Adaptation au repartitionnement de graphes d'une méthode d'optimisation globale par diffusion

Sébastien Fourestier, François Pellegrini

▶ **To cite this version:**

Sébastien Fourestier, François Pellegrini. Adaptation au repartitionnement de graphes d'une méthode d'optimisation globale par diffusion. Rencontres Francophones du Parallélisme, May 2011, Saint-Malo, France. 8 p. hal-00648735

HAL Id: hal-00648735
https://hal.science/hal-00648735v1

Submitted on 6 Dec 2011

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.

RenPar'20 / SympA'14 / CFSE 8
Saint-Malo, France, du 10 au 13 mai 2011

Adaptation au repartitionnement de graphes d'une méthode d'optimisation globale par diffusion

Sébastien FOURESTIER* et François PELLEGRINI†

Université de Bordeaux
IPB, LaBRI et INRIA Bordeaux – Sud-Ouest
351, cours de la Libération, 33405 TALENCE, FRANCE
{fouresti|pelegrin}@labri.fr

Résumé
La résolution parallèle efficace de simulations numériques dont les coûts de calcul évoluent au cours du temps (telles que celles mettant en œuvre un remaillage dynamique) nécessite des méthodes efficaces de redistribution des données attribuées à chaque processeur. L'objet de cet article est d'étudier l'adaptation au repartitionnement de graphes, qui est un modèle courant du problème de redistribution des données, d'un ensemble d'algorithmes utilisés avec succès dans le cadre du partitionnement parallèle de graphes, et tout particulièrement d'un algorithme d'optimisation globale de la forme des partitions par diffusion. Le modèle expérimental exposé ici est séquentiel, mais les algorithmes présentés peuvent facilement être transposés en parallèle. Afin de valider notre approche, nous comparons nos résultats, implémentés au sein du logiciel SCOTCH, à la routine de repartitionnement mise en œuvre au sein du logiciel PARMETIS.

Mots-clés : repartionnement de graphes, raffinement par diffusion, partitionnement biaisé

1. Introduction

Du fait de la masse de calculs et de la quantité de données qu'elles mettent en jeu, les simulations scientifiques de grande taille ne peuvent plus se faire qu'en ayant recours au parallélisme. Pour cela, les données et les calculs qui leur sont associés sont répartis sur plusieurs unités de traitement interconnectées, qui travaillent en parallèle à la résolution du problème. Parce que les machines actuellement utilisées pour les simulations frontières possèdent plusieurs dizaines de milliers d'unités de traitement (appelées par la suite « processeurs », de façon générique), celles-ci ne peuvent accéder simultanément à la même mémoire, qui constituerait un goulot d'étranglement ; ces machines sont donc à mémoire distribuée, c'est-à-dire que chaque processeur dispose de sa propre mémoire, et peut échanger des informations avec ses pairs au moyen d'un réseau d'interconnexion dédié. L'utilisation efficace d'une telle machine parallèle suppose donc à la fois de répartir équitablement la charge de calcul entre les différents processeurs, et de minimiser le surcoût de communication induit par la nécessité, pour les différents processeurs, d'échanger des informations au cours de l'exécution du programme.
Lorsque les processus réalisant les calculs coexistent pendant toute la durée d'exécution du programme parallèle, le problème de répartition de la charge peut être modélisé sous la forme d'un problème de partitionnement d'un graphe non orienté, dont les sommets représentent les calculs à effectuer, et les arêtes, les interdépendances entre calculs. La répartition des calculs sur p processeurs s'exprime alors comme le fait de partitionner le graphe de calcul en p parties de même poids (pris comme la somme du poids des sommets qu'elles contiennent), de telle sorte que la coupe de ce graphe, c'est-à-dire la somme des poids des arêtes dont les extrémités appartiennent à deux parties différentes, soit minimale. On associe alors à chaque partie un processus exécutant tous les calculs attribués à cette partie, et hébergeant pour cela l'ensemble des données qui leur sont attachées. Dans le cas d'architectures hétérogènes et/ou hybrides

† Le travail de cet auteur est financé par une bourse de thèse CNRS-Région Aquitaine.
† Le travail de cet auteur a été partiellement soutenu par l'ANR dans le cadre du projet SOLSTICE (ANR-06-CIS6-010-01).

