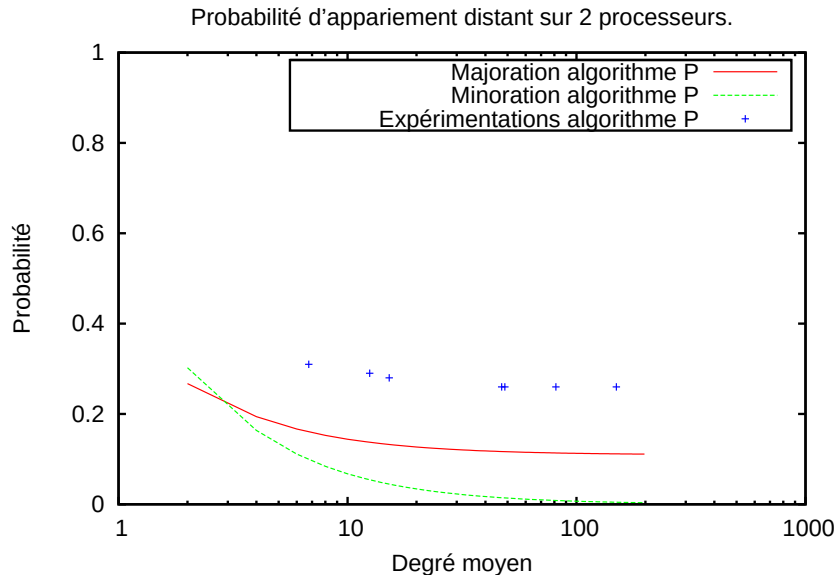
La première expérience, sur deux processeurs, visible en figure 2.4a, montre que notre modèle théorique est dans ce cas un peu pessimiste. Le fait que les probabilités soient supérieures à nos prévisions peut être expliqué par le fait que, le nombre de processeurs étant petit, notre estimation se basant sur un passage à la moyenne est fausse.

L'expérience suivante, figure 2.4b, corrobore pour sa part nos hypothèses et notre modèle théorique. On constate que la diminution de la probabilité d'un appariement distant pour le sommet u se fait dans les mêmes proportions que celle prévue par la théorie. Cette diminution du taux d'appariement distant est une caractéristique importante de l'algorithme P, qui le rend assez difficilement exploitable sur des graphes de degré moyen élevé.

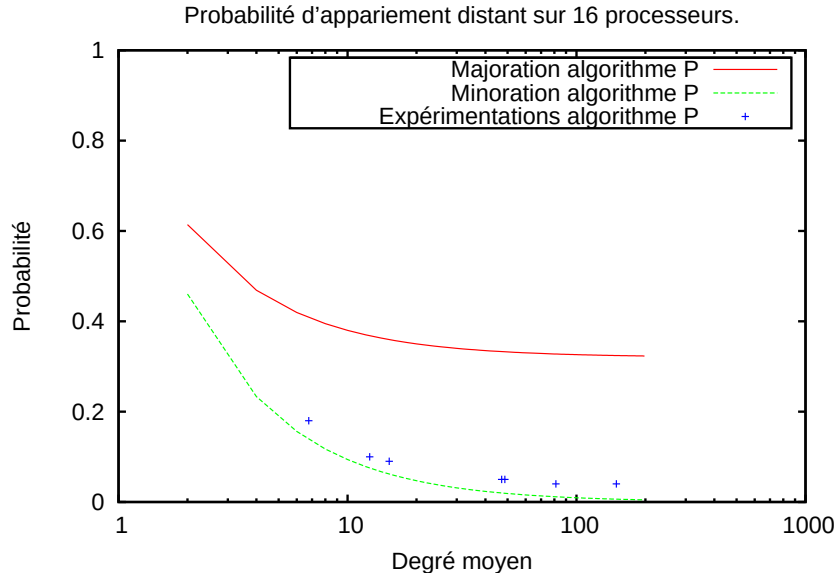
De plus, l'algorithme P est sensible à l'augmentation du nombre de processeurs mais pas de la façon escomptée dans le cadre d'un appariement isotrope. Nous allons voir ce qu'il en est pour l'algorithme L.

2.3.2.3 Probabilité des appariements distants avec l'algorithme L

Le principe du raisonnement est identique à celui suivi précédemment pour l'étude de l'algorithme P. Les différences dans les résultats sont dues au fait que l'algorithme L est plus permissif pour les appariements distants : il suffit que le sommet v demandé par le sommet u souhaite s'associer avec un sommet de p_i pour que l'appariement soit possible.



(a) Résultats obtenus avec deux processeurs. Les résultats expérimentaux sont meilleurs que les résultats théoriques car le passage à la moyenne n'est justifié que lorsque le nombre de processeurs est suffisamment grand, ce qui n'est pas le cas ici.



(b) Résultats obtenus avec 16 processeurs, ce qui est suffisant pour que l'hypothèse concernant l'utilisation, dans le modèle théorique, de la moyenne du nombre de processeurs voisins visités soit réaliste.

FIG. 2.4: Estimation de la probabilité pour un sommet u d'obtenir un appariement distant. Le graphe est distribué sur 2 et 16 processeurs et l'algorithme P est utilisé pour la synchronisation.

Cette caractéristique rend cependant les appariements distants plus difficiles à étudier que pour l'algorithme P, car elle fait perdre la symétrie du problème que nous avons exploité auparavant : « u demande v et v demande u » devient maintenant « u demande v et v demande un sommet sur le même processeur p_i que u ». Notre évaluation peut donc conduire à compter en double certains appariements et le passage à la moyenne pourra quelque peu fausser notre estimation. Néanmoins, sur un assez grand nombre de processeurs, et avec un degré moyen important, notre estimation devrait pouvoir se révéler assez fiable.

Comme la fonction $k \mapsto P(v \in \mathcal{L}_k)$ est croissante, nous nous placerons dans le cas où le nombre de processeurs voisins analysés et dont les réponses sont parvenues, à la j-ième étape, est $\nu(j) = j$. Cette hypothèse est par exemple vérifiée dans le cas où le nombre de couleurs utilisées pour la coloration du graphe de voisinage des processeurs est $\bar{p}$.

$$P_d(u,k) = \sum_{\substack{p_j \in \mathcal{S}_{G_p}(p_i) \\ j \in [\![1;k]\!]}} \sum_{\substack{v \in V_{|p_j} \\ \{u,v\} \in E}} \left(P(u \to v) \cdot \left(\sum_{\substack{w \in V_{|p_i} \\ \{v,w\} \in E}} (P(v \to w)) + P(v \in \mathcal{L}_j) \right) + P(v \to u) \cdot P(u \in \mathcal{L}_j) \right) ,$$

soit, en exploitant la symétrie du problème et la répartition uniforme des sommets sur les processeurs :

$$P_d(u,k) = \sum_{\substack{p_j \in \mathcal{S}_{G_p}(p_i) \\ j \in [\![1;k]\!]}} \sum_{\substack{v \in V_{|p_j} \\ \{u,v\} \in E}} \left(\frac{\bar{\delta}}{\bar{p}+1} \cdot P^2(u \to v) + 2 \cdot P(u \to v) \cdot P(v \in \mathcal{L}_j) \right) . \tag{2.10}$$

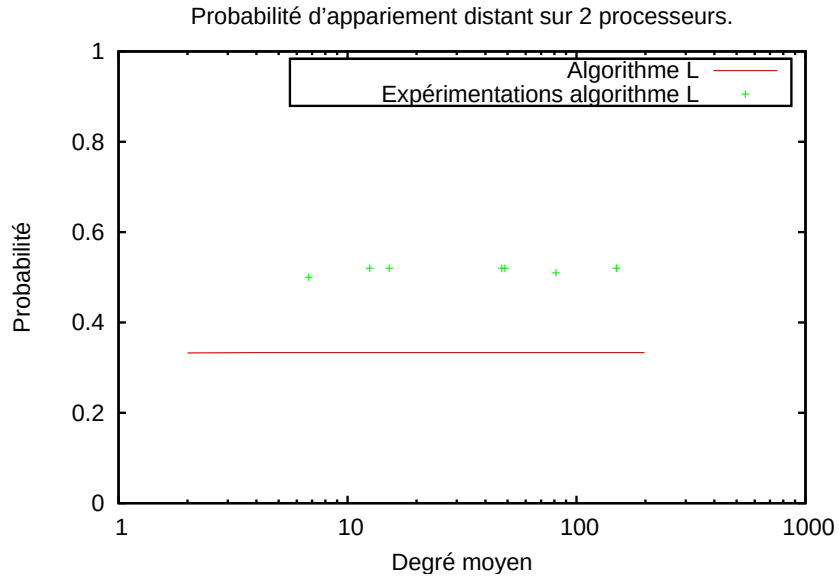
Par le même raisonnement que celui utilisé pour l'étude de l'algorithme P, nous obtenons la majoration suivante :

$$P_{\text{algo 2}} = \frac{p \cdot \bar{\delta}}{\bar{p}+1} \cdot \left(\frac{\bar{\delta}}{\bar{p}+1} - \bar{\delta} \right) \cdot P^2(u \to v) \quad + \frac{2\bar{p} \cdot \bar{\delta}}{\bar{p}+1} \cdot P(u \to v) - \frac{2 \cdot \bar{\delta}}{\bar{p}+1} \cdot P(u \to v) \cdot \sum_{\substack{p_j \in \mathcal{S}_{G_p}(p_i) \\ j \in [\![1;p]\!]}} P_d(u, j-1) . \tag{2.11}$$

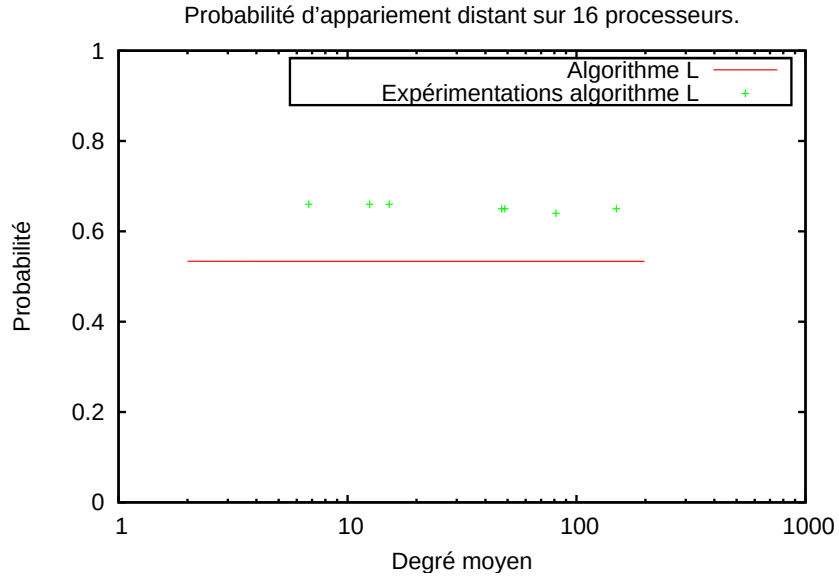
La figure 2.5 montre la courbe représentative de la fonction de majoration de la probabilité d'appariement, en fonction du degré moyen du graphe. Des expérimentations conduites avec le même protocole que précédemment permettent de confronter notre étude théorique à la réalité.

Notre étude théorique semble quelque peu minimiser la probabilité réelle d'existence d'appariements distants mais il est intéressant de remarquer que le comportement des résultats expérimentaux est identique à celui prédit : la probabilité pour un sommet u d'être apparié à distance ne dépend pas du degré moyen du graphe, contrairement à l'approche précédente.

Nous pouvons aussi constater que, contrairement à l'algorithme P, la probabilité d'appariement distant augmente lorsque le nombre de processeurs utilisés augmente.



(a) Probabilité d'appariements distants sur un graphe uniformément réparti sur deux processeurs.



(b) Probabilité d'appariements distants sur un graphe uniformément réparti sur 16 processeurs.

FIG. 2.5: Estimation de la probabilité pour un sommet u d'obtenir un appariement distant. Le graphe est distribué sur 2 et 16 processeurs et l'algorithme L est utilisé pour la synchronisation.

2.3.2.4 Bilan théorique et pratique sur la fréquence des appariements à la première itération des algorithmes de synchronisation

Nous observons que l'algorithme L est très supérieur à l'algorithme P en terme de nombre d'appariements distants réalisés à la première itération, et ce d'autant plus que le degré moyen du graphe est grand et que le nombre de processeurs utilisés est important.

Cependant, quand le nombre de couleurs utilisées dans la coloration du graphe de voisinage des processeurs devient grand, il peut être intéressant d'utiliser l'algorithme P, qui procure alors davantage de parallélisme.

Il reste néanmoins dommage qu'avec l'algorithme P les propriétés topologiques du graphe, comme son degré moyen, aient un tel impact sur les probabilités d'appariements à distance et donc sur le temps d'exécution, ainsi que sur la qualité de la contraction. En effet, parce qu'il modifie la probabilité d'avoir des voisins distants, le biais présent dans le processus d'appariement lui fait perdre un peu de ses capacités de conservation de la topologie d'un niveau de contraction au suivant.

2.3.3 Optimisation de l'appariement : utilisation de probabilités de réussite

Les deux algorithmes présentés précédemment convergent, c'est-à-dire que le nombre de sommets « libres » décroît lorsque le nombre d'itérations de l'algorithme de synchronisation augmente. Néanmoins, leur rendement par rapport au volume de communications effectuées est assez faible, la probabilité d'appariement lors des premières itérations étant petite.

L'idée que nous proposons consiste à essayer d'optimiser le volume de données échangées en essayant d'anticiper le fait que certains appariements n'ont que peu de chance d'être autorisés. Lors du parcours des sommets, nous ne cherchons plus systématiquement un voisin auquel nous apparier mais nous pouvons dorénavant choisir de laisser ce sommet « libre », en passant directement au sommet suivant dans la liste des sommets à traiter. Nous pouvons ainsi augmenter la disponibilité des sommets pour l'appariement distant.

En pratique, nous n'étudions les sommets qu'en fonction de leur probabilité d'être appariés localement. Celle-ci est égale, pour le sommet u du processeur p_i à $\frac{\delta_{u|p_i}}{\delta_u}$. A priori, cette manière de faire ne pénalise pas beaucoup les appariements locaux, car dans les cas réels d'utilisation, le graphe est rarement distribué de manière totalement aléatoire et donc les valeurs de $\delta_{u|p_i}$ et δ_u sont très voisines pour la plupart des sommets.

Nous allons étudier les conséquences de cette sélection dans un cas simple où l'arête $\{u, v\}$ est la seule arête reliant des sommets situés sur des processeurs différents. Cette situation est illustrée par la figure 2.6.

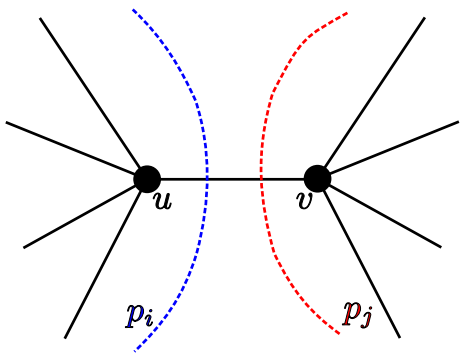


FIG. 2.6: Un cas particulier de voisinage $u - v$.

Nous allons d'abord calculer la probabilité $P_{ss}(u \leftrightarrow v)$ de réaliser l'appariement entre u et v sans sélection. $P_{ss}(u \rightarrow v)$ désignera, comme précédemment, la probabilité que u choisisse v pour s'apparier. Nous noterons $\mathcal{L}$ l'ensemble des sommets libres. Nous nous plaçons dans le cadre de la première méthode de synchronisation. Avec les notations définies ci-dessus, on a :

$$P_{ss}(u \leftrightarrow v) = P_{ss}(u \rightarrow v) \cdot (P_{ss}(v \rightarrow u) + P_{ss}(v \in \mathcal{L})) + P_{ss}(v \rightarrow u) \cdot P_{ss}(u \in \mathcal{L}) \ . \tag{2.12}$$

Maintenant, intéressons nous aux conséquences de l'utilisation d'une sélection des sommets lors du parcours des sommets à apparier. Nous reprenons les mêmes notations que précédemment, en remplaçant les indices ss (sans sélection) par des indices as (avec sélection).

$$\begin{aligned}
P_{as}(u \leftrightarrow v) &= P_{as}(u \rightarrow v) \cdot (P_{as}(v \rightarrow u) + P_{as}(v \text{ «libre»})) \\
&\quad + P_{as}(v \rightarrow u) \cdot P_{as}(u \in \mathcal{L}) \\
&= \frac{\delta_u - 1}{\delta_u} \cdot P_{ss}(u \rightarrow v) \cdot (\frac{\delta_v - 1}{\delta_v} \cdot P_{ss}(v \rightarrow u) + P_{ss}(v \text{ «libre»}) + \frac{1}{\delta_v}) \\
&\quad + \frac{\delta_v - 1}{\delta_v} \cdot P_{ss}(v \rightarrow u) \cdot (P_{ss}(u \text{ «libre»}) + \frac{1}{\delta_u})
\end{aligned}$$

En se plaçant dans le cas où $\delta_u = \delta_v = \delta$, et comme $\frac{\delta-1}{\delta} = 1 - \frac{1}{\delta}$, on obtient :

$$\begin{aligned}
P_{as}(u \leftrightarrow v) &= (1 - \frac{1}{\delta}) \cdot (P_{ss}(u \rightarrow v) \cdot (P_{ss}(v \rightarrow u) + P_{ss}(v \in \mathcal{L})) + P_{ss}(v \rightarrow u) \cdot P_{ss}(u \in \mathcal{L})) \\
&\quad - \frac{1}{\delta^2} \cdot (P_{ss}(u \rightarrow v) + P_{ss}(v \rightarrow u)) + \frac{1}{\delta} \cdot (P_{ss}(u \rightarrow v) + P_{ss}(v \rightarrow u)) \\
P_{as}(u \leftrightarrow v) &= (1 - \frac{1}{\delta}) \cdot P_{as}(u \leftrightarrow v) - \frac{1}{\delta^2} \cdot (P_{ss}(u \rightarrow v) + P_{ss}(v \rightarrow u)) \\
&\quad + \frac{1}{\delta} \cdot (P_{ss}(u \rightarrow v) + P_{ss}(v \rightarrow u)) \ .
\end{aligned} \tag{2.13}$$

La différence due à l'ajout de l'étape de sélection est donc :

$$P_{as}(u \leftrightarrow v) - P_{ss}(u \leftrightarrow v) = \frac{1}{\delta} \cdot (P_{ss}(u \rightarrow v) + P_{ss}(v \rightarrow u)) - P_{ss}(u \leftrightarrow v) - \frac{1}{\delta^2} \cdot P_{ss}(u \rightarrow v) \ . \tag{2.14}$$

Comme $P_{ss}(u \leftrightarrow v)$ est petit devant la somme $P_{ss}(u \rightarrow v) + P_{ss}(v \rightarrow u)$, cette différence est positive. La sélection permet donc d'augmenter la probabilité d'un appariement distant dans ce cas. Nous constaterons en section 2.5.2.2 de la page 64 que cela se vérifie en pratique sur les graphes réellement utilisés.

En rendant nos algorithmes de synchronisation plus efficaces, nous pouvons espérer nous dispenser d'un coûteux pré-calcul de redistribution du graphe, comme cela peut être fait dans d'autres outils de partitionnement parallèle multi-niveaux tels que PARMETIS.

2.4 La construction du graphe contracté

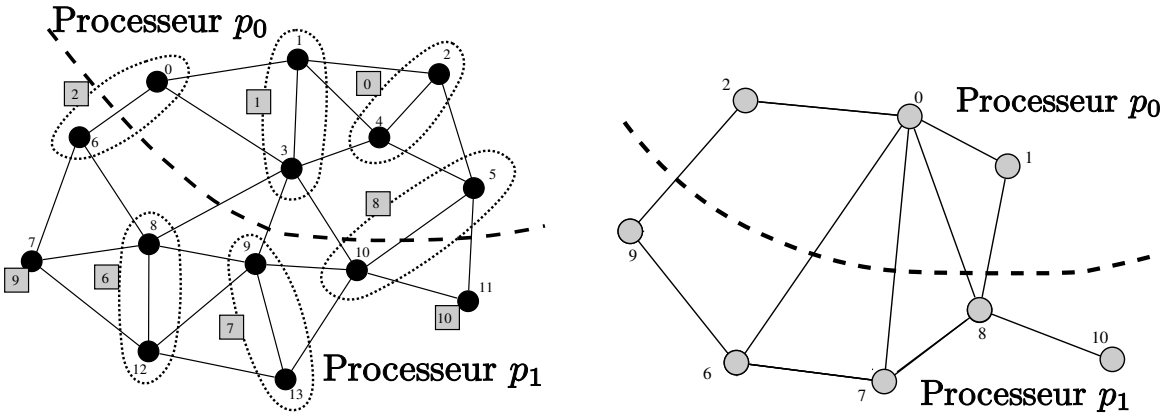
À l'issue de la phase d'appariement présentée précédemment, chaque processeur possède une liste de couples de sommets à fusionner, afin d'obtenir le graphe contracté. Si cette étape est très simple à réaliser en séquentiel, la version parallèle nécessite plus d'attention dans la mesure où interviennent plusieurs types d'échanges de données entre les processeurs :

1. il est nécessaire de connaître le nouvel identifiant de chacun des sommets voisins fusionnés, qu'ils soient distants ou locaux ;
2. lors d'un appariement entre deux sommets situés sur des processeurs distincts, seul l'un des deux processeurs possédera le sommet fusionné nouvellement formé. Il est donc nécessaire à ce processeur de connaître les informations relatives au sommet distant : liste des voisins, poids des arêtes, *etc.*

Nous allons maintenant voir comment ces différentes étapes sont associées entre elles.

2.4.1 Méthodologie de la construction du graphe contracté

La figure 2.7 montre la contraction d'un graphe distribué sur deux processeurs. Nous allons observer quelles sont les étapes nécessaires à l'obtention du graphe contracté.



(a) Graphe original distribué sur les processeurs p_0 et p_1. Les sommets au dessus de la ligne en tirets, numérotés de 0 à 5, appartiennent à p_0, ceux au dessous, numérotés de 6 à 11, à p_1. Les appariements choisis sont indiqué par les pointillés. La nouvelle numérotation du graphe est indiquée par les chiffres encadrés.

(b) Graphe original contracté sur les processeurs p_0 et p_1. Les sommets au dessus de la ligne en tirets, numérotés de 0 à 2, appartiennent à p_0, ceux au dessous, numérotés de 6 à 10, à p_1. La numérotation est donc à trous.

FIG. 2.7: Exemple de contraction d'un graphe distribué sur deux processeurs p_0 et p_1.

2.4.1.1 Appariements des sommets

La première étape est la phase d'appariement, qui permet de définir les regroupements des sommets, mais aussi à quel processeur un sommet contracté sera associé. C'est aussi lors de cette étape que le choix des nouveaux numéros de sommets va être effectué.

Dans PT-SCOTCH, pour des raisons d'équilibre des communications, qui seront explicitées à la section 2.4.1.3, les appariements des sommets locaux sont numérotés en premier, suivis des appariements non locaux, pour finir avec les sommets qui n'ont pas été appariés. On obtient alors les informations visibles sur la figure 2.7a.

La structure de graphe distribué utilisée dans PT-SCOTCH permettant l'utilisation d'une numérotation à trous, la nouvelle numérotation sera obtenue en additionnant à la position du couple dans la liste des couples, le décalage initial du processeur. Il est toutefois possible de recalculer ce décalage en comptant pour chaque processeur le nombre d'éléments de sa liste de

couples et en le diffusant à tous les autres processeurs. Dans ce cas, on obtient une numérotation sans trous mais au prix d'une communication globale supplémentaire de type somme préfixe.

Les deux prochaines étapes vont consister en la diffusion de ces informations dans le graphe, afin de pouvoir construire le graphe contracté.

2.4.1.2 Diffusion des nouveaux identifiants

La première partie de la construction consiste en la diffusion des nouveaux identifiants.

Le nouvel identifiant du sommet est le numéro du sommet qui lui correspond, c'est-à-dire fusionné ou conservé, dans le graphe contracté. Ainsi, en observant la figure 2.7a, il est visible que le processeur p_1 doit connaître la nouvelle numérotation du sommet 3, afin que les arêtes qui le reliaient aux sommets 8 et 9 soient conservées dans le graphe contracté. p_1 doit aussi connaître le numéro du sommet qui sera formé par l'appariement de 6 avec son sommet distant, de même pour p_0 et son sommet 5. Dans ce dernier cas, l'identifiant du sommet est déjà connu car il a été fixé lors de l'autorisation de l'appariement.

En fait, les communications nécessaires à cette phase de diffusion sont seulement réalisées selon les arêtes qui séparent les sommets de processeurs différents. Sur la figure 2.7a cela correspond aux arêtes coupées par la ligne en tirets.

2.4.1.3 Échange des sommets entre processeurs

La construction du graphe contracté nécessite non seulement de renuméroter les sommets, mais surtout de reconstruire les arêtes. Cette construction est réalisée naturellement en respectant le changement du numéro des sommets. On peut en revanche obtenir des arêtes multiples, et c'est pour cela qu'il est nécessaire de fusionner les arêtes qui correspondent aux mêmes identifiants.

On constate qu'il est nécessaire de connaître le voisinage des sommets impliqués lors de la création d'un sommet du graphe contracté et des arêtes qui lui sont associées. Par exemple, si l'on se place dans le cas de la figure 2.7a, le nouveau sommet 0 a besoin de connaître les voisins des sommets 0 et 6 du graphe fin. Avec la version distribuée de la construction, la difficulté survient donc lors des appariements entre sommets se situant sur des processeurs différents. Dans l'exemple précédent, il est nécessaire de transférer la connaissance du voisinage du sommet 6 au processeur p_0.

Si u et v sont deux sommets appartenant respectivement aux processeurs p_i et p_j, et sont appariés, le sommet formé sur le graphe contracté ne sera localisé pour sa part que sur l'un des deux processeurs : soit p_i, soit p_j. Un processeur va donc perdre des informations, l'autre augmentant dans le même temps sa quantité de données en recevant les informations de voisinage.

Si l'on regarde la figure 2.7a, on constate que le sommet 6 va migrer de p_1 vers p_0, de même que le sommet 5 dans l'autre sens, pour former les sommets contractés 0 et 8.

Cependant, même si la répartition du graphe initial est bien équilibrée, un déséquilibre peut toujours survenir pour le graphe contracté, la migration des informations associées aux sommets pouvant effectivement ne pas se produire de manière uniforme sur l'ensemble des processeurs.

C'est pour cela que nous effectuons une phase de sélection des migrations avant de contracter réellement le graphe. Actuellement, notre critère consiste à égaliser le nombre de sommets échangés entre chaque paire de processeurs (c'est-à-dire que si p_i reçoit l sommets de p_j, alors p_j doit aussi recevoir l sommets de p_i). La plupart des graphes ne présentant pas de très grandes variations du degré des sommets, ce critère permet aussi de limiter le volume de communication. Néanmoins, il reste toujours possible d'utiliser des critères de sélection plus performants en vue

des calculs futurs, comme par exemple de travailler directement sur le volume des communications ou même de privilégier la migration des sommets qui ont le plus de voisins sur le processeur distant. Dans ce dernier cas, on optimise la localité des données pour les prochaines itérations de la contraction, ce qui peut permettre d'avoir un appariement mieux localisé, et donc a priori mieux équilibré et plus rapide.

2.4.2 Repliement et duplication du graphe contracté

Nous avons indiqué que le but de la phase de contraction est de produire, à chaque pas d'exécution, un graphe d'environ deux fois moins de sommets que le graphe fourni. Cependant, le graphe reste toujours réparti sur les p processeurs, qui possèdent donc deux fois moins de sommets. Le nombre d'arêtes quant à lui ne diminue que dans une proportion moindre : le nombre de sommets ayant un voisin non local augmente donc. Cette augmentation de la taille des interfaces de communication entre les processeurs sera responsable d'une possible dégradation des performances en temps et, dans une moindre mesure, de la qualité des contractions à venir.

Nous avons donc implanté un système de repliement du graphe contracté sur la moitié des processeurs, c'est-à-dire que seule une moitié des processeurs possédera le graphe contracté. Ce repliement est possible car la taille du graphe contracté étant en $\Theta(\frac{n}{2})$, l'occupation mémoire sur les $\frac{p}{2}$ processeurs possédant ce graphe contracté sera alors en $\Theta(\frac{\frac{n}{2}}{\frac{p}{2}}) = \Theta(\frac{n}{p})$, soit la même que celle du graphe initial. Ce repliement permet de diviser par deux le nombre des interfaces, sans que la taille de celles restantes ne soit modifiée. On divise donc par deux la taille totale de la zone d'échange.

D'autre part, l'utilisation du repliement permet de travailler à charge mémoire constante sur les processeurs possédant le graphe contracté. Cependant, l'autre moitié des processeurs n'effectuant plus de travail, nous perdons la moitié de la capacité totale de calcul. Pour éviter de gaspiller les ressources de calcul, nous effectuons simultanément une duplication du graphe contracté replié sur l'autre moitié des processeurs, comme illustré en figure 2.8.

L'intérêt de cette duplication est que, du fait que l'étape de contraction est en général non-déterministe, les contractions suivantes seront différentes sur les deux ensembles de processeurs, et ces différences pourront avoir un impact significatif sur la qualité du partitionnement obtenu.

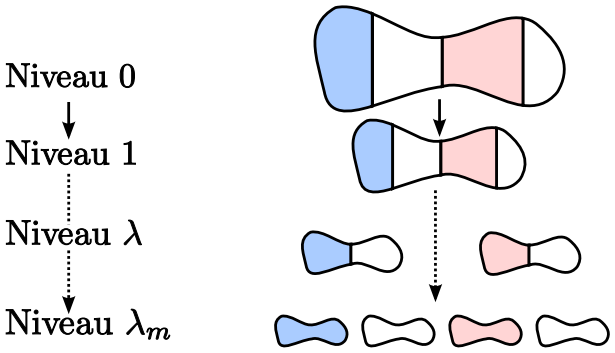


FIG. 2.8: Schéma de la phase de contraction avec repliement et duplication, avec un graphe initial distribué sur quatre processeurs ; au niveau λ, le graphe est replié et dupliqué sur deux groupes de deux processeurs ; au niveau λ_m, chacun des processeurs possède sa version du graphe, qui n'est plus distribué.

Néanmoins, ce repliement avec duplication n'est pas toujours souhaitable car il induit un sur-

coût mémoire. En effet, si n est le nombre de sommets du graphe initial, le nombre de sommets du graphe obtenu par la k-ième contraction est d'environ $\Theta(\frac{n}{2^k})$. L'espace mémoire nécessaire pour les λ_m niveaux de contraction est donc, lorsqu'il n'y a pas de duplication :

$$\begin{aligned}\sum_{k\in[\![1;\lambda_m]\!]} \frac{n}{2^k} &= n \cdot \frac{1-\frac{1}{2^{\lambda_m}}}{1-\frac{1}{2}} \\ &\leq 2n \ .\end{aligned} \tag{2.15}$$

Si l'on effectue une duplication à chaque niveau, l'occupation mémoire requise pour un niveau reste constante et égale à n ; la taille totale nécessaire pour les λ_m niveaux est donc $\lambda_m \cdot n$.

Les besoins de scalabilité mémoire obligent donc à faire un compromis entre les gains possibles en temps et en qualité et les surcoûts mémoire qu'ils induisent par les repliements et les duplications.

Dans la pratique, nous n'effectuons le repliement avec duplication que lorsque la taille occupée par le graphe sur chaque processeur devient plus petite qu'un seuil fourni en paramètre. Nous ne permettons par exemple le repliement et la duplication que si le nombre moyen de sommets par processeur est inférieur à une centaine de sommets.

2.5 Résultats expérimentaux pour la contraction

2.5.1 Implantation des algorithmes

L'implantation des différents algorithmes présentés précédemment repose sur la structure de graphe distribué présentée à la section 1.6.3 de la page 32. La conception est modulaire et se décompose en trois parties :

1. le module d'appariement ;
2. le module de construction du graphe contracté ;
3. le module de repliement/duplication.

2.5.1.1 Implantation de l'appariement

Le parcours des sommets reprend les technologies développées dans la version séquentielle de Scotch [64]. Ainsi, le parcours « aléatoire » des sommets s'effectue de manière à éviter de causer trop de défauts de cache. Le principe est de découper le parcours des sommets en un parcours de blocs de sommets, la taille d'un bloc étant choisie de façon à ce qu'il puisse loger dans le cache. La taille d'un bloc est en fait un nombre premier, ce qui permet de le parcourir en se déplaçant à chaque fois d'un nombre quelconque mais régulier, afin que lorsqu'on retombe sur un sommet déjà traité, on soit sûr que tous les sommets l'ont été. Le principe de ce parcours est basé sur l'arithmétique modulaire et le théorème des restes chinois.

Ensuite intervient la phase de synchronisation, qui exploite l'un des deux algorithmes décrits précédemment (cf 2.2). La sélection aléatoire des sommets, introduite à la section 2.3.3, est simplement implantée par un tirage aléatoire d'un entier entre 1 et le degré du sommet, puis le sommet est étudié quand ce nombre est inférieur au nombre de voisins locaux. Toutes les communications inter-processus utilisent la bibliothèque MPI, et ce code est, à l'heure actuelle, mono-threadé.

Le nombre d'itérations effectuées pour la phase d'appariement n'est pas fixe, au contraire de ParMeTiS, et ne dépend pas non plus d'un taux de contraction à atteindre absolument. En

effet, fixer le nombre de boucles peut impliquer d'en faire soit trop dans le cas où la convergence est rapide, par exemple lorsque les interfaces de communication du graphe distribué sont petites, soit de ne pas aller assez loin et d'obtenir ainsi un taux de contraction assez faible. Ce choix peut être cohérent dans le cas où l'on choisit de limiter la contraction et d'augmenter le nombre de niveaux, mais dans notre cas, nous souhaitons conserver l'efficacité de la contraction.

Le critère d'arrêt retenu fait référence à la vitesse de convergence de l'algorithme qui correspond en fait au taux d'appariements réalisés lors de la dernière boucle, c'est-à-dire au rapport entre le nombre de sommets appariés et le nombre de ceux qui étaient appariables au début de cette boucle. Ce critère permet d'une part de ne pas arrêter la contraction alors qu'elle semblait pouvoir se poursuivre assez facilement, et d'autre part de ne pas continuer lorsque visiblement les itérations perdent de leur efficacité.

Notre principale stratégie pour la synchronisation est en fait un couplage de l'algorithme de synchronisation P « à la ParMeTiS », associé à la technique de sélection des demandes, suivi de l'algorithme L utilisant le coloriage. Le changement de méthode survient lorsque la vitesse d'appariement de l'algorithme P devient inférieure à un certain seuil prédéfini.

À la fin de l'appariement a lieu la phase de répartition des migrations des sommets, pour finaliser les appariements distants. La sélection consiste actuellement simplement à avoir une symétrie entre chaque couple de processeurs, c'est-à-dire que le processeur p_i envoie le même nombre de sommets (à un près) qu'il en reçoit du processeur p_j. Ce critère présente l'avantage d'être rapide à évaluer et, dans le cas d'une distribution initiale bien équilibrée du graphe, permet en pratique de conserver cet équilibre.

2.5.1.2 Implantation de la construction

Ce module exploite conjointement les données contenues dans la structure du graphe distribué initial et les données obtenues lors de la phase d'appariement. Il implante les deux phases de communication décrites à la section 2.4.1.

La première étape de communication correspond à une opération de diffusion, bloquante pour le moment (mais le volume de données à transférer est assez faible), et utilise les routines classiques de diffusion d'informations associées aux sommets du graphe distribué, grâce à l'utilisation de son halo, décrit en section 1.6.3.

Concernant la seconde phase de communication, à savoir le transfert des informations concernant les arêtes, il est nécessaire de préciser que les sommets du graphe contracté issus d'un appariement avec un sommet distant sont numérotés en dernier. Ainsi, lors de la construction, il est possible de traiter les appariements concernant uniquement des sommets locaux pendant que l'on effectue les communications. Dans le cas, fréquent, où les interfaces sont petites par rapport à la taille totale du graphe, le recouvrement des communications est total.

À ce niveau, on dispose d'un graphe contracté distribué sur les p processeurs. Selon les choix de stratégie faits par l'utilisateur, il est possible d'effectuer ou non l'étape de repliement/duplication décrite en section 2.4.2.

2.5.1.3 Implantation de la phase de repliement/duplication

Durant cette phase, le graphe distribué va être redistribué sur la moitié de processeurs (en partie entière supérieure), avec une possibilité de le répliquer sur l'autre moitié. En plus d'optimiser la localisation des données, le repliement permet de rééquilibrer la distribution entre les différents processeurs.

Lorsque la duplication est demandée, la construction des graphes repliés peut être multi-threadée dans le cas où l'implantation de MPI supporte le niveau de threads `MPI_THREAD_MULTIPLE`, requise pour être compatible avec MPI-2. L'intérêt du multi-threading est de permettre un recouvrement des communications, les deux sous-graphes pouvant être construits en même temps, à la manière du repliement survenant pour la méthodes des dissections emboîtées, et décrit en section 1.6.2.

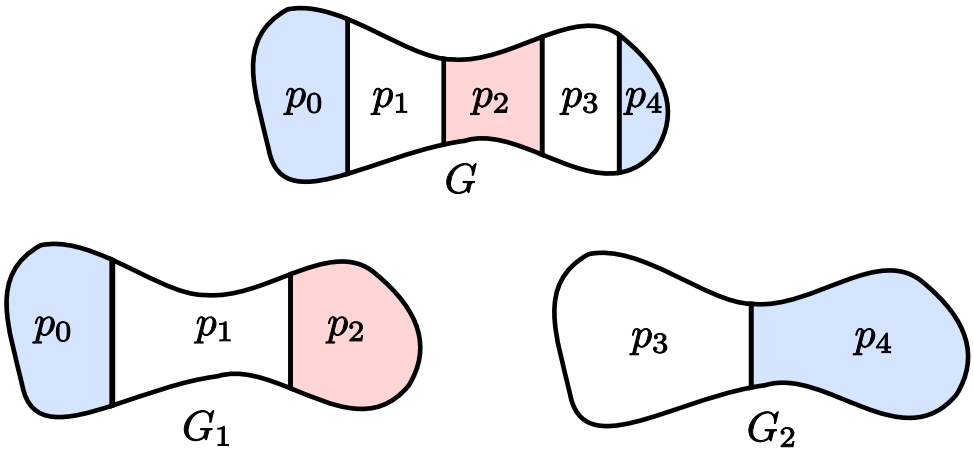


FIG. 2.9: Repliement et duplication d'un graphe G distribué sur cinq processeurs.

La figure 2.9 montre le repliement et la duplication d'un graphe G, initialement distribué sur cinq processeurs, sur respectivement trois et deux processeurs. Les graphes repliés G_1 et G_2 correspondent au même objet que G mais sont néanmoins différents car distribués de façon différente sur un ensemble différent de processeurs.

L'implantation multi-threadée du repliement et de la duplication permet de construire simultanément les graphes G_1 et G_2, chaque processeur utilisant une thread pour chacun des deux graphes. Ces deux threads effectuent en fait des communications antagonistes, l'une ne réalisant que des envois de données, l'autre seulement des réceptions. Ceci permet de profiter des capacités *full-duplex*[8] présentes sur la plupart des réseaux, au lieu de n'exploiter que la moitié de la capacité de transfert qu'ils offrent lors d'une utilisation unidirectionnelle.

Ainsi, en se basant sur l'exemple de la figure 2.9, les processeurs p_0, p_1 et p_2 peuvent envoyer leurs données vers les processeurs p_3 et p_4 tout en recevant simultanément les données de ceux-ci, et inversement.

Le recours à un repliement/duplication multi-threadé permet donc de conserver le même temps de redistribution que celui d'un simple repliement.

2.5.2 Résultats obtenus

2.5.2.1 Probabilités d'appariement à la première itération

Pour cette expérience, nous cherchons à évaluer la probabilité d'appariement distant lors de la première itération, et son influence sur le taux de contraction atteint ainsi que sur le nombre d'itérations nécessaires pour l'atteindre.

Nous prenons l'exemple du graphe `audikw1`, présenté à la section 1.6.4. Nous nous plaçons ici sur seize processeurs.

[8] Possibilité d'utiliser simultanément les deux sens communications.

Stratégie	Probabilité d'appariements distants	Nombre de Boucles	Taux de contraction
P	0.30	11	0.56
L	0.55	12	0.51
P,L	0.31	17	0.51
L,P	0.55	12	0.51

Tab. 2.1: Conséquences de l'utilisation de différents algorithmes de synchronisation sur le calcul du premier niveau de contraction calculé sur le graphe `audikw1`. Les couplages des algorithmes P et L sont notés P,L et L,P lorsqu'on applique P puis L et L puis P respectivement.

Stratégie	Probabilité d'appariements distants	Nombre de Boucles	Taux de contraction
P	0.04	5	0.90
L	0.64	12	0.53
P,L	0.06	12	0.53
L,P	0.64	12	0.53

Tab. 2.2: Conséquences de l'utilisation des différents algorithmes de synchronisation sur le calcul du premier niveau de contraction sur le graphe `audikw1` permuté aléatoirement. Les abréviations sont celles déjà utilisées dans le tableau précédent.

Le tableau 2.1 présente les résultats collectés pour le premier niveau de contraction, en fonction des algorithmes de synchronisation employés. Le graphe est ici utilisé sans permutation. Nous constatons que, comme prévu, l'algorithme P de synchronisation autorise moins d'appariements distants lors de la première boucle. Comme notre critère d'arrêt porte sur la vitesse de contraction, et que celle obtenue avec l'algorithme P est plus petite que les autres, il en résulte un nombre de boucles équivalent à celui obtenu lors de l'exécution des autres algorithmes, le taux de contraction atteint étant plus faible.

Nous pouvons aussi observer que les couplages permettent d'obtenir eux aussi le même taux de contraction que l'utilisation du seul algorithme de synchronisation avec coloriage. Toutefois, une augmentation significative du nombre d'itérations est visible lorsqu'on utilise l'algorithme de coloriage en second, mais ce résultat est seulement ponctuel et ne se reproduit pas dans les autres niveaux.

Le tableau 2.2 présente les résultats obtenus par les mêmes expérimentations sur le même graphe que précédemment mais celui-ci étant permuté aléatoirement. L'intérêt est que, la distribution des sommets se faisant dans PT-Scotch par tranches définies sous la forme d'intervalles dans la numérotation, nous pouvons ainsi changer la manière de distribuer le graphe sur les processeurs. En général, la numérotation initiale des graphes produit une répartition correcte, sans trop de communications. Après permutation aléatoire, au contraire, la distribution des sommets sur les processeurs va aussi devenir aléatoire, rendant les appariements distants encore plus importants.

L'information essentielle procurée par ce tableau est que l'algorithme P de synchronisation voit sa probabilité d'appariements distants lors de la première itération chuter dramatiquement lorsque le graphe est réparti aléatoirement sur l'ensemble des processeurs. Un algorithme iso-

trope, au contraire, privilégierait les appariements distants, les sommets voisins non locaux étant majoritaires. Il en résulte un mauvais taux de contraction pour l'algorithme P, heureusement obtenu après un petit nombre d'itérations, preuve que la vitesse d'appariement est très faible.

Pour les autres algorithmes, l'utilisation du coloriage permet de conserver le nombre de boucles ainsi que le taux de contraction qui avaient été obtenus sur le graphe initial. La qualité de la contraction ne dépend donc avec ces méthodes que de la topologie du graphe, comme l'avait prévu l'étude théorique de la section 2.3.2. Ceci permet de se passer d'une coûteuse phase de redistribution de données avant d'appliquer l'algorithme d'appariement, qui parait indispensable au bon fonctionnement de l'algorithme P. En outre, cette redistribution ne pourra jamais être parfaite, en obtenir une de bonne qualité signifiant que l'on sait déjà partitionner le graphe, ce qui est notre but !

2.5.2.2 Intérêt de la sélection

Stratégie	Probabilité d'appariements distants	Nombre de Boucles	Taux de contraction
P sans sélection	0.04	5	0.90
P avec sélection	0.06	7	0.80

Tab. 2.3: Influence de la sélection des demandes d'appariements distants sur le premier niveau de contraction avec le graphe `audikw1` permuté aléatoirement.

Le tableau 2.3 montre l'influence de l'utilisation de l'algorithme de sélection des demandes d'appariements, dans le cas d'une utilisation avec l'algorithme P de synchronisation, sur le graphe `audikw1` distribué aléatoirement. On constate que cette sélection permet d'augmenter le taux d'appariements distants et d'améliorer le taux de contraction, comme cela avait été prévu à la section 2.3.3.

Cependant, le véritable intérêt de la sélection est de permettre d'utiliser le premier algorithme de sélection, l'algorithme P, d'une manière plus efficace, pour ensuite terminer en utilisant l'algorithme L de synchronisation. Effectivement, les itérations de l'algorithme P de synchronisation sont moins coûteuses car elles nécessitent moins de communications globales. Il peut donc s'avérer plus intéressant, en terme de rapidité des calculs, d'utiliser d'abord cette méthode pour basculer ensuite vers la méthode L avec coloriage qui permet d'obtenir un meilleur taux de contraction.

Le tableau 2.4 montre les conséquences pratiques de l'utilisation des différentes stratégies de contraction dans le cas de la renumérotation de la matrice `audikw1`. Les calculs ont été effectués sur une machine octo-processeurs AMD Opteron dual-core à 1800 MHz, donc comportant seize cœurs au total. La bibliothèque MPI utilisée est MPICH-2, en mémoire partagée.

Si l'on observe la première partie du tableau, qui présente le cas où le graphe est fourni avec sa numérotation initiale, et qui est donc a priori « correctement » réparti sur les différents processeurs, on constate que le taux de contraction ρ varie assez peu selon les stratégies, au contraire des autres colonnes. En effet, les temps de calcul peuvent varier de plus de cinquante pour cent dans le cas où huit processeurs sont utilisés, et quarante pour cent avec seize processeurs. Le coût de la factorisation peut lui varier de 26% dans le cas à huit processeurs, et 33% dans le cas où on utilise seize processeurs.

La stratégie P est la plus lente, et ceci peut s'expliquer par les taux de contraction des

Stratégie	8 processeurs ρ	t	OPC	16 processeurs ρ	t	OPC
Graphe `audikw1`, numérotation originale						
P	0.55	81.22	7.06e+12	0.56	67.72	6.75e+12
SP	0.52	52.89	5.78e+12	0.52	48.34	7.33e+12
L	0.51	55.48	5.77e+12	0.51	52.29	5.58e+12
SP+L	0.51	55.32	5.75e+12	0.51	50.29	5.52e+12
L+SP	0.51	59.86	5.59e+12	0.51	51.45	5.80e+12
Graphe `audikw1`, renuméroté aléatoirement						
P	0.77	97.67	6.11e+12	0.90	74.29	6.71e+12
SP	0.65	82.18	5.67e+12	0.83	66.79	6.66e+12
L	0.53	86.26	5.39e+12	0.53	71.42	5.56e+12
SP+L	0.53	78.99	5.49e+12	0.53	71.93	5.51e+12
L+SP	0.53	81.92	5.59e+12	0.53	74.02	5.63e+12

Tab. 2.4: Conséquences des différentes stratégies d'appariements : SP, P avec sélection des communications ; SP+L, SP puis L ; L+SP, L puis SP. ρ désigne le taux de contraction pour le premier niveau de dissection, t le temps d'exécution en secondes sur un octoprocesseur AMD Opteron dual-core à 1800 MHz, OPC le nombre d'opérations nécessaires pour la factorisation numérique.

autres niveaux, qui ne figurent pas dans le tableau mais sont responsables, par leur faiblesse, d'une augmentation du nombre de niveaux. De plus, à cause d'une perte dans l'isotropisme de l'appariement, les niveaux sont moins proches topologiquement qu'avec les autres stratégies, ce qui conduit à une mauvaise qualité globale de la numérotation.

Le fait d'associer une phase de sélection à la méthode P permet de diminuer de manière évidente les temps de calcul, de respectivement 34% et 29% pour huit et seize processeurs. Cependant, on peut remarquer une très nette diminution de la qualité de la renumérotation lors du calcul de celle-ci sur seize processeurs, alors qu'on observe une amélioration avec huit processeurs. Ce fait peut sans doute s'expliquer par un manque de stabilité du procédé et une forte sensibilité à l'aléa.

Les stratégies utilisant le second algorithme de synchronisation, L, sont, en revanche, relativement équivalentes au niveau des temps de calcul et de la qualité des résultats obtenus, avec un léger avantage pour la méthode SP+L.

La seconde partie du tableau 2.4 concerne les résultats pour la renumérotation du même graphe `audikw1`, mais les sommets de celui-ci étant déjà permutés aléatoirement.

La qualité de renumérotation des stratégies P et SP sont encore les plus mauvaises, la faute visiblement en revenant à une contraction peu efficace en termes de taux et de qualité de contraction.

Concernant les autres approches, la stratégie SP+L semble offrir le meilleur compromis entre vitesse d'exécution et qualité de renumérotation. Ses premières itérations utilisant SP sont assez peu coûteuses en temps et la terminaison avec l'algorithme L permet d'obtenir des taux de contraction satisfaisants. La présence de l'algorithme L permet qui plus est de fournir un résultat indépendant de la numérotation initiale du graphe, comme nous l'avons constaté dans la section 2.5.2.1.

Les faibles différences observées pour le temps d'exécution peuvent provenir du fait que les communications collectives ne présentent pas de surcoût par rapport aux communications

point-à-point, la machine étant à mémoire partagée.

2.5.3 Conclusions des expérimentations sur la contraction parallèle

Nos expérimentations mettent en évidence que la phase de contraction revêt une importance cruciale pour la qualité de bipartitionnement obtenue lors de l'application d'un schéma multi-niveaux. De plus, sa parallélisation est délicate, comme le montrent les résultats produits par l'utilisation de la méthode P de synchronisation, la plus naturelle, utilisée par ailleurs dans PARMETIS.

La méthode L de synchronisation que nous avons introduite, utilisant une coloration de Luby du graphe des processeurs, permet de mieux préserver les données topologiques du graphe lors de sa contraction, et donc d'obtenir un meilleur partitionnement. En revanche, une séquentialisation des échanges de messages peut se produire, rendant cet algorithme a priori moins scalable que le précédent.

L'utilisation de l'algorithme de type PARMETIS semble donc incontournable, au moins dans un premier temps, pour garder une scalabilité en temps. Nous avons donc introduit une phase de sélection, qui permet d'optimiser l'utilisation de cet algorithme.

Le couplage de cet algorithme optimisé par la sélection avec l'algorithme de synchronisation basé sur le coloriage permet d'obtenir de bonnes contractions, indépendantes de la distribution initiale du graphe, tout en conservant de bonnes performances en temps. Il serait cependant intéressant de valider cette approche sur un grand nombre de processeurs et de vérifier sa scalabilité en temps.

Nous avons aussi introduit la notion de repliement/duplication lors de cette phase de contraction, mais l'influence de cette phase sur la qualité du bipartitionnement n'apparaîtra réellement que lors de la phase d'expansion, qui sera présentée dans le prochain chapitre.

Chapitre 3

La phase d'expansion : améliorations sur le raffinement

Sommaire

Ce chapitre porte sur l'étude de la troisième et dernière phase du schéma multi-niveaux : l'expansion. Celle-ci semble être le pendant de la phase de contraction, néanmoins les problèmes rencontrés pour qu'elle se fasse de manière efficace sont très différents.

En effet, comme décrit dans la section 1.3.3 de la page 20, cette phase d'expansion a pour but de projeter sur le graphe initial la partition calculée sur le graphe contracté. Cette phase est en fait une succession de projections, depuis le graphe le plus contracté jusqu'au graphe initial. À chaque niveau, la partition n'est pas seulement projetée mais aussi raffinée, afin de mieux prendre en compte la topologie plus fine des graphes. La figure 3.1 illustre le principe de cette phase d'expansion.

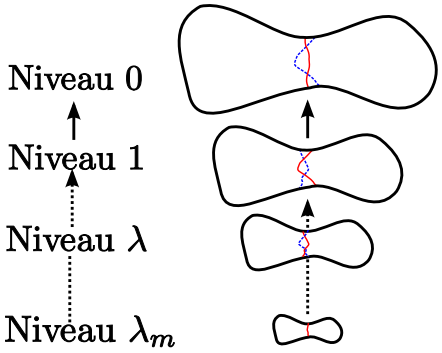


FIG. 3.1: Principe de fonctionnement de l'expansion lors d'un bipartitionnement multi-niveaux du graphe G. La partition calculée au niveau λ_m correspondant à la topologie la plus grossière est projetée de proche en proche jusqu'au niveau 0 qui correspond au graphe initial. À chaque pas, un raffinement est appliqué au séparateur projeté (en tirets) pour obtenir un séparateur localement optimal (en traits pleins).

La principale difficulté du passage en parallèle de cette phase est de paralléliser de manière efficace ce raffinement, qui consiste en une optimisation locale de la solution projetée.

3.1 Exploitation du parallélisme intrinsèque du schéma multi-niveaux

La phase d'expansion ne se résume pas seulement au seul algorithme de raffinement, même s'il s'agit de son constituant le plus important. Effectivement, la construction de la partition projetée est aussi un composant, certes plus basique, mais indispensable au bon fonctionnement de la phase d'expansion.

3.1.1 Projection parallèle

Le principe de la projection est le même en séquentiel ou en parallèle : les sommets du graphe $G_{\lambda-1}$ de niveau $\lambda - 1$ sont placés dans la partie qui était celle du sommet de G_λ issu de leur regroupement.

Nous noterons $p_{\lambda-1}$ la projection de la partition du graphe G_λ sur le graphe $G_{\lambda-1}$. Si $c : V_{\lambda-1} \to V_k$ est la fonction de contraction, qui associe à chaque sommet de $G_{\lambda-1}$ son image dans G_λ, et si $\pi_i(V_\lambda)$ représente l'ensemble des sommets de G_λ se trouvant dans la partie i selon la partition π, on peut définir p_λ de la manière suivante :

$$\forall i \in [\![0; k-1]\!], \forall v \in V(G_{\lambda-1}), c(v) \in \pi_i(V(G_\lambda)) \implies v \in \pi_i(V(G_{\lambda-1})) \ . \tag{3.1}$$

Ce principe très simple rend néanmoins indispensable l'étape de raffinement, certains sommets du séparateur de $G_{\lambda-1}$ pouvant être redondants lorsque les sommets du séparateur de G_λ sont expansés. Si l'on observe le résultat de la projection dans le cas de l'exemple de la figure 3.2, on constate que les sommets u et v appartiennent tous deux au séparateur car leur appariement w en faisait partie. Cependant, u et v ne sont adjacents qu'à des sommets appartenant au séparateur ou à une seule autre partie ; il est donc nécessaire, dans le but de minimiser la taille du séparateur, de faire basculer un de ces deux sommets dans une des parties adjacentes : π_0 dans le cas de u ou π_1 s'il s'agit de v.

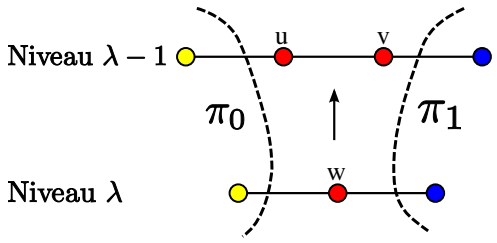


FIG. 3.2: Exemple de perte de qualité du partitionnement due à la projection directe de la partition du niveau λ sur le graphe de niveau $\lambda - 1$. Les sommets u et v sont placés sur le séparateur du graphe $G_{\lambda-1}$ car ils forment le sommet w sur le graphe G_λ et qui appartenait au séparateur. Cependant, seul un des sommets u ou v est utile à la séparation.

La figure 3.3 illustre les différentes étapes de la phase d'expansion, depuis le calcul d'une partition sur le graphe de niveau λ (figure 3.3a), suivi de la projection de celle-ci sur le graphe de niveau $\lambda-1$ (figure 3.3b) et son raffinement pour obtenir le partitionnement localement optimal du graphe $G_{\lambda-1}$ (figure 3.3c).

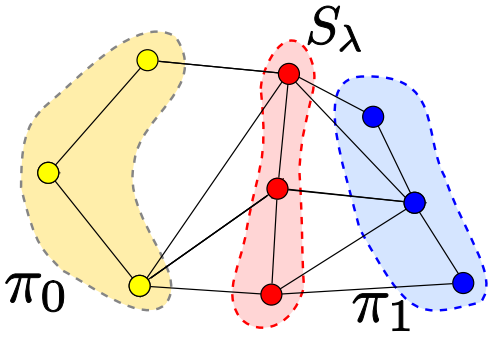
En parallèle, l'algorithme de projection reste très simple, comportant en fait seulement une boucle sur les sommets du graphe le plus grossier G_λ et plaçant les sommets correspondants de $G_{\lambda-1}$ dans la partie associée. Les seules communications surviennent lors de l'étude d'un sommet issu d'un appariement distant, mais l'ordre de parcours des sommets du graphe étant sans importance, elles peuvent être recouvertes par l'analyse des sommets appariés localement.

3.1.2 Exploitation du repliement et de la duplication des graphes grossiers

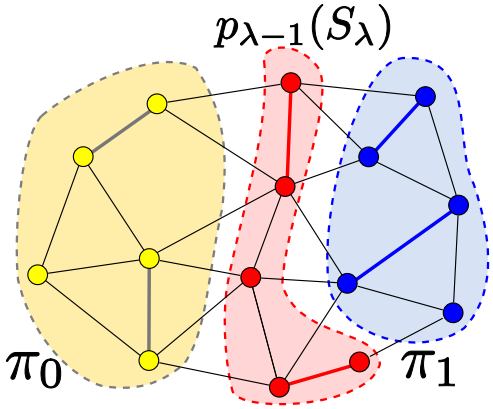
Nous avons précédemment décrit dans la section 2.4.2 de la page 59, comment pouvait survenir un repliement et une duplication du graphe grossier lors du processus de contraction. Nous avions évoqué que la duplication pouvait apporter un gain dans la qualité du partitionnement car elle permet de fait d'effectuer des essais multiples pour un coût en temps pratiquement nul.

Cependant, lors de la phase d'expansion, lorsqu'une duplication a été effectuée pour passer du niveau $\lambda - 1$ au niveau λ, nous sommes en présence de deux solutions sur le niveau grossier λ. Nous avons choisi de continuer les calculs seulement sur la partition de meilleure qualité sur le graphe grossier G_λ. La solution de conserver uniquement le meilleur partitionnement raffiné sur le graphe fin $G_{\lambda-1}$ permet certes d'obtenir au final une meilleure qualité, mais au prix d'un double calcul de la projection et du raffinement. L'objectif de PT-SCOTCH étant d'obtenir la meilleure qualité possible en évitant les surcoûts en temps, nous avons opté pour une sélection a priori plutôt qu'a posteriori.

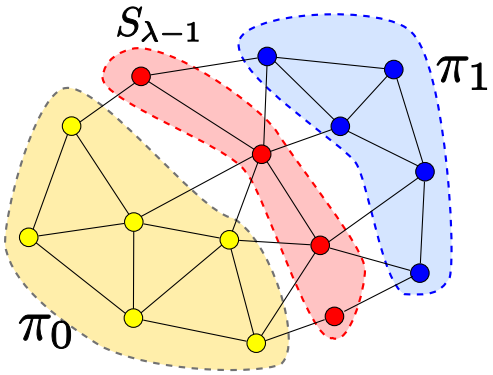
Les résultats concernant l'usage du repliement et de la duplication seront discutés dans la section 3.4.4 de la page 86.



(a) Partition du graphe G_λ.



(b) Partition du graphe $G_{\lambda-1}$ obtenue après la projection de celle du graphe G_λ.



(c) Partition du graphe $G_{\lambda-1}$ obtenue après raffinement de la partition projetée.

FIG. 3.3: Exemple d'évolution d'un partitionnement sommet lors du passage du niveau λ à $\lambda-1$.

3.2 Une amélioration du raffinement : la bande

Le raffinement est la partie de l'expansion dont la parallélisation nécessite le plus de soin. Le principal problème vient du fait que les heuristiques utilisées pour le raffinement sont généralement des heuristiques d'optimisation locale intrinsèquement séquentielles, comme les algorithmes de Fiduccia-Mattheyses ou de Kernighan-Lin présentés dans la section 1.2.2.1 de la page 12 et utilisés ici dans leur version séparation-arêtes [37].

3.2.1 Observation du comportement de l'heuristique de Fiduccia-Mattheyses lors d'un raffinement

Nous avons déjà dit que le bon fonctionnement de l'approche multi-niveaux provient de la conservation de la structure topologique du graphe entre les différents niveaux. Cette proximité topologique nous indique aussi que les partitions localement optimales doivent être elles aussi assez proches entre deux niveaux consécutifs. C'est la transmission de cette proximité, depuis le niveau le plus grossier λ_m jusqu'au graphe initial du niveau 1, qui permet au schéma multi-niveaux de pouvoir proposer une solution « globalement » optimale.

Nous nous plaçons maintenant dans le cas que nous avons étudié pour réaliser le renumeroteur de matrices creuses, à savoir le bipartitionnement récursif de graphes. Une partition se compose donc de deux parties séparées par un séparateur sommet S, la seule donnée de la position de ce séparateur suffisant à déterminer la partition associée. Nous notons $\pi(\lambda)$ la partition localement optimale trouvée pour le graphe contracté de niveau λ et $p_{\lambda-1}(\pi(\lambda))$ sa projection sur le graphe de niveau $\lambda - 1$.

Comme les graphes des niveaux λ et $\lambda-1$ sont proches topologiquement, la partition $\pi(\lambda-1)$, issue du raffinement de $p_{\lambda-1}(\pi(\lambda))$, doit être dans le voisinage de cette dernière. En d'autres termes, la projection du séparateur S_λ du niveau λ sur le niveau $\lambda - 1$, noté $p_{\lambda-1}(S_\lambda)$, a dans son voisinage le séparateur $S_{\lambda-1}$ raffiné du niveau $\lambda - 1$.

La proximité topologique des séparateurs $p_{\lambda-1}(S_\lambda)$ et $S_{\lambda-1}$ peut être étudiée en calculant les distances des sommets de $S_{\lambda-1}$ à $p_{\lambda-1}(S_\lambda)$. La figure 3.4 représente un exemple de positionnement des différents séparateurs entre les niveaux λ et $\lambda - 1$ de contraction d'un graphe.

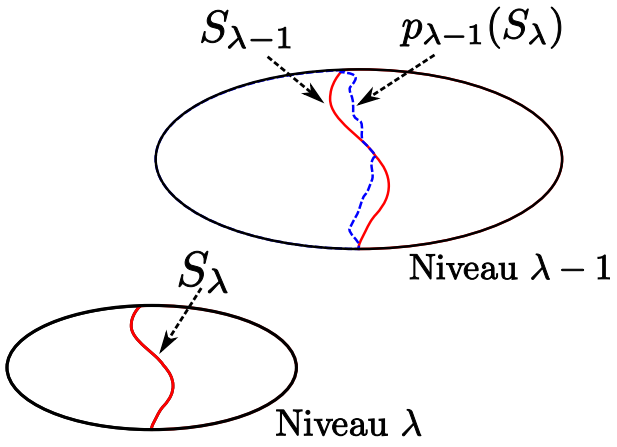


FIG. 3.4: Illustration d'une étape du processus d'expansion : le passage d'un niveau λ au niveau $\lambda - 1$. Le séparateur S_λ du niveau λ est projeté sur le graphe de niveau $\lambda - 1$, noté $p_{\lambda-1}(S_\lambda)$. Le raffinement de celui-ci permet d'obtenir le séparateur amélioré $S_{\lambda-1}$.

Lors d'une étude expérimentale sur la distance des points de $S_{\lambda-1}$ à $p_{\lambda-1}(S_k)$ en utilisant

l'algorithme de Fiduccia-Mattheyses comme raffinement, nous avons obtenu, sur un large ensemble de graphes de types et de tailles différents, que cette distance n'excédait en moyenne que très rarement trois. Le tableau 3.1 montre ces résultats et nous permet de constater que 80% des sommets du séparateur raffiné se trouvaient déjà dans le séparateur projeté, et 99.9% à une distance inférieure à trois.

Distance par rapport à la projection de S_k				
0	1	2	3	≥ 4
80%	17%	1.5%	$< 0.5\%$	$< 0.1\%$

TAB. 3.1: Répartition de la distance des sommets de $S_{\lambda-1}$ par rapport à $p_{\lambda-1}(S_\lambda)$.

Le séparateur localement optimisé se situe donc majoritairement dans une bande de rayon trois autour du séparateur projeté.

3.2.2 Justification de l'existence de la bande

La présence de la grande majorité des sommets du séparateur raffiné dans la bande de rayon trois autour du séparateur projeté peut être expliquée par les propriétés et la construction du schéma multi-niveaux.

En effet, la projection d'un couple de sommets à une distance un sur le graphe G_λ est majorée par trois sur le graphe $G_{\lambda-1}$, comme illustré par la figure 3.5. De fait, les deux sommets contractés peuvent au plus donner chacun deux sommets, dans le cas où la contraction consiste en un appariement. Le sous-graphe des quatre sommets ainsi obtenu a un diamètre d'au plus trois, ce qui signifie que la projection d'un déplacement de distance un sur G_λ correspond à un déplacement d'une distance au plus trois sur $G_{\lambda-1}$.

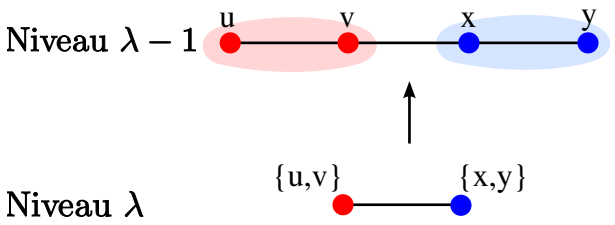


FIG. 3.5: Projection de deux couples adjacents de sommets appariés et conséquence sur leurs distances relatives. La distance dans le graphe de niveau λ était $d(\{u,v\},\{x,y\}) = 1$, alors que dans le graphe de niveau $\lambda - 1$ la distance maximale entre les sommets est $d(u,y) = 3$.

Plus mathématiquement, on peut dire qu'une boule $\mathcal{B}_\lambda(1)$ de rayon un a son image dans une boule $\mathcal{B}_{\lambda-1}(3)$ de rayon trois dans le graphe $G_{\lambda-1}$:

$$\forall (u,v) \in V(G_{\lambda-1}), d(c(u),c(v)) = 1 \implies d(u,v) \leq 3 \ . \tag{3.2}$$

La valeur de la distance image par la projection peut être quantifiée dans le cadre d'un regroupement quelconque de r sommets lors de la contraction : le diamètre d'un regroupement est $r-1$, auquel il faut ajouter le diamètre de l'autre regroupement et l'arête qui relie ces deux sous-graphes. La projection d'une distance 1 dans le cadre du regroupement de r sommets est donc majorée par une distance : $d_r(1) = 2(r-1)+1 = 2r-1$.

Nous notons $d_r : \mathbb{N} \to \mathbb{N}$ la fonction qui associe à une distance d sur le graphe contracté son plus petit majorant pour sa projection sur le graphe non contracté, dans le cas où les

regroupements ont au plus r sommets. Cette fonction est strictement croissante. Par définition, nous avons donc :

$$p_{\lambda-1}(\mathcal{B}_\lambda(1)) \subset \mathcal{B}_{\lambda-1}(p_r(1)) \ . \tag{3.3}$$

Si le séparateur raffiné est déplacé, par rapport à sa position initiale, d'une distance supérieure à $p_r(1)$, cela signifie que ce déplacement était aussi possible sur le graphe contracté. En effet, la fonction p_r étant croissante, le déplacement correspondant sur le graphe contracté est supérieur à un.

Si les algorithmes utilisés pour le raffinement convergent vers l'extremum local sur le graphe contracté, la partition résultante sur ce graphe contracté est de meilleure qualité que celles de son voisinage. Si le raffinement sur le graphe étendu déplace le séparateur S_λ d'une distance supérieure à $d_r(1)$, cela signifie aussi que la projection de ce séparateur $p_{\lambda-1}(S_\lambda)$ sur le graphe contracté est un extremum local. Cependant, ce séparateur se trouve dans le voisinage du séparateur initial $S_{\lambda-1}$, et dans l'hypothèse d'une convergence systématique de l'heuristique de raffinement, cela signifie que les deux partitions engendrées par les séparateurs $S_{\lambda-1}$ et $p_{\lambda-1}(S_\lambda)$ ont le même coût. Ce phénomène est extrêmement rare et c'est pour cela que nous avons décidé de tirer parti des propriétés de localité de la contraction pour utiliser le graphe bande dans la phase d'optimisation locale.

3.2.3 Exploitation de la bande

Il apparaît donc que seuls les sommets situés dans la bande de rayon trois autour de la projection du séparateur du niveau contracté sont réellement utiles aux heuristiques de raffinement.

Notre idée est d'exploiter cette bande pour ne fournir aux heuristiques de raffinement que les sommets dont elles ont théoriquement besoin. Nous formons donc un nouveau graphe composé des sommets de la bande auxquels nous ajoutons deux sommets « ancres », un par partie. Ces deux sommets servent à remplacer les sommets supprimés par rapport au graphe original et sont définis de la manière suivante :

- leur poids est égal à la somme des poids des sommets qu'ils remplacent ;
- ils sont reliés à tous les sommets qui ont un voisin à l'extérieur de la bande.

Grâce à cette définition, le graphe bande possède les mêmes caractéristiques d'équilibre pour les parties que le graphe initial mais aussi, d'après le résultat présenté dans la section 3.2.2, il possède les mêmes caractéristiques topologiques pour le séparateur, et donc les mêmes extrema locaux pour les partitions.

Définition 29 (Graphe bande)
Le graphe bande G_b, dans le cas d'un regroupement par appariements, est donc composé :

- *du sous graphe induit par l'ensemble V_b des sommets de la bande ;*
- *de deux sommets ancres a_0 et a_1 remplaçant les sommets de $V \backslash V_b$ appartenant à chacune des deux parties, et intégrés dans le graphe bande selon les relations (3.5) et (3.6), pour les poids et la connexion aux autres sommets, respectivement,*

avec :

$$V_b = \bigcup_{v \in S_k} \mathcal{B}^3_{\lambda-1}(p_{\lambda-1}(v)) \, ; \tag{3.4}$$

$$\forall i \in \{0,1\}, w_v(a_i) = \sum_{\substack{v \in V \backslash V_b \\ p^{-1}_{\lambda-1}(v) \in \pi_i}} w_v(v) \, ; \tag{3.5}$$

$$\forall v \in V, \forall i \in \{0,1\}, \exists \{v, a_i\} \in E(G_b) \Leftrightarrow (\exists u \in V, u \notin E(G_b) \text{ et } \{u,v\} \in E) \ . \tag{3.6}$$

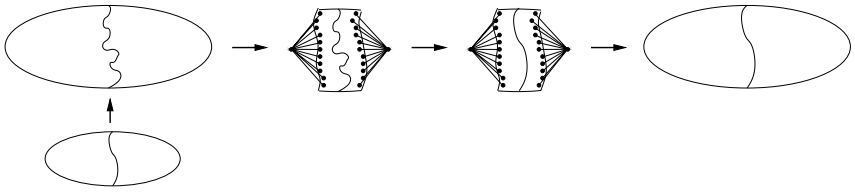


FIG. 3.6: Utilisation de la bande dans le cadre d'un schéma multi-niveaux. Un graphe bande est formé par les sommets situés à proximité du séparateur projeté sur le niveau plus fin, ainsi que de deux sommets ancres représentant les sommets enlevés au graphe initial. Après l'application d'un algorithme d'optimisation, le séparateur raffiné du graphe bande est projeté sur l'intégralité du graphe, et le processus d'expansion peut continuer.

L'intérêt de travailler sur un tel graphe bande est que la taille du séparateur est généralement inférieure de plusieurs ordres de grandeur à celle du graphe. Par exemple, dans le cas d'un graphe $2D$ de taille n, la taille du séparateur est en $\mathcal{O}(n^{\frac{1}{2}})$; dans le cas d'un graphe $3D$ de taille n, elles est en $\mathcal{O}(n^{\frac{2}{3}})$ [53, 70]. La taille du graphe bande est généralement de l'ordre de la taille du séparateur, multiplié par la largeur de la bande, $p_r(1)$. La taille du graphe bande G_b est :

$$|V(G_b)| = |S| \cdot p_r(1) \ll |V(G)| = n \ . \tag{3.7}$$

On peut voir ici un intérêt d'avoir simplement des appariements pour les regroupements lors de la phase de contraction : le ratio de contraction est moins élevé qu'avec des regroupements plus grands mais la proximité topologique est plus simple à garantir, ce qui fait que le graphe bande utile pour le raffinement comporte moins de sommets.

3.3 Utilisations de la bande

L'utilisation du graphe bande à la place du graphe original ouvre de nouvelles perspectives pour le raffinement, de par sa taille et aussi par la connaissance topologique de l'emplacement du séparateur.

3.3.1 Exploitation de la bande par les heuristiques séquentielles classiques de raffinement local

Nous avons d'abord testé la viabilité de la solution consistant à travailler seulement sur le graphe bande avec l'algorithme de raffinement.

Le coût de la construction de ce graphe bande est proportionnel à sa taille et est donc, d'après le résultat (3.7), petit sur la plupart des graphes. L'algorithme utilisé pour extraire ce graphe est un algorithme dont le cheminement est celui d'une diffusion à partir des sommets du séparateur, et peut donc être comparé à des algorithmes de parcours en largeur du type de celui de Gibbs-Poole-Stockmeyer [26].

Nous avons ensuite comparé les résultats de renumérotations de matrices creuses par dissections emboîtées calculées à l'aide d'un schéma multi-niveaux, d'une part en utilisant un raffinement avec Fiduccia-Mattheyses, et d'autre part, en extrayant le graphe bande puis en appliquant sur celui-ci l'algorithme de Fiduccia-Mattheyses. La mesure de la qualité de la renumérotation est celle présentée à la page 29, à savoir OPC, le nombre d'opérations de la factorisation numérique.

À la lumière de ces expérimentations, il apparaît que non seulement le fait d'utiliser la bande ne dégrade pas les solutions obtenues, mais peut même les améliorer et les rendre plus stables.

En effet, dans la version séquentielle de Scotch, la stratégie par défaut pour le raffinement utilise deux exécutions du schéma multi-niveaux et conserve le meilleur résultat, à l'image de la stratégie citée à la page 14.

La stabilité constatée en terme de qualité peut s'expliquer par le fait que comme la localisation du séparateur est contrainte dans une bande définissant le voisinage d'un extremum pseudo-global, l'heuristique de raffinement n'est pas happée par des artefacts dus à la contraction.

Cette première expérimentation nous montre que l'utilisation du graphe bande pour le raffinement lors de l'expansion multi-niveaux est intéressante, même en séquentiel. En effet, dorénavant, seule une exécution de Fiduccia-Mattheyses sur le graphe bande est nécessaire, alors qu'auparavant la stratégie était d'effectuer deux fois le schéma multi-niveaux complet sur le graphe initial. De plus, nous avons déjà dit que le coût de calcul de la bande est peu sensible par rapport à celui d'une exécution de l'heuristique de raffinement. Comme l'utilisation du graphe bande nous dispense d'un calcul par l'algorithme de Fiduccia-Mattheyses, nous sommes globalement gagnants au niveau du temps.

Cependant, il est possible de mieux tirer profit des bénéfices de la réduction de la taille du problème. En effet, cette réduction peut permettre de conserver, dans un premier temps, les heuristiques séquentielles de raffinement lors d'un calcul utilisant un schéma multi-niveaux parallèle.

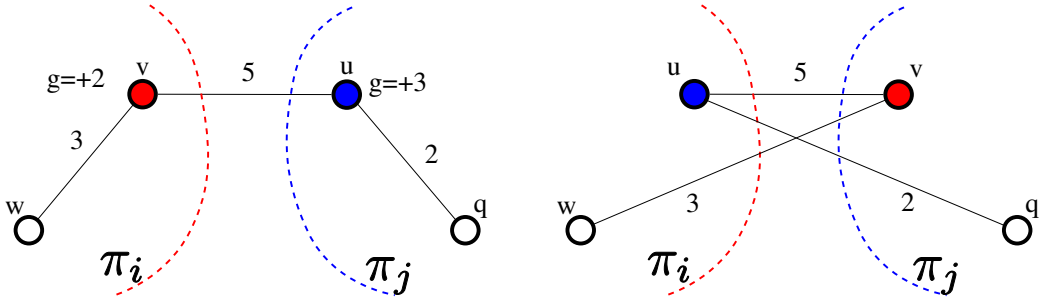
3.3.2 Vers un raffinement parallèle en utilisant le graphe bande

3.3.2.1 Limites du raffinement parallèle

Tout d'abord, rappelons les limites des méthodes de raffinement actuellement exploitées en parallèle.

Nous avons présenté les algorithmes de Kernighan-Lin (algorithme 1 de la page 13) et de Fiduccia-Mattheyses (algorithme 2 de la page 15) et constaté qu'ils étaient intrinsèquement séquentiels. Les tentatives de parallélisation déjà explorées par d'autres auteurs ont toutes été basées sur le relâchement des contraintes qui conduisaient à cette séquentialité. En effet, si l'on prend l'exemple de la situation illustrée par la figure 3.7a, l'application de l'algorithme de Fiduccia-Mattheyses sans autres contraintes conduit au résultat de la figure 3.7b. On constate une très nette augmentation de la coupe alors qu'une diminution était espérée. Ce phénomène n'aurait pas eu lieu en séquentiel car un seul sommet u ou v aurait pu être déplacé à la fois. Une solution pour la parallélisation consiste à verrouiller les sommets pour empêcher les déplacements antagonistes, mais il est alors nécessaire d'effectuer de nombreuses communications, qui nuisent à l'efficacité et à la scalabilité de l'algorithme.

La solution retenue dans PaMeTiS consiste à interdire ces mouvements antagonistes en coloriant le graphe afin de dégager des ensembles indépendants. Les couleurs sont considérées l'une après l'autre, et seuls les individus de la couleur courante peuvent être déplacés. Cette limitation modifie le comportement de l'heuristique par rapport à sa version séquentielle pour deux raisons. La première est que les sommets sélectionnés sont ceux de plus grand gain pour la couleur courante et non forcément ceux de plus grand gain sur le graphe. La seconde concerne justement le calcul du gain, celui-ci n'étant pas mis à jour globalement, mais seulement localement, au sein du voisinage du sommet de la couleur actuelle. D'autre part, la variante de l'algorithme de Fiduccia-Mattheyses utilisée par PaMeTiS interdit les mouvements qui ne font pas progresser directement la qualité de la partition. La capacité de sortir des extrema locaux, qui faisait la force de l'heuristique en séquentiel, est donc perdue. On peut dorénavant voir cet



(a) Situation initiale. Les gains associés aux sommets u et v sont respectivement $+3$ et $+2$. La valeur de la coupe est 5. Des gains de 3 ou 2 peuvent être obtenus en déplaçant u ou v.

(b) Situation finale. Les sommets u et v ont été simultanément changés de parties. La valeur de la coupe est maintenant 10, soit une augmentation de 5 par rapport à la situation initiale.

FIG. 3.7: Problème de mouvements concurrents lors de la parallélisation de l'algorithme de Fiduccia-Mattheyses. Les sommets u et v appartiennent à des processeurs différents, qui exécutent chacun l'heuristique de Fiduccia-Mattheyses.

algorithme de raffinement comme une méthode de gradient, suivant la ligne de plus forte pente de la fonction de coût pour chacune des couleurs considérées tour à tour, pour trouver la solution locale optimale.

Cette limitation, associée au système de coloriage des sommets décrit précédemment, est responsable d'une importante dégradation de la qualité des renumérotations obtenues en parallèle, à mesure que le nombre de processeurs augmente, comme le montre le graphique de la figure 1.13 de la page 30.

3.3.2.2 La bande comme moyen de passage entre le parallèle et le séquentiel

La faible taille du graphe bande par rapport à celle du graphe initial va nous permettre d'exploiter la technique de traitement séquentiel et centralisé du raffinement, même dans les cas où la taille des graphes nécessite l'utilisation d'un outil parallèle de partitionnement. En effet, sur un graphe $3D$ comportant un milliard de sommets, le graphe bande, en vertu des théorèmes de séparation, possède environ un million de sommets. Le graphe original, stocké sur mille processeurs, possède donc en moyenne un graphe bande qui peut être stocké localement sur un seul processeur.

Cet exemple illustre exactement le principe d'une de nos méthodes de calcul du raffinement, dans le cas où le graphe bande est stockable sur un processeur local. Durant le processus de raffinement, le graphe bande est construit, de façon distribuée, puis centralisé, afin que les heuristiques séquentielles de raffinement lui soient appliquées.

En pratique, nous exploitons tous les processeurs disponibles en dupliquant le graphe bande sur chacun d'entre eux, chaque processeur calculant sa version de l'optimisation en exécutant localement l'heuristique de Fiduccia-Mattheyses sur des graphes dont la numérotation a été légèrement modidifiée afin que chacune des exécutions puisse explorer différemment l'espace des solutions. Nous retrouvons ainsi l'idée d'exécutions multiples, mais ici sans surcoût en temps puisque les processeurs sont disponibles. La bande nous permet donc de nous ramener d'un cadre de raffinement parallèle à une méthode multi-séquentielle.

Nous pouvons résumer notre manière de conduire l'expansion ainsi :

1. projection parallèle de la partition calculée au niveau inférieur sur le graphe courant ;
2. extraction parallèle du graphe bande ;
3. multi-centralisation du graphe bande sur chacun des processeurs ;
4. exécution séquentielle de l'heuristique de raffinement sur chacun des processeurs ;
5. diffusion de la meilleure partition obtenue.

Ce cheminement est illustré en figure 3.8.

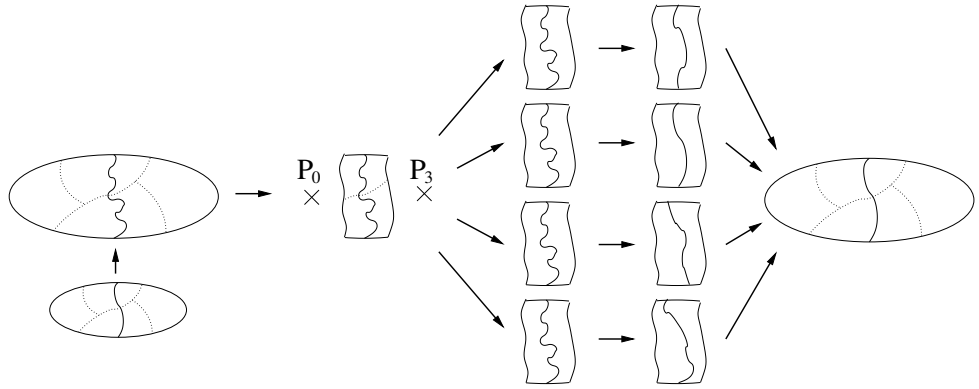


FIG. 3.8: Schéma du raffinement multi-séquentiel du séparateur projeté depuis le graphe distribué sur quatre processeurs, du niveau grossier au niveau plus fin. Une fois le graphe bande construit à partir du graphe fin, une version centralisée est diffusée à chaque processeur. Une version séquentielle de l'algorithme de Fiduccia-Mattheyses est alors exécutée indépendamment sur chaque copie. Le meilleur séparateur obtenu est ensuite redistribué sur le graphe fin.

Les résultats obtenus avec ce mécanisme, appelé « bande FM » car on utilise l'heuristique de Fiduccia-Mattheyses pour le raffinement du séparateur du graphe bande, seront présentés plus tard, dans la section 3.4.3 de la page 81.

3.3.3 Utilisation d'une nouvelle heuristique grâce à la bande : le tonneau des Danaïdes

Cependant, le recours au graphe bande permet aussi l'adoption d'autres stratégies qui peuvent exploiter la meilleure connaissance topologique du voisinage du séparateur.

Nous avons évoqué dans le chapitre 1, l'existence d'heuristiques utilisant un principe de diffusion, comme les algorithmes à bulles, page 14. Ces algorithmes ont été conçus pour partitionner le graphe tout en optimisant la forme des différentes parties. Les algorithmes de raffinement comme celui de Fiduccia-Mattheyses n'exploitent, en effet, que des données locales et ne procurent aucune garantie sur la topologie des différentes parties. Cependant, certaines applications du partitionnement de graphes comme la décomposition de domaines nécessitent d'apporter un soin particulier à la forme des sous-domaines. Pour quantifier la forme d'une partie, on utilise la notion d'*aspect ratio*.

Définition 30 (Aspect ratio)
L'aspect ratio d'une partie est le rapport entre le diamètre de la plus grande boule pouvant être contenue dans la partie et le diamètre de la plus petite boule pouvant contenir la partie.

Cette définition de l'aspect ratio nécessite donc, dans le cadre de l'utilisation de la distance euclidienne, la connaissance de la géométrie du maillage. À défaut, les distances topologiques

peuvent être utilisées, mais en fonction du type de maillage elles peuvent avoir moins de signification vis-à-vis du problème physique étudié.

Il a été montré [14, 12, 73] que les méthodes itératives parallèles nécessitent, pour converger dans de bonnes conditions, que les sous-domaines soient compacts et donc aient des aspects ratio proches de un, ce qui revient à avoir des disques sur un graphe 2D muni de la distance euclidienne. Le principe de la méthode des bulles, qui consiste à faire grossir des boules, et donc à créer des zones ayant un aspect ratio optimal, lorsque les centres des bulles sont judicieusement positionnés. C'est cette recherche des positions des centres des bulles qui rend la méthode lente et délicate à implanter.

Cependant, les sommets ancres du graphe bande peuvent apporter une solution à ce problème : ils sont par définition équidistants de tous les sommets du séparateur, qui représente l'interface entre les deux parties qui sont presque localement optimales.

Les décompositions de domaines utilisant en général des partitionnements arêtes du graphe, nous introduisons donc une extension de la notion de graphe bande à ce type de partitionnement, simplement en considérant dorénavant la distance aux sommets adjacents aux arêtes du séparateur.

Une implantation utilisant l'association de la technique du graphe bande et des algorithmes à bulles a été introduite par François Pellegrini et publiée sous le nom de « tonneau des Danaïdes » [66]. Celle-ci exploite le schéma multi-séquentiel avec un raffinement combinant la technique du graphe bande avec un algorithme de diffusion depuis les sommets ancres.

Dans cet algorithme, les sommets sont identifiés à des réservoirs de liquide qui peuvent s'échanger leur contenu par des connexions correspondant aux arêtes. La quantité de liquide pouvant être transférée à chaque unité de temps est proportionnelle au poids de l'arête associée à la connexion, et la direction du déplacement s'effectue dans le sens normal d'écoulement, c'est-à-dire vers le sommet possédant le moins de liquide.

L'étape de diffusion correspond à injecter $\frac{|V|}{2}$ unités de liquide, du scotch pour le sommet ancre associé à la partie 0 et de l'anti-scotch pour celui associé à la partie 1, à chaque unité de temps depuis les sommets ancres. Les deux liquides se propagent dans le graphe et s'annihilent lorsqu'ils se rencontrent. Après un certain nombre de pas de temps, le système devient stable et une partition peut être définie grâce à la nature du liquide contenu par les sommets, partie 0 pour le scotch, partie 1 pour l'anti-scotch. Le principe de l'algorithme de diffusion est illustré en figure 3.9.

Cette diffusion correspond en fait à une étape de croissance des algorithmes à bulle, mais une seule itération de celle-ci est suffisante dans le cadre de l'algorithme du tonneau des Danaïdes car les centres de diffusion ne sont pas à rechercher comme dans le cas des algorithmes à bulles.

Remarquons que cet algorithme ne considère pas explicitement la contrainte de minimisation de la valeur de la coupe. En fait, comme les mouvements sont restreints aux sommets du graphe bande et que celui-ci est suffisamment étroit, les variations des valeurs de la coupe sont somme toute minimes.

Les résultats de l'algorithme du tonneau des Danaïdes sont observables grâce à la figure 3.10b. Le premier constat est que les frontières et les parties ont une apparence plus douce, suivant mieux les propriétés physiques du maillage. La coupe de la partition reste du même ordre que celle obtenue précédemment.

Cet algorithme présente une nouvelle approche pour le raffinement en vue de réaliser des décompositions de domaines, en exploitant l'organisation structurelle du graphe bande. Cependant, nous pouvons aussi coupler d'autres algorithmes avec ce graphe bande, comme nous allons le voir dans la section suivante.

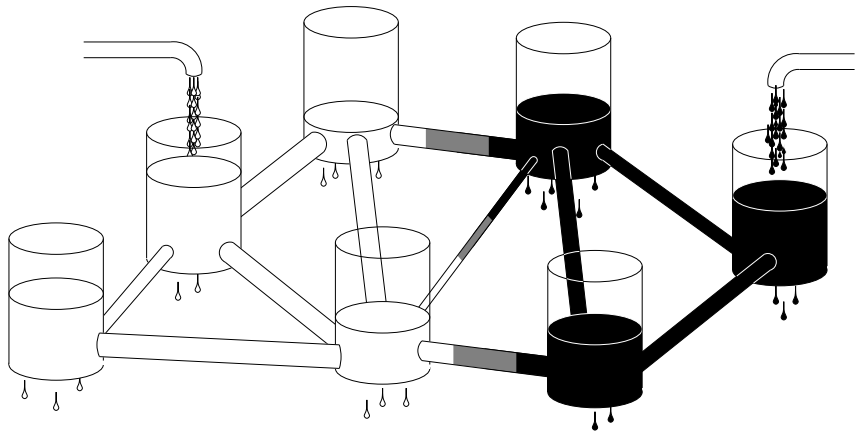
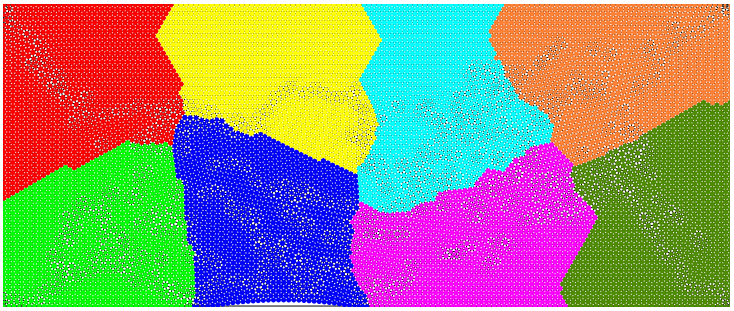
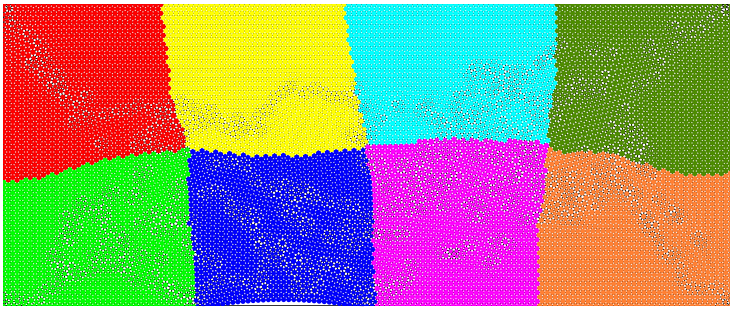


FIG. 3.9: Schéma de principe de la diffusion dans l'algorithme du tonneau des Danaïdes.



(a) Décomposition en 8 domaines en utilisant l'heuristique de Fiduccia-Mattheyses pour le raffinement. La coupe est égale à 714 arêtes.



(b) Décomposition en 8 domaines en utilisant l'algorithme du tonneau des Danaïdes pour le raffinement. La coupe est égale à 713 arêtes.

FIG. 3.10: Exemple de décomposition en 8 domaines du maillage `bump`. Les résultats sont obtenus grâce à une stratégie multi-niveaux associée à la bande.

3.3.4 Autres utilisations possibles de la bande

L'utilisation du graphe bande permet aussi d'ouvrir le raffinement à l'utilisation d'autres types d'heuristiques qui n'étaient jusqu'à présent que peu ou pas exploitées. En effet, dans l'exemple précédent, il est décrit une nouvelle stratégie de raffinement utilisant les techniques de bulles mais il est aussi possible de se servir de la bande avec, par exemple, les différentes classes d'heuristiques présentées dans la section 1.2.2 de la page 11.

Dans le cadre de notre projet PT-Scotch, cette ouverture a surtout été mise en pratique par l'introduction d'algorithmes génétiques. La description de cette intégration est l'objet du chapitre 4. Cependant, cette approche est compatible avec d'autres méta-heuristiques, comme le recuit simulé ou la recherche tabou.

3.4 Résultats expérimentaux

3.4.1 Implantation du graphe bande dans PT-Scotch

La construction du graphe bande est effectuée par un module permettant de lui appliquer n'importe quelle stratégie de partitionnement passée en paramètre. Cette modularité permet de combiner l'utilisation du graphe bande avec les heuristiques de raffinement que l'on souhaite.

Nous avons étudié l'opportunité d'ajouter une limitation au graphe bande directement dans notre implantation de Fiduccia-Mattheyses, afin d'éviter le surcoût dû à l'appel d'un module externe. Cependant, le temps d'extraction du graphe bande étant négligeable par rapport à la durée d'exécution de l'algorithme de Fiduccia-Mattheyses, le choix d'utiliser un module spécifique nous a semblé plus adapté.

3.4.2 Résultats obtenus avec la bande FM en séquentiel

Avant d'utiliser la technique de l'extraction du graphe bande associée à une optimisation de type Fiduccia-Mattheyses dans un contexte parallèle, nous avons observé le comportement de cette méthode de raffinement en séquentiel sur différents types de graphes.

Le tableau 3.2 montre les résultats obtenus en utilisant Scotch, dans sa version séquentielle, avec différentes stratégies de raffinement. La méthode associant le graphe bande à l'algorithme de Fiduccia-Mattheyses est notée bande FM, l'algorithme de Fiduccia-Mattheyses sur le graphe total est noté FM (1) ou FM (2) en fonction du nombre d'exécutions du schéma multi-niveaux.

Le principal constat est que l'utilisation de la bande FM n'augmente pas les temps de calculs tout en ne détériorant pas la qualité de la renumérotation. Au contraire, les résultats obtenus avec la bande FM peuvent même se révéler meilleurs que ceux obtenus avec une seule exécution de l'algorithme de Fiduccia-Mattheyses, comme avec les graphes `cage14` ou `thermal2`.

Une explication possible, déjà formulée en section 3.3.1, est que l'utilisation du graphe bande évite que l'algorithme de Fiduccia-Mattheyses soit happé par des extrema locaux dus aux artefacts de la contraction, et s'éloigne ainsi de l'extremum pseudo-global trouvé par le schéma multi-niveaux. L'amélioration courante de la qualité avec la stratégie FM(2) peut s'expliquer non-seulement par l'intérêt d'exécuter plusieurs fois l'heuristique de raffinement, mais aussi par le fait que la phase de contraction ne produit généralement pas le même résultat aux deux appels. L'utilisation du graphe bande avec l'heuristique de Fiduccia-Mattheyses semble donc permettre d'atténuer l'impact des artefacts de contraction.

Les résultats en OPC montrent néanmoins que la méthode du graphe bande fonctionne mieux lorsque les graphes sont issus de maillages, comme les graphes `altr4`, `audikw1`, `coupole8000`, par

Graphe	Bande FM		FM (1)		FM (2)	
	OPC	Temps (s)	OPC	Temps (s)	OPC	Temps (s)
altr4	3.46e+8	0.74	3.46e+8	0.70	3.46e+8	1.32
audikw1	5.64e+12	22.11	5.68e+12	21.04	5.28e+12	37.60
bmw32	3.69e+10	1.93	3.04e+10	1.87	3.21e+10	3.17
conesphere1m	1.89e+12	49.69	1.90e+12	47.21	1.81e+12	88.43
coupole8000	7.62e+10	140.22	7.64e+10	135.70	7.46e+10	252.91
oilpan	3.83e+9	0.31	3.49e+9	0.31	2.97e+9	0.46
thread	4.10e+10	0.61	4.10e+10	0.59	3.85e+10	1.02
bcsstk32	1.67e+9	0.48	1.61e+9	0.46	1.30e+9	0.82
G3_circuit	6.60e+9	42.01	6.10e+10	43.22	5.54e+10	80.07
af_shell10	4.57e+11	8.70	4.48e+11	8.80	4.03e+11	14.34
bone010	4.99e+12	18.93	4.99e+12	18.23	4.59e+12	32.40
thermal2	1.61e+10	30.90	1.75e+10	29.64	1.57e+10	54.26
cage14	2.36e+15	128.38	2.37e+15	113.43	2.50e+15	236.17
cage15	4.23e+16	739.00	4.33e+16	682.58	4.06e+16	1242.79
brgm	2.65e+13	95.26	2.65e+13	89.59	2.45e+16	159.33

TAB. 3.2: Résultats obtenus avec un raffinement de type graphe bande couplé avec Fiduccia-Mattheyses (Bande FM) et un raffinement de type Fiduccia-Mattheyses classique avec une (FM(1)) ou deux exécutions (FM(2)) de l'heuristique. Les calculs ont été effectués sur un processeur IBM Power 5 à 1.5GHz.

exemple. Cependant, sur certains graphes moins réguliers, les résultats se montrent aussi assez satisfaisant sur `thread`, `bone010` et même `bcsstk32`. Enfin, d'autres graphes comme `G3_circuit` ou `af_shell10` semblent être moins propices à un raffinement utilisant le graphe bande. En effet, on observe sur ces graphes une légère dégradation de la qualité par rapport à l'utilisation classique de l'heuristique de Fiduccia-Mattheyses. Cependant, cette perte de qualité reste contenue en étant inférieure à une dizaine de pour-cent.

Le graphe `bmw32` semble à part : l'utilisation de la bande augmente le coût de la factorisation de 21% par rapport au meilleur résultat obtenu avec une seule exécution de Fiduccia-Mattheyses. Ceci peut s'expliquer par le fait que l'OPC est certes une mesure de la qualité de la renumérotation mais seulement un estimateur indirect de sa qualité, un meilleur partitionnement n'impliquant pas forcément une meilleure renumérotation. Néanmoins, les solutions FM(2) et Bande FM semblent fournir des arbres d'élimination plus équilibrés, ce qui peut permettre une meilleure efficacité pour les outils parallèles de factorisation.

3.4.3 Résultats obtenus avec la bande FM parallèle

La technique du graphe bande semble garantir des résultats voisins de ceux obtenus en travaillant sur la totalité du graphe ; c'est du moins ce qui ressort de l'étude effectuée avec la version séquentielle de SCOTCH. Le passage en parallèle de cette technique a donc semblé envisageable car elle pourrait permettre de conserver la même qualité pour les résultats que celle obtenue par l'exécution des algorithmes séquentiels. En outre, l'exploitation de la multi-séquentialité du schéma du graphe bande en parallèle devrait permettre d'améliorer sensiblement la qualité des renumérotations calculées.

Une version parallèle de l'algorithme d'extraction du graphe bande a donc été développée. Couplée à un algorithme de centralisation du graphe, utilisant la primitive `dgraphGather` présentée en section 1.6.3, elle permet d'obtenir les résultats présentés ci-dessous.

3.4.3.1 Évaluation de la qualité des résultats produits

Les résultats visibles dans le tableau 3.3 ont été obtenus par l'utilisation d'un graphe bande de largeur trois associé à un raffinement multi-séquentiel utilisant l'algorithme de Fiduccia-Mattheyses. PT-Scotch est exécuté sur la machine IBM P575 du pôle M3PEC décrite en section 1.6.4.

Graphe	Nombre de processeurs					
	2	4	8	16	32	64
altr4	5.54e+8	3.99e+8	3.95e+8	4.01e+8	3.98e+8	4.16e+8
audikw1	6.02e+12	5.89e+12	5.41e+12	5.66e+12	5.75e+12	5.49e+12
bmw32	3.16e+10	3.72e+10	3.52e+10	3.93e+10	4.04e+10	3.71e+10
conesphere1m	3.17e+12	1.90e+12	2.00e+12	1.98e+12	1.88e+12	1.91e+12
coupole8000	8.67e+12	8.54e+12	8.31e+12	8.14e+12	8.16e+12	8.19e+12
oilpan	3.58e+9	3.36e+9	3.43e+9	3.85e+9	3.93e+9	3.66e+9
thread	4.28e+10	4.38e+10	4.06e+10	4.24e+10	4.07e+10	4.06e+10
bcsstk32	1.54e+10	1.60e+9	1.65e+9	1.68e+9	1.48e+9	1.74e+9
Qimonda07	†	†	5.80e+10	6.38e+10	6.94e+10	7.70e+10
G3_circuit	6.54e+10	7.94e+10	7.06e+10	6.50e+10	8.11e+10	6.22e+10
af_shell10	4.69e+11	4.73e+11	4.60e+11	4.47e+11	4.85e+10	4.82e+10
bone010	4.78e+12	4.51e+12	4.75e+12	4.80e+12	5.17e+12	5.34e+15
cage14	2.19e+15	2.38e+15	2.48e+15	2.48e+15	2.33e+15	2.44e+15
cage15	4.58e+16	5.01e+16	4.64e+16	4.95e+16	4.58e+16	4.50e+16
thermal2	1.90e+10	2.08e+10	1.88e+10	2.05e+10	2.16e+10	1.59e+10
bgrm	2.70e+13	2.55e+13	2.65e+13	2.88e+13	2.86e+13	2.87e+13
23millions	1.45e+14	2.91e+14	3.99e+14	2.71e+14	1.94e+14	2.45e+14

Tab. 3.3: Mesure de la qualité des renumérotations issues de PT-Scotch, en fonction du nombre de processeurs utilisés. Le raffinement consiste en une application multi-séquentielle de l'algorithme de Fiduccia-Mattheyses sur le graphe bande. Le repliement et la duplication sont activés dans la phase de contraction du schéma multi-niveaux, lorsque le nombre moyen de sommets par processeurs devient inférieur à cent. Le symbole † indique que le calcul n'a pas pu être conduit à son terme, faute d'espace mémoire suffisant.

On remarque que pour tous les types de graphes la qualité est maintenue lorsque le nombre de processeurs augmente. De plus, si on la compare aux données du tableau 3.2, on constate que celle-ci est du même ordre que celle obtenue avec la version séquentielle de Scotch.

En outre, sur plusieurs graphes, tels `audikw1`, `coupole8000` ou encore `thread`, on observe une diminution du coût associé à la renumérotation à mesure que le nombre de processeurs augmente. Cette progression peut s'expliquer par la multi-séquentialité utilisée avec l'algorithme de Fiduccia-Mattheyses sur le bande graphe qui, en augmentant l'espace de recherche, permet de trouver des séparateurs de meilleure qualité.

Cependant, cette augmentation de la qualité n'est pas systématique. Cela peut s'expliquer d'une part par le fait que la qualité de la contraction parallèle est a priori inférieure à celle

séquentielle, d'autre part par le fait que l'utilisation du graphe bande peut quelquefois trop limiter l'espace de recherche disponible pour l'heuristique de raffinement. Néanmoins, il faut noter que, dans la plupart des cas, ces effets négatifs sont contre-balancés par la multiplicité des exécutions des raffinements centralisés.

Sur deux graphes, `Qimonda07` et `G3_circuit`, la qualité de la renumérotation diminue de manière significative. L'observation des points communs à ces deux graphes conduit à remarquer qu'ils ont tous deux un degré moyen très faible, et qu'ils sont sont aussi assez bien adaptés à des renumérotations de type degré minimum. Il est donc possible que ce soit l'utilisation de la méthode des dissections emboîtées qui soit mise en cause pour ces deux graphes.

Le tableau 3.4 permet de comparer nos résultats à ceux obtenus avec le principal renumeroteur parallèle de graphes disponible : ParMeTiS. On remarque que, contrairement à ce qui se produisait avec Scotch, la qualité de la renumérotation produite par ParMeTiS est très sensible à l'augmentation du nombre de processeurs, on peut notamment avoir une explosion du coût de factorisation pour les graphes `audikw1`, `thread` ou encore `cage15`. Force est de constater que toutes les expérimentations avec 64 processeurs sont favorables à PT-Scotch.

Graphe	**Nombre de processeurs**					
	2	4	8	16	32	64
`altr4`	4.20e+8	4.49e+8	4.46e+8	4.64e+8	5.03e+8	5.16e+8
`audikw1`	†	†	7.78e+12	8.88e+12	8.91e+12	1.07e+13
`bmw32`	3.22e+10	4.09e+10	5.11e+10	5.61e+10	5.74e+10	6.31e+10
`conesphere1m`	2.20e+12	2.46e+12	2.78e+12	2.96e+12	2.99e+12	3.29e+12
`coupole8000`	†	†	8.17e+12	8.26e+12	8.58e+12	8.71e+12
`oilpan`	3.58e+9	3.36e+9	3.43e+9	3.85e+9	3.93e+9	3.66e+9
`thread`	3.98e+10	6.60e+10	1.03e+11	1.24e+11	1.53e+11	‡
`bcsstk32`	1.55e+10	1.62e+9	3.09e+9	4.11e+9	5.85e+9	4.01e+9
`Qimonda07`	‡	‡	‡	‡	‡	‡
`G3_circuit`	6.29e+10	8.00e+10	7.06e+10	7.54e+10	8.57e+10	8.58e+10
`af_shell10`	4.11e+11	4.48e+11	4.60e+11	5.36e+11	5.80e+10	5.67e+10
`bone010`	4.32e+12	4.73e+12	4.75e+12	6.90e+12	7.92e+12	8.63e+15
`cage14`	2.15e+15	3.70e+15	2.48e+15	3.31e+15	3.41e+15	3.44e+15
`cage15`	4.47e+16	6.64e+16	4.64e+16	7.36e+16	7.03e+16	6.64e+16
`thermal2`	1.57e+10	1.74e+10	1.88e+10	2.18e+10	2.16e+10	2.30e+10
`bgrm`	†	‡	‡	‡	‡	‡
`23Millions`	‡	‡	‡	‡	‡	‡

Tab. 3.4: Mesure de la qualité des renumérotations issues de ParMeTiS, en fonction du nombre de processeurs utilisés. Le symbole † indique que le calcul n'a pas pu être conduit à son terme, faute d'un espace mémoire suffisant. Le symbole ‡ signifie que le calcul a été interrompu suite à la production d'un message d'erreur par la bibliothèque MPI.

Toujours au niveau de l'évaluation qualitive des résultats, un autre phénomène, non visible dans ces tableaux, peut être observé avec l'augmentation du nombre de processeurs lors de l'utilisation de PT-Scotch : il s'agit de l'amélioration de l'équilibre de l'arbre d'élimination. C'est en fait une conséquence directe de l'amélioration probable des différents bipartionnements, qui n'affecte pas forcément le nombre d'opérations nécessaires à la factorisation séquentielle des matrices. Cependant, le fait de fournir des sous-arbres bien équilibrés permet d'extraire plus

facilement du parallélisme pour les solveurs parallèles comme PaStiX [38] ou MUMPS [2].

Ce phénomène contribue à expliquer la perte de qualité constatée sur le graphe `Qimonda07` dont les arbres d'élimination associés à ses meilleures renumérotations sont très déséquilibrés, l'équilibre forcé par notre schéma de renumérotation induisant un surcoût de calcul en séquentiel.

3.4.3.2 Étude de la scalabilité du processus de renumérotation parallèle

La qualité des résultats semblant être au rendez-vous, il est nécessaire de s'intéresser également aux temps de calcul. Ceux-ci sont présentés dans le tableau 3.5.

Graphe	Nombre de processeurs					
	2	4	8	16	32	64
altr4	4.30	0.23	0.23	0.29	3.17	4.30
audikw1	24.46	55.94	44.45	31.58	25.71	24.46
bmw32	6.89	5.47	4.05	2.91	4.23	6.89
conesphere1m	13.62	14.45	12.47	9.40	9.90	13.62
coupole8000	114.41	68.81	44.98	29.89	22.42	15.82
oilpan	2.10	1.34	0.92	0.80	2.42	5.00
thread	3.66	2.42	1.99	1.87	3.04	5.66
bcsstk32	1.15	0.85	0.83	0.90	3.07	5.11
Qimonda07	†	†	34.85	20.09	17.05	13.70
G3_circuit	18.76	13.52	9.85	9.79	9.39	11.04
af_shell10	51.28	29.83	20.46	15.58	7.43	9.50
bone010	65.21	44.52	31.39	24.45	17.74	17.81
cage14	130.85	122.74	66.48	52.89	62.35	63.7
cage15	540.46	427.38	371.70	340.78	351.38	226.10
thermal2	13.71	9.66	7.78	7.09	4.71	9.10
brgm	276.9	167.26	97.69	61.65	42.85	41.00
23Millions	666	416	288	209	140	103.37

Tab. 3.5: Mesure du temps (en s) de calcul, sur la machine IBM P575 de l'université de Bordeaux I, des renumérotations de différents graphes par PT-Scotch, en fonction du nombre de processeurs utilisés. La méthode utilisée est celle décrite dans la légende du tableau 3.3. Le symbole † indique que le calcul n'a pas pu être conduit à son terme, faute d'espace mémoire suffisant.

Nous constatons que sur les plus gros cas, comme `Qimonda07`, `cage15` ou `brgm`, la scalabilité en temps est relativement bonne. Pour les autres graphes, on constate l'existence d'un plateau à partir d'un certain nombre de processeurs, les étapes de communications devenant trop importantes par rapport aux phases de calcul.

Le tableau 3.6 montre les temps obtenus avec ParMeTiS. On constate que ce dernier est très scalable en temps et effectue les renumérotations plus rapidement que PT-Scotch. Cependant, l'ordre de grandeur des temps de calcul reste comparable entre les deux logiciels et les précédents résultats en terme de qualité de renumérotation sont aussi à considérer. La meilleure scalabilité de ParMeTiS s'explique aussi par le fait que le calcul séquentiel équivalent est toujours identique, alors qu'à l'heure actuelle, la méthode de raffinement utilisée par PT-Scotch est séquentielle. Si l'étape d'expansion n'est pas scalable en temps ni en mémoire, le reste du processus de renumérotation et de partitionnement l'est, ce qui permet néanmoins de conserver

Graphe	**Nombre de processeurs**					
	2	4	8	16	32	64
altr4	0.31	0.20	0.13	0.11	0.13	0.33
audikw1	32.64	23.09	17.13	9.80	5.67	3.82
bmw32	3.37	2.29	1.51	0.91	0.68	1.08
conesphere1m	22.35	11.98	6.76	3.89	2.28	1.87
coupole8000	63.39	37.75	20.01	10.81	5.88	3.14
oilpan	0.78	0.53	0.35	0.26	0.24	0.27
thread	1.24	1.05	0.68	0.51	0.40	‡
bcsstk32	0.62	0.40	0.29	0.23	0.22	0.24
Qimonda07	‡	‡	‡	‡	‡	‡
G3_circuit	15.83	8.87	5.00	2.93	1.92	1.80
af_shell10	15.14	9.71	5.68	3.49	2.04	1.80
bone010	24.90	15.52	9.42	5.81	3.59	8.80
cage14	46.41	28.00	16.66	9.95	6.53	5.75
cage15	195.93	117.77	65.15	40.30	22.56	17.83
thermal2	13.55	7.55	4.15	2.42	2.12	2.50
brgm	†	‡	‡	‡	‡	‡
23Millions	‡	‡	‡	‡	‡	‡

TAB. 3.6: Mesure du temps de calcul (en s), sur la machine IBM P575 de l'université de Bordeaux I, des renumérotations de différents graphes par PARMETIS, en fonction du nombre de processeurs utilisés. Le symbole † indique que le calcul n'a pas pu être conduit à son terme, faute d'un espace mémoire suffisant. Le symbole ‡ indique que le programme a terminé prématurément, à cause d'une erreur MPI.

la scalabilité en temps pour les gros graphes.

En outre, la scalabilité mémoire est aussi présente, comme le montre le tableau 3.7. Effectivement, la présence du repliement et de la duplication n'intervenant finalement que pour des graphes contractés d'assez petite taille par rapport à celle du graphe original ne semble pas être très coûteuse en terme de mémoire. De plus, et cela a été aussi vérifié expérimentalement, son utilisation tend à mieux répartir la charge mémoire sur les différents processeurs, tout en contribuant à diminuer la taille des données uniquement nécessaires à la distribution comme les sommets fantômes. Ce déséquilibre mémoire peut par exemple être responsable de l'échec de l'exécution de PT-Scotch sur le graphe `cage15`, le gestionnaire de tâches de la machine utilisée ne permettant que de définir par processeur, et non globalement, l'occupation mémoire maximale autorisée.

On peut néanmoins relever que sur les graphes ayant un degré moyen très élevé comme `thread` la scalabilité mémoire atteinte ralentit assez nettement à partir de 16 processeurs. Ce fait s'explique conjointement par la faible taille de ce graphe et par l'importance des interfaces entre les différents processeurs, la probabilité d'avoir un voisin distant pour un sommet étant pratiquement proportionnelle à son degré.[9]

Graphe	**Nombre de processeurs**					
	2	4	8	16	32	64
`audikw1`	1.42+09	9.23+08	5.38+08	3.06+08	1.75+08	9.91+07
`conesphere1m`	3.91+08	2.31+08	1.36+08	7.70+07	4.37+07	2.48+07
`coupole8000`	1.67+09	1.02+09	5.80+08	3.31+08	1.88+08	1.02e+8
`oilpan`	4.93+07	3.41+07	2.04+07	1.21+07	7.03+06	4.10e+6
`thread`	7.31+07	4.76+07	2.77+07	1.85+07	1.43+07	1.15e+7

Tab. 3.7: Consommation mémoire moyenne (en octets) pour un processus utilisé pour la renumérotation de différents graphes par PT-Scotch, avec la même méthode de renumérotation que précédemment, en fonction du nombre de processeurs utilisés.

Il en ressort que nous disposons avec PT-Scotch d'un outil parallèle efficace pour la renumérotation de matrices creuses : la qualité des résultats est comparable à celle obtenue avec les renuméroteurs séquentiels, et est obtenue avec une scalabilité en temps, ainsi qu'en mémoire, raisonnable.

3.4.4 Résultats obtenus avec le repliement et la duplication

Nous allons maintenant nous intéresser aux conséquences de l'utilisation de la technique de repliement et duplication lors de la contraction du schéma multi-niveaux et de sa prise en charge durant l'étape d'expansion.

Le tableau 3.8 montre les temps de calculs nécessaires à la renumérotation du graphe `bgrm` avec PT-Scotch. On constate que la version utilisant la duplication en plus du repliement est la plus rapide, et que la version exploitant un repliement seul est la plus lente, la version classique se situant entre les deux.

Le coût temporel de l'opération de repliement seul ou de repliement avec duplication étant a priori identiques, la différence constatée est selon toute vraissemblance due à la qualité des

[9] Elle est proportionnelle au degré dans le cas d'un graphe distribué uniformément.

Stratégie	**Nombre de processeurs**					
	2	4	8	16	32	64
Pas de repliement	260.31	156.65	102.37	71.19	45.33	†
Repliement seul	284.62	185.00	122	92.86	†	†
Repliement et duplication	276.91	167.26	97.69	61.65	42.85	41.00

TAB. 3.8: Résultats en temps (en s) de la renumérotation du graphe `brgm` avec PT-SCOTCH. Les temps sont exprimés en secondes et ont été mesurés sur la machine *decrypthon* du pôle M3PEC. La stratégie avec repliement effectue un repliement lorsqu'il y a moins de 100 sommets par processeur, la stratégie avec repliement et duplication utilise le même seuil. Le symbole † indique que le calcul n'a pas pu être conduit à son terme, faute d'un espace mémoire suffisant par processeur.

Stratégie	**Nombre de processeurs**					
	2	4	8	16	32	64
Pas de repliement	545.40	404.51	304.64	280.20	†	†
Repliement seul	525.33	410.33	320.97	302.18	†	†
Repliement et duplication	515.70	396.69	308.25	213.71	167.54	†

TAB. 3.9: Résultats en temps de la renumérotation du graphe `brgm` permuté aléatoirement avec PT-SCOTCH. Les conditions expérimentales sont les mêmes que celles décrites pour le tableau 3.8. Le symbole † indique que le calcul n'a pas pu être conduit à son terme, faute d'un espace mémoire suffisant par processeur.

résultats produits par les premières bipartitions récursives. En effet, dans le cadre des dissections emboîtées, un mauvais choix dans le partitionnement des premiers niveaux peut avoir de lourdes conséquences au niveau des temps et de la complexité des calculs à conduire aux niveaux suivants. Cette hypothèse semble être confirmée à la vue des résultats relatifs à la qualité de renumérotation disponibles dans le tableau 3.10.

Le fait que la méthode avec repliement seul soit significativement plus lente que la méthode classique sans aucun repliement est par contre plus surprenant. Certes, il existe un surcoût dû au temps de redistribution, mais la meilleure localisation des données devrait permettre d'en recouvrir une partie. Cependant, il faut préciser que la taille importante du graphe `brgm` permet de conserver une assez bonne localité mémoire si le graphe initial est correctement distribué ; le gain de temps espéré pour le repliement doit donc être très faible. En fait, il est possible que surcoût en temps ne soit pas uniquement dû à la construction du graphe replié mais soit en partie imputable à une plus mauvaise qualité des premiers bipartitionnements.

On remarque que les temps de renumérotation sur le même graphe `brgm` mais distribué aléatoirement sont beaucoup plus importants que pour le graphe initial, à cause de communications beaucoup plus nombreuses. La méthode utilisant la duplication est toujours la plus efficace et l'écart avec les autres méthodes s'accroît, preuve que la localisation des données est bien fondamentale pour notre problème.

Les échecs à l'exécution observés lorsque le nombre de processeurs devient grand s'expliquent par une mauvaise répartition de la charge mémoire sur les processeurs et par le fait que la machine employée pour nos tests ne peut effectuer qu'un contrôle par processus de la mémoire maximale autorisée. L'augmentation du nombre de processeurs rend plus sensible les écarts de besoins mémoire pour un processeur par rapport à la valeur optimale qui est la moyenne des tailles mémoire. Il est donc plus fréquent qu'un processeur devienne trop chargé par rapport

aux autres et dépasse la quantité de mémoire à laquelle il est limité par la machine, avec pour conséquence un abandon du calcul. Nous avons constaté que l'utilisation du repliement et de la duplication procure une répartition plus homogène de la charge mémoire sur l'ensemble des processeurs, ce qui permet en pratique de mener à terme les calculs sur plus de processeurs qu'avec les deux autres méthodes, alors qu'au niveau de la mémoire totale elle est pourtant la plus coûteuse des trois.

Stratégie	**Nombre de processeurs**					
	2	4	8	16	32	64
Pas de repliement	2.53e+13	2.51e+13	2.66e+13	2.83e+13	2.81e+13	†
Repliement seul	2.54e+13	3.06e+13	3.12e+13	3.14e+13	†	†
Repliement/duplication	2.70e+13	2.55e+13	2.65e+13	2.88e+13	2.86e+13	2.87e+13

TAB. 3.10: Coûts de factorisation induit par la renumérotation du graphe `brgm` avec PT-SCOTCH. Le seuils d'application des repliements est de 100 sommets par processeur. Le symbole † indique que le calcul n'a pas pu être conduit à son terme, faute d'un espace mémoire suffisant par processeur.

Le tableau 3.10 fournit les résultats relatifs à la qualité de renumérotation obtenue avec PT-SCOTCH sur le graphe `brgm` avec sa numérotation initiale, les résultats obtenus sur le graphe permuté aléatoirement étant identiques.

Deux méthodes, celle sans repliement et celle avec repliement et duplication, donnent les mêmes résultats, ceux obtenus avec la stratégie utilisant un repliement seul étant signicativement inférieurs. Ce fait confirme l'hypothèse que nous avions formulée sur l'origine du surcoût temporel de la méthode avec repliement seul, à savoir que celui-ci provenait certainement d'une baisse de la qualité des dissections dans les niveaux.

Une explication de cette perte de qualité est que le schéma avec repliement seul exploite moins la multi-séquentialité lors du raffinement, comme il se prive à chaque repliement de la moitié des processeurs. Les deux approches, la duplication ou l'absence de repliement, exploitent toujours les p instances de l'algorithme de Fiduccia-Mattheyses qui sont exécutées à chaque niveau de contraction. Or nous avons déjà évoqué l'intérêt d'effectuer plusieurs fois cette heuristique, notamment dans les sections 1.2.2.1 et 3.4.2.

Il faut aussi noter que la version avec duplication semble fournir des renumérotations dont les arbres d'élimination sont mieux équilibrés que ceux obtenus avec les autres méthodes, et ceci est d'autant plus visible que le nombre de processeurs est grand. Il reste néanmoins à vérifier que ce parallélisme puisse être effectivement utilisé dans les solveurs parallèles.

La version actuelle de PT-SCOTCH, en utilisant par défaut un schéma multi-niveaux avec repliement et duplication, associé à un raffinement multi-séquentiel utilisant le graphe bande, permet ainsi de compenser la perte de qualité pouvant être induite par la contraction parallèle, et fournit des résultats qualitativement comparables à ceux obtenus par la version séquentielle de SCOTCH.

Cependant, cette technique ne peut être appliquée que si le graphe bande peut être centralisé sur un processeur, c'est-à-dire que sa taille reste inférieure à la mémoire disponible sur un processeur. Dans le cas de très grands graphes ou avec certains graphes pouvant avoir de grands séparateurs, cette méthode ne sera donc pas applicable en l'état. C'est pourquoi, devant les échecs des précédentes parallélisations de l'heuristique de Fiduccia-Mattheyses, nous avons choisi d'étudier, dans le chapitre suivant, le comportement des algorithmes génétiques utilisés pour le raffinement dans la phase d'expansion du schéma multi-niveaux.

Chapitre 4

Une approche génétique pour le raffinement

Sommaire

Les algorithmes génétiques sont des méta-heuristiques de la classe des algorithmes évolutionnistes brièvement présentés à la page 16. Un excellent résumé sur le fonctionnement des algorithmes génétiques peut être trouvé dans [85].

4.1 Quelques notions sur les algorithmes génétiques

Le principe de base des algorithmes génétiques consiste à explorer dynamiquement l'espace des solutions au moyen d'algorithmes possédant deux caractéristiques essentielles :

– l'innovation, causée par les mutations ;
– la sélection naturelle.

Pour effectuer la sélection naturelle, il est nécessaire de pouvoir comparer les individus entre eux, au moins deux à deux. Classiquement, on ajoute aussi à ces deux éléments un opérateur de reproduction entre plusieurs individus afin d'essayer de cumuler leurs avantages au sein des les individus engendrés : on le nomme aussi opérateur de conservation.

La structure d'un algorithme génétique est donnée par l'algorithme 10.

Algorithme 10 Schéma de principe d'un algorithme génétique.

```
1:  Fonction AG
2:      Initialisation de la population
3:      Tant que (pas de convergence ou nombre maximal de générations non atteint) Faire
4:          Évaluer les individus
5:          Sélectionner les individus
6:          Si (une solution de qualité suffisante a été trouvée) alors
7:              Sortir
8:          Fin de Si
9:          Construction de la nouvelle génération
10:     Fin de Tant que
11: Fin de Fonction
```

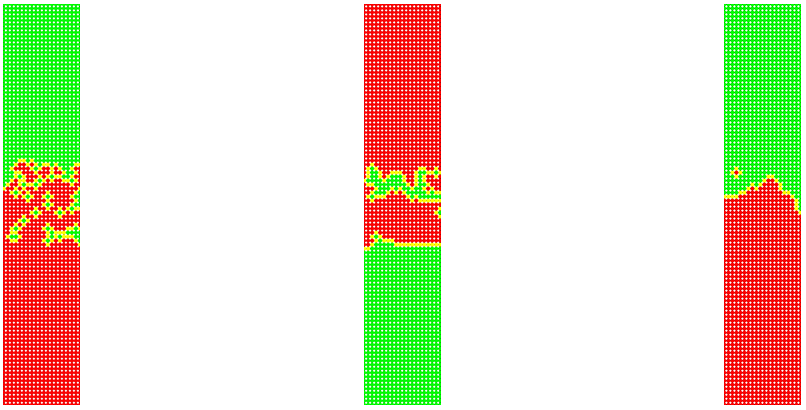
Le cheminement nécessaire pour réaliser un algorithme génétique peut être résumé ainsi :

– choix de la représentation des individus ;
– choix de la population ;
– choix des opérateurs d'innovation, qui vont permettre de balayer l'espace de recherche ;
– choix des opérateurs de conservation, qui vont permettre de s'orienter vers les zones a priori intéressantes de l'espace de recherche ;
– choix de l'opérateur de comparaison des qualités des individus ;
– choix de l'opérateur de sélection.

Les opérateurs d'innovation et de conservation sont utilisés pour créer des individus qui vont peupler la nouvelle génération, en ligne 9 de l'algorithme. L'opérateur de sélection est quant à lui le garant du fonctionnement de la méthode, en indiquant dans quelle direction il faut parcourir l'espace des solutions. C'est cet opérateur qui différencie les algorithmes génétiques d'un parcours purement aléatoire de l'espace des solutions. Nous pouvons constater qu'il dépend aussi fortement de l'opérateur de comparaison, qui permet de classer les individus, la plupart du temps en utilisant une quantification de leur qualité. Cette évaluation numérique de la qualité n'est pas forcément indispensable, de la même manière que nous pouvons classer des objets selon leur taille sans les mesurer, juste en les comparant.

La figure 4.1 montre les meilleurs individus obtenus lors du déroulement d'un algorithme génétique très simple faisant du partitionnement de graphes. L'algorithme semble converger

vers l'individu idéal, mais la vitesse de convergence est très lente dans ce cas. Il est important de préciser que cet exemple a été obtenu avec un moteur très simple d'algorithme génétique, ayant pour population initiale uniquement des individus aléatoires, et travaillant avec des contraintes très lâches. L'intérêt de ces images est simplement de donner une idée de la convergence d'un algorithme génétique. L'algorithme implanté dans PT-Scotch est tout autre et les choix qui ont été effectués vont être présentés dans la prochaine section.



(a) Meilleur individu de la première génération. On observe la taille importante du séparateur, ainsi que l'irrégularité de son contour.

(b) Meilleur individu de la centième génération. La taille du séparateur a diminué, les contours deviennent plus réguliers.

(c) Meilleur individu de la millième génération. Les parties sont maintenant pratiquement connexes et la taille du séparateur a beaucoup diminué.

Fig. 4.1: Exemple de résultats obtenus sur une grille 2D avec un partitionnement utilisant un algorithme génétique seul, sans multi-niveaux ni heuristiques d'optimisation locale. Le résultat obtenu n'est pas celui de l'algorithme utilisé dans PT-Scotch ; il s'agit juste d'une mise en évidence d'une convergence, même avec des critères lâches.

4.2 Algorithmes génétiques dans PT-Scotch

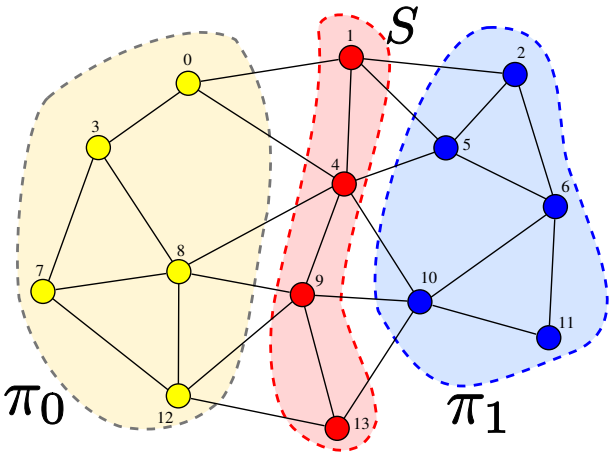
4.2.1 Principe de base

4.2.1.1 Définition de la structure des individus

Les algorithmes génétiques utilisés dans PT-Scotch sont des algorithmes faisant intervenir des individus mono-chromosomiques. L'unique chromosome représentant chaque individu correspond à la partition du graphe : c'est en fait un simple tableau indicé par les sommets du graphe, qui leur associe la partie dans laquelle ils se trouvent : 0 ou 1 pour chacune des deux parties et 2 pour le séparateur. La figure 4.2 permet de visualiser un individu et sa représentation dans le cadre de notre algorithme génétique.

La taille de nos individus est donc constante, ce qui permet de rester dans le cadre classique [39] des utilisations d'algorithmes génétiques.

Concernant la représentativité de ce codage, nous paraissons dépendre de la numérotation du graphe, mais nous verrons plus loin que le déroulement de l'algorithme n'y est pas sensible. Bien que la représentation chromosomique d'un individu change lorsque la numérotation du



(a) Partition correspondant à un individu.

0	2	1	0	2	1	1	0	0	2	1	1	0	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13

(b) Chromosome représentant cet individu : tableau indicé par les numéros des sommets et contenant trois symboles : 0, 1 et 2, selon que le sommet considéré est situé dans la partie π_0, π_1 ou S.

FIG. 4.2: Représentation d'un individu dans le cadre de nos algorithmes génétiques.

graphe change, les opérateurs d'évolution sont définis de façon à ce que leur comportement soit invariant par rapport à la numérotation du graphe.

Nous pouvons aussi constater que ce codage est suffisant, dans la mesure où nous connaissons la topologie du graphe associé et que le tableau représentant l'unique chromosome d'un individu permet de caractériser la partition associée à l'individu. Il faut, en revanche, noter que la donnée seule du codage de l'individu ne permet pas de dire si la partition correspondante est valide. La validité d'un individu nécessite d'analyser la validité de la partition représentée par le codage du chromosome.

Dans la suite, nous identifierons l'individu à son chromosome et nous parlerons donc indifféremment d'un individu ou de son chromosome associé.

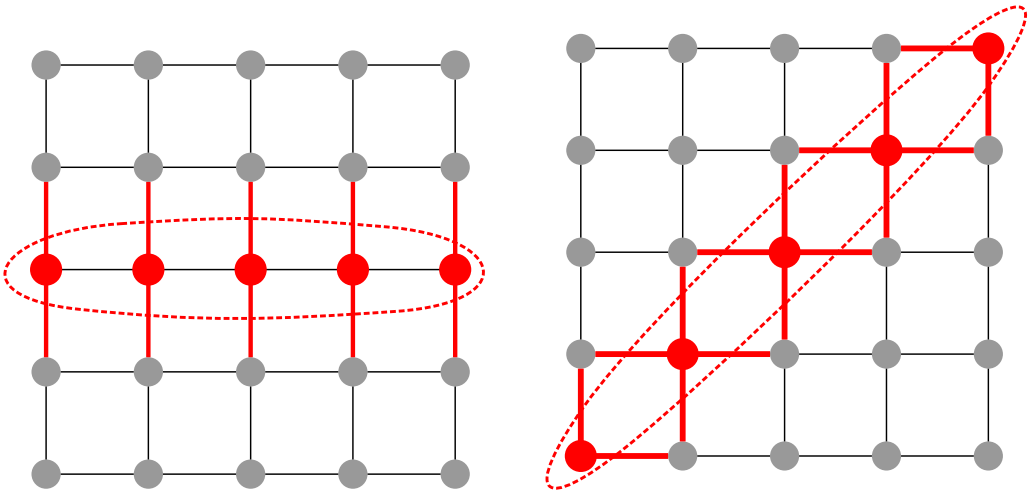
4.2.1.2 Évaluation de la qualité des individus : fonction d'adéquation

L'évaluation de la qualité des individus est attribuée à une fonction d'adéquation (en anglais, « *fitness* ») qui leur attribue une valeur dans l'intervalle $[0;1]$. Le principal problème est que, dans le cas du partitionnement de graphes, l'individu de qualité optimale, correspondant à une valeur 1, n'est pas connu, et qu'il est donc difficile de pouvoir donner une note absolue à la qualité d'un individu. En outre, une mauvaise répartition des valeurs sur l'intervalle $[0;1]$ peut rendre difficile la sélection, en fonction de la stratégie choisie, des individus pour la reproduction. La solution que nous avons retenue consiste donc à noter nos individus par rapport à notre meilleur individu, pour que celui-ci ait une valeur d'adéquation proche de un [31].

Notre fonction d'adéquation peut prendre en compte plusieurs critères, par ordre décroissant d'importance :

1. le nombre de sommets du séparateur, qui est le critère le plus important ;

2. le poids des arêtes de liaison entre les sommets du séparateur et les sommets des autres parties. Ce critère permet de s'assurer de la géométrie du séparateur. Un exemple de l'utilité de ce facteur est montré en figure 4.3 ;
3. dans une moindre mesure, nous pouvons aussi faire intervenir le nombre de composantes connexes des parties, l'idéal étant une composante par partie dans le cas d'un graphe connexe.



(a) Premier séparateur de taille 5. Le nombre d'arêtes qui lui sont adjacentes est 10.

(b) Second séparateur de taille 5. Le nombre d'arêtes qui lui sont adjacentes est 16. Sa qualité est évaluée comme inférieure à celle du séparateur précédent.

Fig. 4.3: Évaluation de la forme du séparateur à l'aide des arêtes qui lui sont incidentes ; exemple avec deux séparateurs à cinq sommets sur une grille 5×5. Chacun des deux séparateurs définit une partition parfaitement équilibrée, la seule différence résidant en leur forme. Les arêtes en gras sont les arêtes incidentes aux sommets du séparateur.

Tous ces critères sont pris en compte en faisant entre eux une moyenne arithmétique, pondérée selon leur importance. Cependant, l'équilibre du poids des parties est aussi essentiel pour qualifier la qualité d'une partition. Cette moyenne est donc ensuite multipliée par un coefficient dépendant de l'équilibre du partitionnement. La valeur du coefficient varie selon l'écart de poids des parties, comme illustré par la figure 4.4. On voit que la courbe comporte un plateau, centré sur 0 et de très faible pente dans l'intervalle $[0; 2]$, qui correspond à la zone de tolérance pour le déséquilibre que nous avons fixé pour nos algorithmes, identique à celui utilisé par d'autres heuristiques comme Fiduccia-Mattheyses. La valeur de la fonction en dehors de cette zone décroît très rapidement à mesure de l'éloignement de la zone cible d'équilibre, tout en maintenant une valeur non nulle afin de permettre à l'algorithme de pouvoir quantifier la progression de tous les individus, même ceux qui sont a priori insatisfaisants pour notre problème mais qui aident à maintenir une diversité génétique suffisante. Il faut aussi noter l'importance du plateau par rapport au choix d'une simple gaussienne : ce plateau permet de ne pas trop désavantager les individus ne présentant pas un équilibre parfait.

Les coefficients de pondération des différents critères ont été choisis expérimentalement, et leurs valeurs seront donc fournies dans la section 4.4.1 traitant des premières expérimentations.

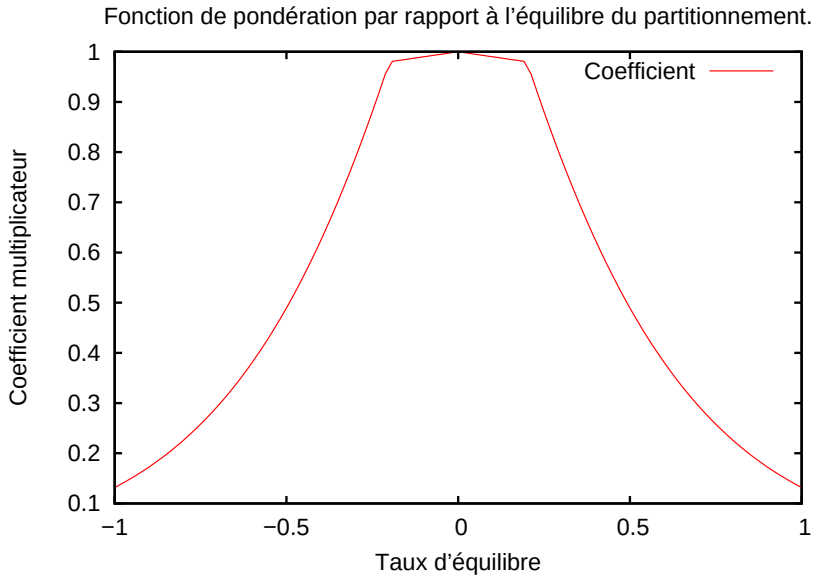


FIG. 4.4: Coefficient multiplicateur affecté à l'équilibre des partitions. Les individus qui sont en dehors de l'intervalle autorisé (ici $[-0.2; 0.2]$) sont pénalisés. Pour les autres, ceux qui sont le plus près possible de l'équilibre sont récompensés.

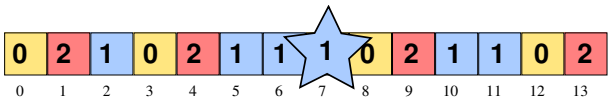
4.2.1.3 Opérateur d'innovation : la mutation

Cet opérateur est extrêmement simple et permet d'augmenter l'entropie du processus d'exploration de l'espace des solutions, c'est-à-dire de favoriser la diversité génétique en permettant d'obtenir des individus dont l'accès était impossible par une combinaison des individus de la population de base. Cet opérateur sert donc à créer de la diversité au sein de la population. Néanmoins, il est couramment admis que sa fréquence d'intervention doit être faible et diminuer quand le nombre de générations écoulées augmente, ceci afin de ne pas perturber la convergence de l'algorithme génétique.

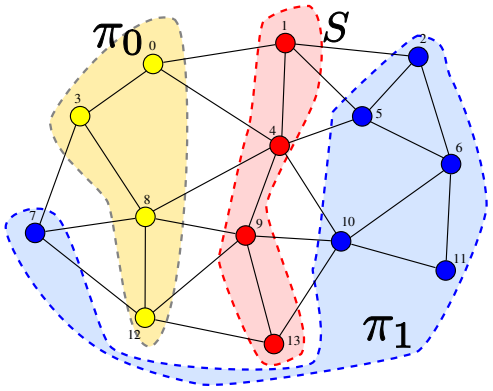
Pour nos individus, une mutation peut être vue comme un changement spontané de partie d'un de leurs sommets. Ainsi, lorsqu'une mutation doit avoir lieu sur un individu, un sommet est choisi aléatoirement et est transféré de sa partie π_i à l'autre partie. Tous les sommets sont susceptibles de pouvoir être changés : dans le cas d'un sommet appartenant initialement au séparateur, son affectation à l'une des deux parties s'effectue aussi de manière aléatoire.

La figure 4.5a illustre comment une mutation survient, en se basant sur l'individu de la figure 4.2 de la page 92. Nous pouvons remarquer, en figure 4.5b, que cette définition de la mutation ne produit pas toujours des individus correspondant à un partitionnement licite du graphe. Nous devons donc introduire également un opérateur de réparation, qui sera décrit plus loin, et qui permet d'obtenir des individus mutants valides, comme sur la figure 4.5c. On remarque que la perturbation due à la mutation est beaucoup plus importante que la seule portée individuelle de la modification aléatoire initiale ne le laissait supposer.

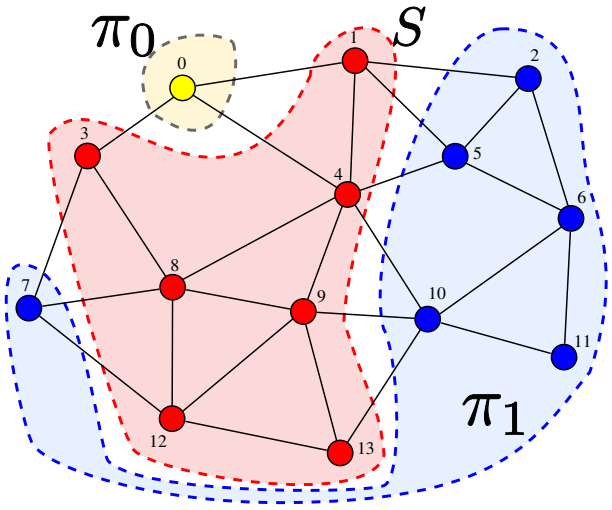
Dans notre implantation, les mutations interviennent lors du processus de reproduction mais leur probabilité d'apparition est décroissante par rapport à l'évolution dans le temps. L'exploration de l'espace de recherche est donc plutôt effectuée grâce à l'opérateur de conservation.



(a) Mutation sur le chromosome à la position 7. Le sommet passe de la partie π_0 à π_1, cette modification aléatoire étant symbolisée par l'étoile.



(b) Visualisation de l'individu mutant en terme de partition sur le graphe. Le partitionnement est illicite puisque le sommet 7 de la partition π_1 est voisin des sommets 3, 8 et 12 de la partition π_0.



(c) Visualisation du mutant finalement obtenu après application de l'opérateur de réparation.

Fig. 4.5: Exemple de mutation avec réparation sur l'individu décrit précédemment.

4.2.1.4 Opérateur de conservation : crossover

La reproduction consiste essentiellement à un mélange entre deux individus par un « *crossover* », complété par des mutations aléatoires.

Le principe du crossover est simple. Il consiste à mixer deux individus, les « parents », par collage, c'est-à-dire que l'on choisit un sommet qui sera un pivot, tel que chaque enfant recevra une partie seulement des gênes de chacun de ses parents, celle située avant ou après le point de crossover. Un tel crossover est appelé crossover à un seul point et peut être visualisé en figure 4.6.

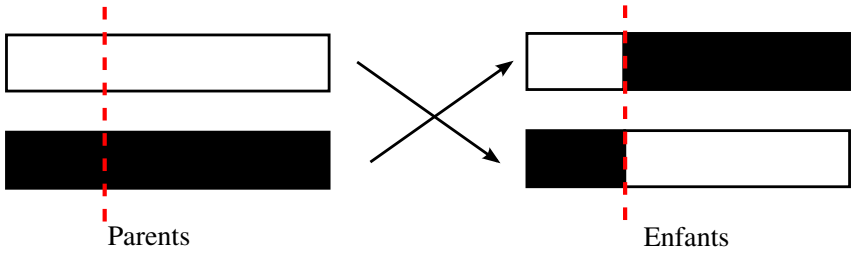


FIG. 4.6: Principe du crossover à un seul point. Le point d'échange est déterminé aléatoirement et est ici représenté par la ligne en tirets.

Nous pouvons aussi réaliser des crossover à plusieurs points, dans le but d'augmenter le mélange, mais des résultats [24, 72] montrent que le nombre de points n'est finalement pas un paramètre important dans la plupart des problèmes. Aucune différence significative n'ayant été observée au cours des essais que nous avons effectués, nous avons utilisé par la suite un crossover à un seul point.

On peut néanmoins remarquer que, dans notre cas, un simple collage tel que décrit précédemment ne suffit pas pour engendrer un individu valide : en effet, la partition obtenue n'est pas forcément valide au sens où des sommets de parties différentes peuvent être adjacents. Comme pour la mutation, il est donc nécessaire d'avoir ici encore recours à une phase de *réparation* afin que tous les individus produits soient valides. L'algorithme que nous exploitons pour valider nos individus comporte deux phases. La première phase consiste à ajouter les sommets nécessaires au séparateur afin d'avoir une partition valide. Les sommets sont parcourus aléatoirement, et leurs voisins sont placés dans le séparateur lorsque cela est nécessaire. La seconde phase consiste en une première optimisation triviale de la partition, qui consiste à enlever les sommets inutiles du séparateur en les faisant basculer dans la même partie que les sommets adjacents n'appartenant pas au séparateur.

L'opérateur de mutation peut être appliqué avant le traitement de validation, afin d'augmenter la capacité d'exploration de l'espace de recherche tout en n'effectuant qu'une seule fois la phase de réparation.

Une autre remarque concerne l'importance de l'aléa dans le choix du point de crossover : c'est cet aspect aléatoire qui rend la phase de reproduction indépendante de la numérotation du graphe. C'est aussi une justification sur l'inutilité d'avoir un crossover à points d'échange multiples dans le cadre de notre algorithme.

4.2.1.5 Modes de sélection

Un autre point crucial de la mise en place des algorithmes génétiques est le choix du mode de sélection des individus pour la reproduction. N'ayant pas trouvé de critères théoriquement

pertinents concernant la comparaison des différentes approches, nous avons implanté les principales méthodes référencées dans la littérature et laissons à l'utilisateur le choix de celle qui lui convient le mieux.

Parmi les méthodes de sélection que nous avons implantées, nous pouvons citer :

- la sélection par roulette de casino (« *roulette wheel* ») [39]. Elle consiste à sélectionner aléatoirement deux individus, en leur associant une probabilité de sélection proportionnelle à leur adéquation. Le nom de cette sélection vient du fait que l'on peut physiquement la représenter par une roulette dont la taille des secteurs est proportionnelle à l'adéquation des individus qu'ils représentent (figure 4.7a).
- la sélection par classement [31, 84]. C'est une variante de la sélection par roulette de casino qui a été introduite pour préserver de la diversité lorsqu'un petit nombre d'individus domine le reste de la population au sens du fitness. Dans ce cas, les individus dominants ont une très forte probabilité de se reproduire, et souvent seulement entre eux. Une solution est de classer les individus en fonction de leur valeur d'adéquation et de les sélectionner avec une probabilité inversement proportionnelle à leur rang. La sélection par classement est donc une sélection par roulette qui se base non plus sur la valeur de la fonction d'adéquation, mais effectue sa sélection sur le rang des individus classés pour déterminer les probabilités de sélection. Cette forme de sélection évite donc les dominations comme celle de l'individu numéro 4 de la figure 4.7a.
- la sélection par tournoi [28]. Dans cette méthode, on sélectionne un ensemble d'un ou plusieurs individus et l'on conserve l'individu qui domine l'ensemble. Un individu est dit dominant si, lorsqu'on classe les différents critères par importance, ses valeurs dans ces critères sont supérieures à celles des autres individus de l'ensemble. Cette technique permet de se passer de la fonction d'adéquation et autorise de manière simple les optimisations multi-critères dans le cas où les critères ne sont pas comparables entre eux [40]. Dans le cadre du partitionnement de graphe, on peut ainsi découpler les différents critères qui ont été décrits pour la création de la fonction d'adéquation. La sélection par tournoi pourrait s'avérer une bonne solution dans le cadre d'une future évolution vers un partitionneur de graphes multi-critères. Un exemple de tournoi est visible en figure 4.7b.

Ces différentes stratégies sont appliquées dans le cadre du renouvellement de la population lors du passage à la génération suivante. Dans notre implantation, les individus issus de la reproduction remplacent aléatoirement d'autres éléments tirés au sort, avec une probabilité inversement proportionnelle à leur qualité.

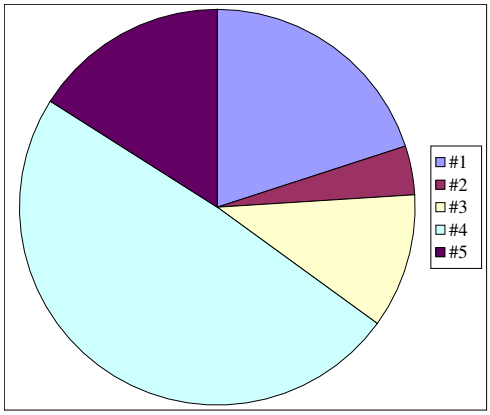
Néanmoins, afin d'être sûr de conserver les meilleurs individus, nous utilisons également une méthode d'élitisme, qui consiste à garder systématiquement les meilleurs représentants de la population. Seule une faible partie de celle-ci est concernée, de manière à ne pas perturber le processus d'évolution.

4.2.2 Parallélisme et algorithmes génétiques

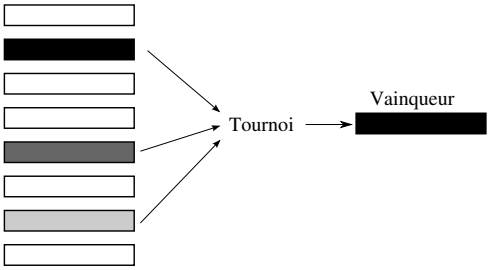
Nous avions justifié notre volonté d'utiliser des algorithmes génétiques par le fait qu'ils permettaient de mettre en œuvre facilement un raffinement parallèle. Nous allons décrire dans cette section comment y parvenir.

4.2.2.1 Manières de paralléliser les algorithmes génétiques

Il existe plusieurs façons de paralléliser les algorithmes génétiques. La plus connue est basée sur l'utilisation de plusieurs populations, et sera décrite dans la section suivante.



(a) Sélection par la roulette de casino : les individus sont répartis sur la table de casino dans une aire proportionnelle à leur adéquation. Les individus ayant les secteurs les plus grands possèdent donc plus de chance d'être sélectionnés. Les pourcentages représentent la part des individus dans le total des fitness, par exemple l'individu #4 représente 49% de ce total.



(b) Sélection par tournoi. Trois individus (deux en gris et un en noir) sont présélectionnés pour concourir entre eux : l'individu vainqueur, ici en noir, sera celui sélectionné parmi les trois.

FIG. 4.7: Illustration de différentes méthodes de sélection.

Une autre approche consiste à utiliser une sélection par tournoi [28], comme décrit précédemment. En effet, seuls les représentants appartenant au sous-ensemble des individus présélectionnés sont impliqués dans un tournoi. Il n'est donc pas nécessaire d'avoir une fonction d'adéquation ayant une valeur globale et qui nécessiterait des communications pour être calculée. Cependant, dans le cas où les individus présélectionnés sont répartis sur plusieurs processeurs, leurs comparaisons vont cependant nécessiter des communications. La solution qui consiste à autoriser uniquement des tournois entre individus locaux est quant à elle inenvisageable car elle biaise trop fortement l'aspect aléatoire du tirage.

Une dernière approche, décrite comme étant massivement parallèle, est celle des algorithmes génétiques cellulaires [17, 56]. L'idée principale est de permettre les reproductions entre individus, mais seulement dans leurs voisinages. Selon la taille du voisinage et la distribution des individus, on peut retrouver le modèle multi-populations que nous allons maintenant décrire.

4.2.2.2 Modèle multi-dèmes

Un premier niveau de parallélisme dans l'utilisation des algorithmes génétiques peut être obtenu en utilisant un modèle d'îles [29, 75, 76, 86]. Dans ce modèle, l'algorithme manipule non plus une mais plusieurs populations ; il est alors dit « multi-dèmes ».

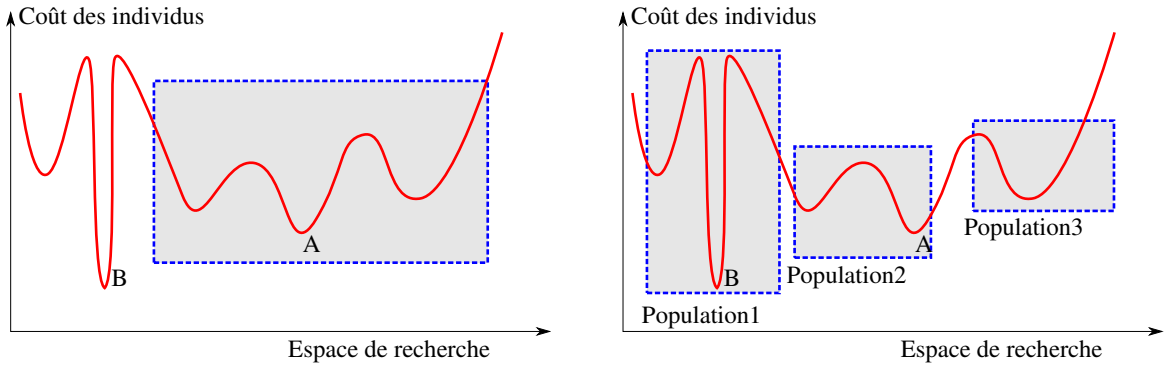
Le concept de cette approche est de faire évoluer plusieurs populations, autrement appelées dèmes ou îles, indépendamment les unes des autres. On limite donc les risques de convergence vers un unique super-individu qui en fait peut ne correspondre qu'à un maximum local de la fonction d'adéquation (figure 4.8a). Pour profiter malgré tout de la taille et de la capacité d'exploration de la population totale, des migrations entre les différents dèmes peuvent être effectuées. La fréquence de ces migrations peut être adaptée selon différentes politiques :

- des migrations régulières, au bout d'un certain nombre de générations ;
- des migrations aléatoires ;
- des migrations lorsqu'il y a convergence vers un super individu chez l'émetteur ;

– des migrations lorsque le nombre d'individus du récepteur devient trop faible, cette stratégie n'étant disponible que dans le cas d'un algorithme à population non constante.

L'intérêt des migrations peut être double, une migration pouvant augmenter la diversité pour l'émetteur comme pour le récepteur. En effet, si les individus migrants sont sélectionnés parmi ceux qui ont la valeur d'adéquation la plus élevée, le risque de convergence vers un super-individu est diminué. La réception des immigrants peut quant à elle augmenter l'entropie, c'est-à-dire la diversité génétique, dans la population réceptrice, surtout dans le cas où la convergence était amorcée vers un autre extremum local.

La figure 4.8b illustre cette capacité des algorithmes multi-dèmes à mieux explorer l'espace des solutions, à population totale équivalente.



(a) Espace de recherche après quelques itérations dans le cas d'un algorithme génétique mono-dème. Le plateau des coûts dans la zone du minimal local A a happé la plupart des individus. La probabilité de trouver le minimum global B est de fait très faible puisqu'il devient uniquement accessible par mutation.

(b) Espace de recherche après quelques itérations d'un algorithme génétique utilisant trois populations. Chaque population a tendance à converger vers un minimum local, mais la population 1 n'est pas perturbée par le plateau situé aux alentours de A et pourra donc très certainement converger vers le minimum global B.

Fig. 4.8: Comparaison des capacités de recherche d'un algorithme génétique à une seule population avec un algorithme utilisant trois dèmes.

Le parallélisme induit par cette stratégie multi-dèmes est clairement visible : il suffit de répartir chaque sous-population sur un processeur, et les seules communications concerneront uniquement les migrations. Ces dernières peuvent de plus être restreintes géographiquement comme c'est le cas en figure 4.9 où chaque population est associée à un processeur et où l'on contraint ceux-ci à ne pouvoir échanger d'informations qu'avec leurs voisins immédiats au sein de l'architecture parallèle.

Dans le cadre de PT-Scotch, nous autorisons les migrations entre n'importe quels processeurs, le nombre de populations utilisées étant généralement largement suffisant pour éviter que l'optimum global échappe à l'une des populations.

Les algorithmes génétiques ainsi présentés peuvent fonctionner seuls pour partitionner un graphe. Néanmoins, leur coût en temps est beaucoup trop important par rapport à la taille des graphes que nous souhaitons traiter.

4.2.3 Mise en œuvre dans le contexte de raffinement

L'exploitation du graphe bande permet de fournir un espace de recherche beaucoup plus petit que le graphe original. Cependant, le fait d'être dans la phase de raffinement procure d'autres

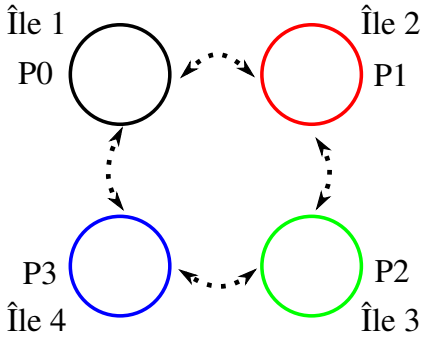


FIG. 4.9: Un algorithme génétique à quatre dèmes : distribution possible sur les processeurs et communications autorisées. Les cercles représentent les populations, qui sont chacune attribuée à un processeur. Les flèches indiquent les migrations possibles entre les différentes îles.

informations, car nous connaissons également une partition a priori de bonne qualité, qui est celle projetée depuis le niveau précédent. Notre implantation des algorithmes génétiques exploite ce fait lors du choix de la population initiale, qui se compose de trois types d'individus :

- un individu correspondant à la partition projetée ;
- une partie de la population constituée de mutations aléatoires de l'individu représentant la partition projetée ;
- le reste de la population, composé d'individus aléatoires.

Comme nous utilisons également la méthode d'élitisme et gardons en mémoire de ce fait le meilleur individu trouvé lors de l'exploration, notre algorithme ne peut pas dégrader la solution obtenue par la projection de celle du niveau inférieur.

Les individus aléatoires permettent d'avoir une certaine diversité, qui est nécessaire pour faire progresser les algorithmes génétiques en évitant que l'on ait seulement des « mariages entre cousins ». Les individus issus de la partition initiale sont, dans l'idée, les individus de base qui, en se reproduisant avec des individus aléatoires, peuvent améliorer la qualité du partitionnement.

Toujours dans un souci d'efficacité, nous pouvons remarquer qu'il est possible d'appliquer à certains individus des optimisations locales. En effet, l'application d'une procédure d'optimisation locale, sur les champions notamment, permet d'être certain que ceux-ci correspondent à un optimum local de la fonction d'adéquation. Dans notre cas, nous utilisons une méthode de type gradient, qui correspond en fait à un Fiduccia-Mattheyses auquel nous interdisons tout mouvement qui détériorerait la qualité du partitionnement. Le coût d'application d'une telle heuristique est faible et permet d'éviter d'inutiles générations pour converger vers l'optimum local.

Un tel couplage entre un algorithme d'optimisation local et un algorithme génétique s'appelle un algorithme « mimétique » [59, 60]. Son idée de base peut être comprise en introduisant la notion de connaissance au sein du modèle génétique : ainsi, au lieu d'une simple progression au gré du hasard, on progresse en étant guidé par une fonction d'optimisation qui correspond à notre problème. Le danger de cette approche est une application systématique de la méthode d'optimisation locale, qui va à l'encontre du besoin de diversité des individus nécessaire au bon fonctionnement des algorithmes évolutionnistes.

4.2.4 Vers plus de parallélisme

4.2.4.1 Choix pouvant améliorer le parallèlisme du modèle multi-dèmes

Le modèle multi-dèmes présenté précédemment permet d'avoir un parallélisme pratiquement total entre les calculs effectués par les différents processeurs, dans le cas où chaque processeur possède sa propre population.

En pratique, le degré de parallélisme pouvant être atteint est fortement dépendant de la politique de migration qui a été retenue.

Une première façon de faire est d'exploiter des populations de taille non constante. Les individus possèdent un âge et meurent lorsqu'un seuil est dépassé, faisant ainsi diminuer la population. Une autre variante pourrait être d'associer une probabilité de survie aux individus en fonction de leur âge. Lorsque la taille d'une population devient critique, c'est-à-dire devient insuffisante pour préserver un certain niveau de diversité génétique, cette population émet des demandes d'immigration en direction de ses voisins. Ceux-ci envoient leurs meilleurs individus avec une certaine probabilité et selon la taille de leur population. Le fait pour le donneur de perdre des individus lui permet de profiter aussi de la migration en diminuant le risque de convergence prématurée vers un super-individu.

Le problème de cette stratégie est qu'elle introduit une légère synchronisation entre les différentes populations et qu'elle peut donc se révéler coûteuse dans le cas d'un grand nombre de populations différentes.

Dans une autre méthode, les migrations sont cette fois initiées par les donneurs. Par exemple, lorsqu'une convergence prématurée est détectée, ou alors tout simplement à un instant choisi aléatoirement, certains individus sont envoyés vers des populations voisines, où ils seront accueillis en remplacement d'individus locaux. Il est possible avec cette technique de travailler avec des populations constantes, mais à ce moment là l'émetteur ne bénéficie plus de la protection contre une convergence prématurée. En fait, le principal intérêt de cette méthode est qu'elle ne nécessite aucune synchronisation entre les différents dèmes et qu'elle tolère même la perte d'individus durant les migrations.

C'est cette dernière solution que nous avons pour l'instant retenue dans l'implantation des algorithmes génétiques au sein de Scotch.

4.2.4.2 Parallélisme induit par la distribution des individus

La principale limitation de l'approche génétique telle qu'elle a été présentée précédemment est qu'elle ne distribue que des populations sur les différents processeurs, et donc qu'elle devient inexploitable dans le cas où les individus ne peuvent plus être stockés dans la mémoire d'une seule machine.

Il est toutefois possible de réaliser une distribution des individus sur les processeurs tout en conservant le parallélisme procuré par notre modèle multi-dèmes. Les différentes populations ne sont maintenant plus associées à un seul processeur chacune mais à un ensemble de processeurs. Un individu est donc réparti sur les processeurs associés à une population.

Le schéma parallèle résultant est illustré en figure 4.10. Cependant, il n'a pas encore été implanté dans PT-Scotch, les individus traités pour l'instant pouvant encore loger en mémoire.

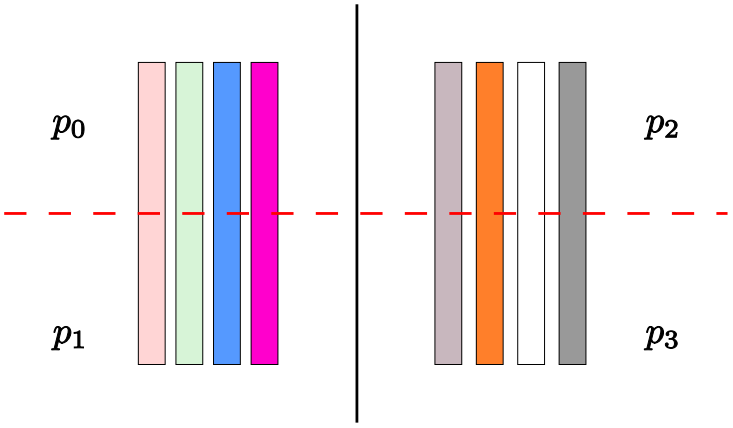


FIG. 4.10: Exemple d'algorithme génétique multi-dèmes à individus distribués. Les individus, représentés par des rectangles teintés, sont distribués sur deux processeurs, selon leur population. Ainsi, les processeurs p_0 et p_1 possèdent la population composée des quatres individus à gauche, les processeurs p_2 et p_3 possédant ceux de droite. La ligne continue verticale indique la limite entre les deux populations, la ligne horizontale en tirets indique la limite de distribution des individus.

4.3 Algorithmes génétiques pour le partitionnement de graphes : comparaison avec l'existant

La principale différence entre nos algorithmes génétiques et ceux présentés jusqu'alors réside dans l'exploitation du graphe bande pour limiter la taille de l'espace de recherche. Comme nous avons dit précédemment, le coût des algorithmes génétiques est fortement corrélé à l'étendue de l'espace des solutions. Puisqu'une solution pseudo-globale peut être trouvée grâce au graphe bande, notre idée a été de la chercher au moyen d'algorithmes génétiques au lieu des traditionnels algorithmes d'optimisation locale tels que Kernighan-Lin ou Fiduccia-Mattheyses.

À la différence d'autres approches [3, 4, 74], nous utilisons donc des algorithmes génétiques dans le cadre du raffinement lors de l'utilisation d'un schéma multi-niveaux. Areibi [3, 4] utilise directement les algorithmes génétiques pour trouver une partition, ce qui peut encore être réalisable sur des graphes VLSI comportant au maximum 25 000 sommets, mais devient rapidement trop coûteux sur des graphes plus importants tels que ceux que nous souhaitons manipuler.

Walshaw [74] utilise pour sa part une approche évolutionniste qui consiste à perturber légèrement l'organisation topologique du graphe, en changeant très légèrement les poids des arêtes, afin de changer le comportement de la phase de contraction du schéma multi-niveaux. Une partition est donc calculée sur ce graphe modifié et est projetée sur le graphe original, en espérant que la perturbation initiale n'était pas trop importante et donc que la qualité de la partition projetée puisse être bonne. L'algorithme d'initialisation utilisé respecte cette condition mais on peut constater que chaque individu subit une exécution complète de l'algorithme multi-niveaux classique implanté dans JOSTLE [82].

Le défaut majeur de cette approche est qu'elle oblige, pour chaque individu, à l'exécution complète d'un algorithme de partitionnement de type multi-niveaux avec un raffinement utilisant une optimisation locale. Le temps total d'exécution est beaucoup plus important que celui des méthodes usuelles puisqu'elles sont elles-mêmes utilisées plusieurs fois.

Nous constatons donc que notre approche diffère fortement des travaux précédents, en re-

cherchant tout d'abord une certaine qualité dans les résultats mais en respectant des contraintes en temps assez fortes.

Nous allons étudier dans la section suivante si ces algorithmes peuvent être réellement exploitables dans des cas concrets.

4.4 Expérimentations des algorithmes génétiques

4.4.1 Observations générales

Nous avons implanté notre algorithme génétique au sein d'un prototype purement séquentiel dans un premier temps, puis multi-threadé dans un second temps. Ce prototype nous a permis d'apprendre comment notre algorithme réagissait en situation réelle, lorsqu'on modifie les différents paramètres.

Plusieurs constats peuvent être faits : dans notre cadre d'utilisation, nous exploitons les connaissances disponibles grâce au schéma multi-niveaux, ce qui rend certains paramètres peu déterminants.

Ainsi, le changement de la méthode utilisée pour la sélection semble en fait ne pas avoir beaucoup de conséquences. C'est pour cela que, dans la suite, nous utiliserons la sélection la plus classique, à savoir celle de la roulette de casino. Toujours en ce qui concerne la sélection, la présence d'élitisme paraît en revanche indispensable, afin d'obtenir une meilleure convergence.

Les premiers tests séquentiels ont pour leur part surtout permis de fixer les coefficients de la fonction d'adéquation. L'expression que nous avons retenue est la suivante :

$$f = f_1(bal) \cdot (\alpha f_2(|S|) + \beta f_3(|E(S)|)) \ , \tag{4.1}$$

avec

$$\alpha = 0.9 \text{ et } \beta = 0.1 \ . \tag{4.2}$$

Les fonctions f_1, f_2 et f_3 sont des fonctions à valeurs dans $[0, 1]$ et dépendant :

- pour f_1, de l'équilibre des partitions. Un exemple des valeurs prises par cette fonction est disponible en section 4.2.1.2 ;
- pour f_2, de la taille du séparateur. Son expression est relative au meilleur séparateur non nul connu à un instant donné et est :

$$f_2(t) = 0.8 \exp\left(-4 \cdot \frac{w_v(S) - \min_{\sigma \in \text{ séparateurs connus}} w_v(\sigma)}{w_v(G)}\right) \ ;$$

- pour f_3, du poids des arêtes incidentes au séparateur. Elle permet de privilégier les séparateurs plus lisses. Son expression est, en notant $E(S)$ l'ensemble des arêtes incidentes à S :

$$f_3(t) = \frac{w_E(E(S)) - \min_{\sigma \in \text{ séparateurs connus}} w_E(E(\sigma))}{w_E(G)} \ .$$

Comme nous nous plaçons dans le contexte d'algorithmes génétiques pour le raffinement dans le cadre multi-niveaux, nous avons déjà précisé que la population initiale comportait la projection de la partition à raffiner et, comme celle-ci est généralement d'une qualité sommetoute correcte, le critère du nombre de composantes connexes n'est pas un critère discrimant, plusieurs partitions ayant vraisemblablement le même nombre de composantes connexes : une par partie.

4.4.2 Conséquences du modèle multi-dèmes

La version multi-threadée de notre prototype a été réalisée pour vérifier la viabilité du modèle multi-dèmes avant d'en réaliser une version parallèle MPI. L'implantation repose directement sur la version séquentielle utilisée précédemment et des threads sont créées pour gérer les différentes populations. Chaque population est gérée par un thread, et les évolutions sont calculées de manière indépendante et parallèle dans le cas d'une exécution sur une machine à mémoire partagée.

4.4.2.1 Descriptif des paramètres utilisés dans l'algorithme génétique

Le fonctionnement global de notre algorithme génétique est celui du modèle multi-dèmes décrit en section 4.2.3. Cependant, nous avions laissé dans la partie théorique un nombre pléthorique de valeurs pour les différents paramètres régissant le comportement de l'algorithme.

Notre prototype fonctionne avec autant de populations que de threads, dont le nombre est fourni en paramètre. Toutes ces populations suivent les mêmes lois de sélection, de reproduction et de migration.

La fonction d'adéquation utilisée est celle issue des expérimentations séquentielles et qui a été présenté en section 4.4.1. On peut noter que deux individus similaires, mais n'appartenant pas à la même population, ne sont pas forcément associés à la même valeur de fitness, s'ils ont un individu de référence différent. La fonction d'adéquation a une valeur seulement locale, son utilité étant seulement la comparaison de la qualité d'individus locaux.

La procédure de sélection utilisée est celle de la roulette de casino, associée à de l'élitisme pour conserver les meilleurs individus. Ces critères de sélection ont été choisis à la lumière des expérimentations séquentielles qui ont montré que, pour notre problème, les différentes stratégies semblent équivalentes et que la roulette de casino est parmi les plus rapides à effectuer.

Parlons maintenant du processus de migration, qui doit permettre d'améliorer la qualité déjà procurée par l'aspect multi-séquentiel du modèle multi-dèmes. Nous avons étudié plusieurs processus de migration et avons finalement retenu d'effectuer les migrations quand une convergence vers un super-individu local est détectée, c'est-à-dire lorsque la diversité génétique baisse. La diversité génétique est estimée en calculant le nombre de différences entre des individus de qualité équivalente et le meilleur individu de la population. Cette évaluation est coûteuse, mais elle n'est effectuée que sur un petit nombre d'individus et seulement si le meilleur individu de la population locale change.

Toujours dans le but d'éviter une convergence trop rapide vers un super-individu, nous utilisons des populations de tailles variables, en attribuant à chacun de nos individus un âge. La diminution du nombre d'individus dans le dème permet aussi de diminuer le temps de calcul pour changer de génération, lorsque le nombre d'itérations devient grand.

4.4.2.2 Résultats avec le prototype de raffinement génétique multi-threadé

Le tableau 4.1 présente les résultats obtenus pour la renumérotation de différents graphes par notre prototype d'algorithme génétique multi-dèmes.

Le premier constat est que la qualité obtenue est en général assez peu dépendante du nombre de processeurs. Le comportement de l'algorithme semble effectivement très stable sur les graphes `coupole8000` et `audikw1` notamment.

Sur d'autres graphes, comme `altr4`, `conesphere1m` ou encore `threads`, on constate que la qualité se dégrade quelque peu lorsque le nombre de populations utilisées augmente. Cependant, cette dégradation doit être comparée avec celle générée par l'utilisation de ParMeTiS dans les

Graphe	**Nombre de dèmes et de threads**						
	1	2	4	8	16	32	64
`bcsstk32`	1.60e+9	1.55e+9	1.67e+9	1.82e+9	1.83e+9	1.53e+9	2.07e+9
`audikw1`	5.68e+12	5.91e+12	5.70e+12	5.82e+12	5.99e+12	6.44e+12	6.02e+12
`bmw32`	3.04e+10	3.44e+10	3.75e+10	4.13e+10	4.64e+10	4.57e+10	5.01e+10
`altr4`	3.46e+8	3.71e+8	4.23e+8	4.06e+8	4.31e+8	4.92e+8	4.71e+8
`conesphere1m`	1.90e+12	1.92e+12	1.99e+12	2.37e+12	2.34e+12	2.53e+12	2.63e+12
`coupole8000`	7.64e+10	7.64e+10	7.62e+10	7.65e+10	7.66e+10	7.68e+10	7.66e+10
`thread`	4.10e+10	3.99e+10	4.41e+10	4.64e+10	4.43e+10	4.59e+10	5.19e+10

TAB. 4.1: Coût des renumérotations (OPC) obtenues par notre algorithme (LGA) multi-niveaux utilisant le graphe bande et les algorithmes génétiques sur les $\log_2(p)$ premiers niveaux.

Graphe	**Nombre de dèmes et de threads**						
	1	2	4	8	16	32	64
`bcsstk32`	0.42	0.88	0.84	0.97	2.07	(2.86)	(4.06)
`audikw1`	19.78	22.77	29.55	32.89	60.24	(74.64)	(91.78)
`bmw32`	1.69	1.79	2.48	2.36	3.67	(5.11)	(7.80)
`altr4`	0.65	1.78	2.25	1.95	3.36	(5.43)	(7.20)
`conesphere1m`	44.03	69.66	86.47	90.44	120.87	(134.85)	(158.07)
`coupole8000`	125.69	75.40	55.19	49.16	52.59	(61.93)	(77.26)
`thread`	0.56	2.33	3.10	2.93	4.22	(5.02)	(5.92)

TAB. 4.2: Temps de renumérotations (en s) avec notre algorithme (LGA) multi-niveaux utilisant le graphe bande et les algorithmes génétiques sur les $\log_2(p)$ premiers niveaux. Les temps entre parenthèses correspondent à des temps d'exécutions simulés car réalisés sur une machine ne disposant que de seize cœurs.

mêmes conditions et l'on constate alors que l'on n'a pas la même explosion du coût induit par les renumérotations.

La qualité obtenue par notre algorithme génétique est, en effet, très satisfaisante par rapport à celle obtenue par PARMETIS (cf. tableau 3.4 de la page 83) avec par exemple près de deux fois moins d'opérations à effectuer pour factoriser la matrice `audikw1` renumérotée à l'aide de 64 processus ou threads.

Par rapport aux résultats obtenus avec la version parallèle de SCOTCH (cf. tableau 3.3 de la page 82), exploitant un raffinement multi-séquentiel du graphe bande par l'algorithme de Fiduccia-Mattheyses, on constate que les resultats sont globalement un peu moins flatteurs lorsque le nombre de processeurs augmente, mais que l'on reste finalement assez proche de la version FM, l'écart se chiffrant à environ 10% pour le graphe `audikw1` par exemple. En outre, la paramétrisation des algorithmes génétiques peut vraisemblablement permettre d'augmenter la qualité de renumérotation, en augmentant la taille des populations ou le nombre de générations employés, au prix d'une augmentation des temps de calculs.

Le résultat important qui ressort des expérimentations est que, si la taille des populations reste raisonnable, la qualité du partitionnement obtenu dépend essentiellement de la taille de la population totale plus que du nombre de ces populations. Ces méthodes sont donc fortement parallélisables.

Les temps de calcul sont au sein du tableau 4.2. On constate tout d'abord que ces temps

ne sont pas prohibitifs pour l'utilisation des algorithmes génétiques, le temps de renumérotation obtenu sur `coupole8000` avec huit threads étant par exemple équivalent à celui obtenu avec la version parallèle de SCOTCH (cf. tableau 3.5) utilisant la stratégie décrite précédemment. Toujours avec huit processeurs, on constate que la version génétique est plus rapide de 35% sur le graphe `audikw1`, tout en produisant une renumérotation d'une qualité assez proche. En revanche, sur d'autres graphes, comme `conesphere1m`, les temps calculs sont défavorables à la version génétique, et ceci d'autant plus que le nombre de processeurs augmente.

En fait, pour comprendre ce phénomène, il est nécessaire d'étudier le comportement de notre algorithme LGA lorsque le nombre de processeurs augmente. Le nombre de niveaux sur lesquels sont utilisés les algorithmes génétiques est égal à $\log_2(p)$ et donc croît aussi quand le nombre de processeurs augmente. Or, l'application des algorithmes génétiques est environ 80 fois plus coûteuse que celle de l'algorithme de Fiduccia-Mattheyses, ce qui peut justifier l'effet anti-scalable constaté sur les temps de calculs certains graphes, l'accélération due à la parallélisation des algorithmes génétiques n'étant plus suffisante pour compenser le nombre de leurs applications.

Les algorithmes génétiques permettent donc de bénéficier d'un certain niveau de parallélisme, mais ils semblent aussi présenter des limites qui semblent liées à la nature et à la taille du problème à traiter. Cependant, leurs utilisations sur les plus hauts niveaux de la méthode des dissections emboîtées et du schéma multi-niveaux semblent être une solution viable et prometteuse pour travailler sur des graphes dont les graphes bandes nécessitent d'être distribués.

Conclusion et perspectives

Conclusion

Nous avons présenté dans ce document le travail que nous avons effectué autour de la conception d'algorithmes parallèles efficaces de partitionnement de graphe et de renumérotation de matrices creuses. Nous avons aussi évoqué sa mise en pratique au sein de la bibliothèque Scotch.

Afin de maintenir le même niveau de qualité dans les résultats produits que les meilleures approches séquentielles, nous avons conservé l'approche utilisé dans la version 4.0 de Scotch, c'est-à-dire utilisation de la méthode des dissections emboîtées avec un bipartitionnement utilisant une technique multi-niveaux. Nous avons donc parallélisé la méthode des dissections emboîtées, ainsi que les phases clés du schéma multi-niveaux que sont l'étape de contraction et la phase d'expansion.

Concernant l'étape de contraction, nous l'avons parallélisée en introduisant tout d'abord une amélioration des algorithmes de synchronisation des décisions inter-processus relatives à l'appariement des sommets par l'ajout une étape de sélection probabiliste des messages à échanger.

Nous avons également proposé un nouvel algorithme de synchronisation des choix d'appariements en mettant en place un système de classement des processus en fonction de leur couleur dans le graphe de voisinage des processeurs. Cette technique, utilisant un coloriage de Luby, a été validée avec succès et permet d'augmenter de manière significative l'efficacité et la qualité de l'étape de contraction parallèle.

Nous avons aussi ajouté à notre procédé de contraction la capacité de replier et de dupliquer les graphes des niveaux les plus grossiers afin de profiter des effets positifs des exécutions multiples du schéma constatées en séquentiel. En outre, cette phase n'induit pas de surcoût en temps, seulement un surcoût mémoire.

La phase d'expansion a, quant à elle, fait l'objet de plusieurs contributions. La première est l'introduction du concept de graphe bande qui permet de réduire de manière très significative la taille des problèmes à optimiser localement lors du raffinement.

L'utilisation du graphe bande est même dorénavant intégrée dans la version séquentielle de Scotch 5.0, où elle peut être combinée avec n'importe quelle méthode d'optimisation locale.

La seconde évolution majeure concerne l'utilisation du graphe bande dans le cas d'un partitionnement parallèle. La faible taille de ce graphe bande par rapport au graphe original permet actuellement de pouvoir traiter son raffinement sur un ordinateur séquentiel. Comme nous disposons de plusieurs processeurs lors de l'expansion, nous effectuons en fait une optimisation multi-séquentielle du graphe bande. Cette stratégie nous permet d'obtenir des résultats du même ordre de qualité que ceux qui auraient été obtenus par un renuméroteur séquentiel, s'il avait été possible de l'utiliser sans avoir de problème de mémoire.

La taille des graphes pouvant être traités avec ce procédé est par contre limitée et il a été nécessaire d'explorer de nouvelles voies pour pouvoir partitionner les graphes encore plus grands qui seront disponibles dans un futur proche.

C'est pour cela qu'une approche utilisant des algorithmes génétiques a été développée. L'originalité de notre méthode est d'exploiter la réduction de la taille de l'espace de recherche procurée par la méthode du graphe bande. Cette approche a été validée avec succès dans le cadre d'un prototype multi-threadé implantant un algorithme génétique multi-dèmes.

Toutes ces approches ont été validées dans le cadre de la renumérotation parallèle de matrices creuses. Nous disposons dorénavant dans Scotch 5.0 d'un renuméroteur parallèle et distribué efficace et qui a été comparé très favorablement par rapport à l'outil parallèle de référence, ParMeTiS.

L'objectif d'avoir un renuméroteur parallèle aussi efficace en terme de qualité que les solutions séquentielles a donc été atteint, sur un grand nombre de cas industriels. De plus, nous avons déployé notre outil sur plusieurs systèmes différents sans problème majeur et c'est pourquoi, mis à part le prototype d'algorithme génétique, tous les algorithmes que nous avons décrit sont intégrés dans la version 5.0 de la bibliothèque Scotch.

Perspectives

Dans les perspectives immédiates, nous pouvons citer le besoin de valider notre approche sur un plus grand ensemble de cas tests, de tailles plus importantes, et aussi sur un plus grand nombre de processeurs.

Les résultats obtenus dans le domaine de la renumérotation de matrices creuses sont très encourageants et mériteraient donc d'être étendus au partitionnement k-aire de graphes ou de maillages, qui sont des fonctionnalités présentes dans la version séquentielle de Scotch.

Les étapes clés du schéma multi-niveaux étant communes avec le renuméroteur, l'obtention d'un partitionneur parallèle de graphes ou de maillages devrait être assez rapide.

Ensuite, plusieurs aspects algorithmiques mériteraient d'être ré-étudiés, parmi lesquels :

- un module de chargement permettant d'effectuer une meilleure distribution initiale des données, afin d'améliorer les temps de calcul et la consommation mémoire sur les graphes mal numérotés ;
- la mise en place d'une meilleure stratégie de rééquilibrage de la charge plus efficace pendant le calcul, qui est facilement intégrable au module de contraction ;
- l'utilisation des algorithmes génétiques dans un contexte distribué, afin d'obtenir une scalabilité plus importante et de pouvoir utiliser des architectures massivement parallèles comme les machines BlueGene ;
- la réalisation d'une phase d'expansion multi-threadée qui permettrait d'exploiter au mieux les clusters de nœuds SMP en partageant le graphe bande au sein d'un nœud ;
- une étude plus poussée des comportements mémoire et des conséquences de l'utilisation du repliement et de la duplication dans le schéma multi-niveaux ;
- une observation précise des conséquences de la multi-séquentialisation du raffinement dans le cadre des renumérotations de matrices creuses. L'étude de son impact avec un solveur parallèle pourrait permettre d'éventuellement changer de critères pour la sélection de la meilleure partition ;

– ajouter une perturbation topologique sur le graphe à raffiner dans le contexte multi-séquentiel, à la manière du schéma imaginé par Walshaw pour ses algorithmes génétiques [74] ;
– une manière différente d'aborder la contraction, se basant sur des groupements de sommets qui peuvent se superposer, à la manière de la méthode multigrille algébrique proposée par Ron dans [71]. Cette méthode pourrait permettre d'éviter la présence de conflits lors des regroupements inter-processeurs de sommets ;
– l'exploration de méthodes « *out of core* » pour permettre de limiter la consommation mémoire du schéma multi-niveaux et de généraliser la mise en œuvre des repliements et des duplications.

Annexe A

Bibliographie principale

[1] P. Amestoy, T. Davis et I. Duff : An approximate minimum degree ordering algorithm. *SIAM J. Matrix Analysis and Applications*, 17:886–905, 1996.

[2] P. Amestoy, I. Duff et J.-Y. L'Excellent : Multifrontal parallel distributed symmetric and unsymmetric solvers . *Computer methods in applied mechanics and engineering.*, 184: 501–520, 2000. Special issue on Domain Decomposition and Parallel Computing.

[3] S. Areibi : Effective exploration & exploitation of solution space of vlsi design via memetic algorithms. Search techniques, School of Engineering, University of Guelph, Guelph, Ontario, Canada, 2002.

[4] S. Areibi et Y. Zeng : Effective memetic algorithms for VLSI design automation = genetic algorithms + local search + multi-level clustering. *Evolutionary Computation*, 12(3):327–353, 2004.

[5] S. T. Barnard et H. D. Simon : A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency : Practice and Experience*, 6(2):101–117, 1994.

[6] R. Battiti et A. A. Bertossi : Greedy, prohibition, and reactive heuristics for graph partitioning. *IEEE Transactions on Computers*, 48(4):361–385, 1999.

[7] S. W. Bollinger et S. F. Midkiff : Processor and link assignment in multicomputers using simulated annealing. *In Proceedings of the 11th Int. Conf. on Parallel Processing*, pages 1–7. The Penn. State Univ. Press, août 1988.

[8] E. G. Boman, K. D. Devine, L. A. Fisk, R. Heaphy, B. Hendrickson, V. Leung, C. Vaughan, U. Catalyurek, D. Bozdag et W. Mitchell : Zoltan home page. `http://www.cs.sandia.gov/Zoltan`, 1999.

[9] T. Bui, C. Heigham, C. Jones et T. Leighton : Improving the performance of the kernighan-lin and simulated annealing graph bisection algorithms. *In DAC '89 : Proceedings of the 26th ACM/IEEE conference on Design automation*, pages 775–778, New York, NY, USA, 1989. ACM Press.

[10] T. Bui et C. Jones : A heuristic for reducing fill-in in sparse matrix factorization. *In Sixth SIAM Conference on Parallel Processing*, pages 445–452, 1993.

[11] T. N. Bui et B. R. Moon : Genetic algorithm and graph partitioning. *IEEE Trans. Comput.*, 45(7):841–855, 1996.

[12] G. W. Brown C. Farhat, N. Maman : Mesh partitioning for implicit computations via iterative domain decomposition : Impact and optimization of the subdomain aspect ratio. *International Journal for Numerical Methods in Engineering*, 38(6):989–1000, 1995.

[13] V. Cerny : A thermodynamical approach to the travelling salesman problem : an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[14] T. F. Chan : Analysis of preconditioners for domain decomposition. *SIAM Journal of Numerical Analysis*, 24(2):382–390, 1987.

[15] T.-Y. Chen, J. R. Gilbert et S. Toledo : Toward an efficient column minimum degree code for symmetric multiprocessors. *In Proc. 9^{th} SIAM Conf. on Parallel Processing for Scientific Computing, San-Antonio*, 1999.

[16] T. Chockalingam et S. Arunkumar : A randomized heuristics for the mapping problem : The genetic approach. *Parallel Computing*, 18:1157–1165, 1992.

[17] Y. Davidor : A naturally occurring niche and species phenomenon : The model and first results. *In ICGA*, pages 257–263, 1991.

[18] T. Davis : University of Florida Sparse Matrix Collection. `http://www.cise.ufl.edu/research/sparse/matrices`.

[19] J. J. Dongarra, J. Du Croz, S. Hammarling et R. J. Hanson : An extended set of FORTRAN Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–17, mars 1988.

[20] C. M. Fiduccia et R. M. Mattheyses : A linear-time heuristic for improving network partitions. *In Proceedings of the 19th Design Automation Conference*, pages 175–181. IEEE, 1982.

[21] M. Fiedler : Algebraic connectivity of graphs. *Czechoslovak Math. J.*, 23:298–305, 1973.

[22] M. Fiedler : A property of eigenvectors of non-negative symmetric matrices and its application to graph theory. *Czechoslovak Math. J.*, 25:619–633, 1975.

[23] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham et T. S. Woodall : OpenMPI : Goals, Concept, and Design of a Next Generation MPI Implementation. *In Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, septembre 2004.

[24] Y. Gao : An upper bound on the convergence rates of canonical genetic algorithms. *Complexity International*, 5:1–9, 1998.

[25] M. R. Garey, D. S. Johnson et L. Stockmeyer : Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[26] N. E. Gibbs, W. G. Poole et P. K. Stockmeyer : A comparison of several bandwidth and profile reduction algorithms. *ACM Trans. Math. Software*, 2:322–330, 1976.

[27] F. Glover : Tabu search – part 1. *ORSA Journal of Computing*, 1(3):190–206, 1989.

[28] D. E. Goldberg et K. Deb : A comparative analysis of selection schemes used in genetic algorithms. *In FOGA*, pages 69–93, 1990.

[29] M. Gorges-Schleuter : Explicit parallelism of genetic algorithms through population structures. *In PPSN*, pages 150–159, 1990.

[30] P. Graham et T.S. Collett : View-based navigation in insects : how wood ants (formica rufa l.) look at and are guided by extended landmarks. *Journal of Experimental Biology*, 205:2499–2509, 2002.

[31] J. J. Grefenstette et J. E. Baker : How genetic algorithms work : a critical look at implicit parallelism. *In Proceedings of the third international conference on Genetic algorithms*, pages 20–27, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[32] W. Gropp, E. Lusk, N. Doss et A. Skjellum : A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, septembre 1996.

[33] L. Hagen, D. Huang et A. Kahng : On implementation choices for iterative improvement partitioning algorithms, 1997.

[34] B. Hajek : Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

[35] K. M. Hall. : An r-dimensional quadratic placement algorithm. *Management Science*, 17(3):219–229, 1970.

[36] B. Hendrickson et R. Leland : A multilevel algorithm for partitioning graphs. *In Proceedings of Supercomputing*, 1995.

[37] B. Hendrickson et E. Rothberg : Improving the runtime and quality of nested dissection ordering. *SIAM Journal of Scientific Computing*, 20(2):468–489, 1998.

[38] P. Hénon, P. Ramet et J. Roman : PaStiX : A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems. *Parallel Computing*, 28(2):301–321, janvier 2002.

[39] J. H. Holland : *Adaptation in natural and artificial systems.* MIT Press, Cambridge, MA, USA, 1975.

[40] J. Horn, N. Nafpliotis et D. E. Goldberg : A niched Pareto genetic algorithm for multiobjective optimization. *In* NJ : IEEE Service Center, éditeur : *IEEE World Congress on Computational Intelligence*, volume 1, pages 82–87, 1994.

[41] IEEE. *The Open Group Technical Standard Base Specifications*, IEEE Std 1003.1 édition, avril 2004.

[42] G. Karypis : Multilevel hypergraph partitioning. Rapport technique 02-025, University of Minnesota, 2002.

[43] G. Karypis et V. Kumar : A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report 95-035, University of Minnesota, juin 1995.

[44] G. Karypis et V. Kumar : Metis *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices.* University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN 55455, U.S.A., 4 édition, septembre 1998.

[45] G. Karypis et V. Kumar : ParMetis *Parallel Graph Partitioning and Sparse Matrix Ordering Library.* University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN 55455, U.S.A., août 2003.

[46] MeTiS : Family of multilevel partitioning algorithms. `http://glaros.dtc.umn.edu/gkhome/views/metis`.

[47] B. W. Kernighan et S. Lin : An efficient heuristic procedure for partitioning graphs. *BELL System Technical Journal*, pages 291–307, février 1970.

[48] S. Kirkpatrick, C. D. Gelatt et M. P. Vecchi : Optimization by simulated annealing. *Science*, 220(4598):671–680, mai 1983.

[49] P. Korošec, J. Šilc et B. Robič : Solving the mesh-partitioning problem with an ant-colony algorithm. *Parallel Computing*, 30(5-6):785–801, 2004.

[50] B. Krishnamurthy : An improved min-cut algorithm for partitioning vlsi networks. *IEEE Trans. Computers*, 33(5):438–446, 1984.

[51] A. E. Langham et P. W. Grant : Using competing ant colonies to solve k-way partitioning problems with foraging and raiding strategies. *In ECAL '99 : Proceedings of the 5th European Conference on Advances in Artificial Life*, pages 621–625, London, UK, 1999. Springer-Verlag.

[52] C. L. Lawson, R. J. Hanson, D. R. Kincaid et F. T. Krogh : Basic Linear Algebra Subprograms for Fortran usage. *ACM Transactions on Mathematical Software*, 5(3):308–323, septembre 1979.

[53] R. J. Lipton, D. J. Rose et R. E. Tarjan : Generalized nested dissection. *SIAM Journal of Numerical Analysis*, 16(2):346–358, avril 1979.

[54] M. Luby : A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1055, 1986.

[55] M3PEC-Mésocentre, Modélisation Microscopique et Mésoscopique en Physique, dans l'Environnement, en Chimie, Mathématiques, Infomatique et Médecine. `http://www.m3pec.u-bordeaux1.fr/`.

[56] B. Manderick et P. Spiessens : Fine-grained parallel genetic algorithms. *In ICGA*, pages 428–433, 1989.

[57] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller et E. Teller : Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, juin 1953.

[58] H. Meyerhenke et S. Schamberger : A parallel shape optimizing load balancer. *In Proc. Europar, Dresden, LNCS 4128*, pages 232–242, septembre 2006.

[59] P. Moscato : On evolution, search, optimization, genetic algorithms and martial arts, towards memetic algorithms. Rapport technique 826, California Intitute of Technology, Pasadena, CA 91125, U.S.A., 1989.

[60] P. Moscato et M. G. Norman : A 'Memetic' Approach for the Traveling Salesman Problem. Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems. *Parallel Computing and Transputer Applications*, pages 187–194, 1992.

[61] Mpi-2.0 standards. `http://www.mpi-forum.org/docs/docs.html`, 1997.

[62] C. H. Papadimitriou : The NP-completeness of the bandwidth minimization problem. *Computing*, 16:263–270, 1976.

[63] PARASOL project (EU ESPRIT IV LTR Project No. 20160), 1996–1999.

[64] F. Pellegrini : *Scotch and libScotch 4.0 User's Guide*. ENSEIRB, LaBRI, INRIA Futurs.

[65] F. Pellegrini : *Application de méthodes de partition à la résolution de problèmes de graphes issus du parallélisme*. Thèse de Doctorat, LaBRI, Université Bordeaux I, 351 cours de la Libération, 33405 Talence, France, janvier 1995.

[66] F. Pellegrini : A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries. *In Proc. Europar, Rennes*. ENSEIRB, LaBRI, INRIA Futurs, 2007.

[67] F. Pellegrini, J. Roman et P. Amestoy : Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. *Concurrency : Practice and Experience*, 12:69–84, 2000.

[68] Scotch : Static mapping, graph partitioning, and sparse matrix block ordering package. `http://www.labri.fr/~pelegrin/scotch/`.

[69] A. Pothen, H. D. Simon et K.-P. Liou : Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Analysis*, 11(3):430–452, juillet 1990.

[70] J. Roman : Calculs de complexité relatifs à une méthode de dissection emboîtée. *Numerische Mathematik*, 47:175–190, 1985.

[71] D. Ron, S. Wishko-Stern et A. Brandt : An algebraic multigrid based algorithm for bisectioning general graphs. Rapport technique MCS05-01, Department of Computer Science and Applied Mathematics, the Weizmann Institute of Science, 2005.

[72] G. Rudolph : Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5(1):96–101, janvier 1994.

[73] Y. Saad : *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.

[74] J. Soper, C. Walshaw et Cross M. : A combined evolutionary search and multilevel optimisation approach to graph partitioning. Rapport technique 00/IM/58, University of Greenwich, London, UK, avril 2000.

[75] T. Starkweather, L. D. Whitley et K. E. Mathias : Optimization using distributed genetic algorithms. *In PPSN*, pages 176–185, 1990.

[76] R. Tanese : Distributed Genetic Algorithms. *In Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 434–439, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[77] L. Tao et Y. C. Zhao : Multi-way graph partition by stochastic probe. *International Journal of Computers & Operations Research*, 20(3):321–347, 1993.

[78] W. F. Tinney et J. W. Walker : Direct solutions of sparse network equations by optimally ordered triangular factorization. *J. Proc. IEEE*, 55:1801–1809, 1967.

[79] A. Trifunović : *Parallel Algorithms for Hypergraph Partitioning*. Thèse de doctorat, Imperial College London, février 2006.

[80] A. Trifunović et W. Knottenbelt : Par*k*way2.0 : A Parallel Multilevel Hypergraph Partitioning Tool. *In Proc. 19th International Symposium on Computer and Information Sciences*, volume 3280 de *Lecture Notes in Computer Science*, pages 789–800. Springer, 2004.

[81] R. van Driessche et D. Roose : A graph contraction algorithm for the calculation of eigenvectors of the laplacian matrix of a graph with a multilevel method. Rapport technique TW 209, Katholieke Universiteit Leuven, mai 1994.

[82] Jostle : Graph partitioning software. `http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/`.

[83] Y. Wan, S. Roy, A. Saberi et B. Lesieutre : A stochastic automaton-based algorithm for flexible and distributed network partitioning. *In Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pages 273–280, juin 2005.

[84] D. Whitley : The GENITOR algorithm and selection pressure : Why rank-based allocation of reproductive trials is best. *In* J. D. Schaffer, éditeur : *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121, San Mateo, CA, 1989. Morgan Kaufman.

[85] D. Whitley : A genetic algorithm tutorial. *Mathematics and Statistics*, 4:65–85, juin 1994.

[86] D. Whitley, S. Rana et R. B. Heckendorn : The island model genetic algorithm : On separability, population size and convergence. *Journal of Computing and Information Technology*, 7:33–47, 1999.

Annexe B

Liste des publications

[Che05] C. Chevalier : Towards parallel graph partitioning in Scotch. Poster, octobre 2005. Workshop on "Mesh creation, domain decomposition and parallel comp uting in 3D geophysics", Université de Pau, France.

[CP06a] C. Chevalier et F. Pellegrini : Improvement of the efficiency of genetic algorithms for scalable parallel graph partitioning in a multi-level framework. *In Proc. Europar, Dresden, LNCS 4128*, pages 243–252, septembre 2006.

[CP06b] C. Chevalier et F. Pellegrini : PT-Scotch : Un outil pour la renumérotation parallèle efficace de grands graphes dans un contexte multi-processeurs. *In Renpar 17 Proceedings*, octobre 2006.

[CP06c] C. Chevalier et F. Pellegrini : PT-Scotch : A tool for efficient parallel graph ordering. *In Proc. PMAA'2006, Rennes, France*, octobre 2006. `http://www.labri.fr/~pelegrin/papers/scotch_parallel_ordering_pmaa.pdf`.

[CP07a] C. Chevalier et F. Pellegrini : The PT-Scotch project : purpose, algorithms, first results. *In Proc. SIAM workshop on Combinatorial Scientific Computing, Costa Mesa, USA*, février 2007.

[CP07b] C. Chevalier et F. Pellegrini : The PT-Scotch project : purpose, algorithms, intermediate results. *In Proc. workshop on Combinatorial Tools for Parallel Sparse Matrix Computations, PPAM'07, Gdansk, Poland*, septembre 2007.

[CP07c] C. Chevalier et F. Pellegrini : PT-Scotch : A tool for efficient parallel graph ordering. *Parallel Computing*, 2007. Soumission en cours.