(mêlant mémoire distribuée et partagée), différents coûts de calcul et de communication doivent être considérés, et l'on parle alors plutôt de placement statique.
Dans de nombreux cas, la quantité de calculs à effectuer en chaque sommet du graphe peut varier dans le temps, ou bien des sommets peuvent être ajoutés ou enlevés (cas des raffinements et dé-raffinements de maillages, par exemple). De fait, l'équilibrage de charge initial peut devenir inadapté, et un nouveau partitionnement doit être calculé. L'enjeu de ce nouveau partitionnement est alors d'obtenir à nouveau un équilibrage de la charge, toujours en minimisant la taille de la coupe, mais de plus en minimisant également la quantité de données devant être redistribuées d'un processeur à un autre. Afin d'être applicable à des simulations (et donc des graphes) de grandes tailles, le repartitionnement de graphes doit lui-même être parallèle ; il sera alors effectué au cours de la simulation numérique, à intervalles réguliers ou lorsque le déséquilibre constaté sera trop important.
L'objet de cet article est d'étudier l'adaptation au repartitionnement d'un ensemble d'algorithmes mis en œuvre dans le logiciel PT-SCOTCH [5] dans le cadre du partitionnement simple. La section suivante est dédiée à un état de l'art des principales méthodes employées pour le repartitionnement. La section 3 présentera nos adaptations des algorithmes utilisés au sein du logiciel SCOTCH de partitionnement et de placement statique séquentiel de graphes, qui nous servira de modèle expérimental, afin de doter celui-ci de capacités de repartitionnement. Nous détaillerons ensuite dans la section 4 les résultats expérimentaux obtenus avec cette version, par rapport à ceux produits par le logiciel PARMETIS [11]. Nous conclurons par des considérations sur la transposition au cas parallèle de nos résultats.

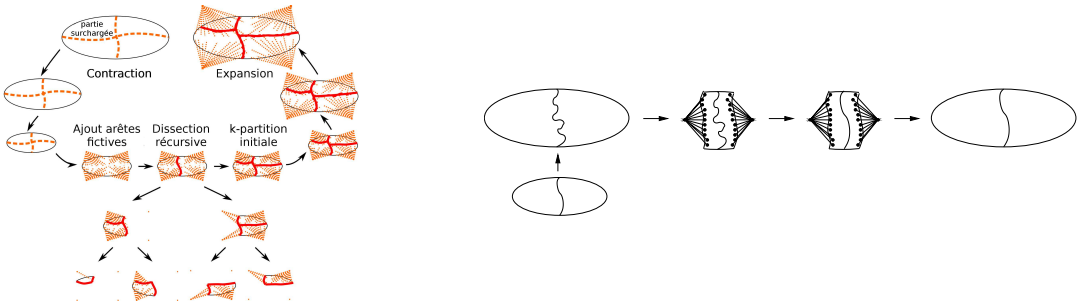
2. État de l'art

Le problème du repartitionnement de graphes a déjà été bien étudié. Comme le problème de partitionnement simple, il est NP-difficile [8], et ne peut être abordé, pour les tailles de problèmes qui nous intéressent (au delà du milliard de sommets), qu'au moyen de méthodes heuristiques. Parmi les algorithmes proposés (voir [3] et ses références), on peut distinguer deux classes principales. Les méthodes de type *scratch-remap* [14, 19] découplent le problème du rééquilibrage de la charge du problème de choix des sommets à migrer. Elles calculent un partitionnement simple du nouveau graphe, puis associent les nouvelles parties ainsi formées aux processus, en s'assurant que le nombre de sommets migrés soit minimal. Les méthodes de diffusion [6, 10, 18, 21, 22, 23], pour leur part, modifient itérativement le partitionnement existant, en migrant les sommets situés en bordure des parties les plus chargées vers leurs parties voisines, de proche en proche, jusqu'à ce que le déséquilibre ait suffisamment décru.
Les méthodes de type *scratch-remap* privilégient l'optimisation des indicateurs pris en compte lors des partitionnements simples, notamment l'équilibrage de la charge et la taille coupe, au détriment de la minimisation du nombre de sommets à migrer, qui peut être très important. Les méthodes de diffusion, du fait de leur caractère intrinsèquement local, donnent de bons résultats lorsque le nouveau graphe est proche de l'ancien, mais peuvent aboutir à une solution globalement non optimale en termes de coupe lorsque les modifications topologiques et de poids du graphe deviennent importantes. Ces méthodes sont, de plus, coûteuses en temps et difficilement parallélisables, que la migration soit réalisée de façon itérative [19], ou bien calculée au moyen d'un solveur d'optimisation linéaire [13].
Un compromis entre ces deux types de méthodes consiste à utiliser des algorithmes de partitionnement simple sur une version modifiée du graphe, qui intègre, sous forme de sommets et d'arêtes supplémentaires, des informations relatives aux coûts induits par la migration des sommets [2, 4, 20]. En pratique, on ajoute au graphe d'origine, pour chaque partie, un sommet ancre représentant cette partie, que l'on relie par des arêtes fictives à tous les sommets subsistant du partitionnement initial et appartenant à la partie considérée. Le poids des sommets ancres est considéré comme nul, afin qu'il n'ait pas d'impact sur le repartitionnement. Avec ce nouveau graphe, le fait de migrer un sommet vers une autre partie va induire une augmentation de la taille de la coupe égale au poids de l'arête fictive, et permet ainsi d'intégrer l'optimisation du nombre de sommets migrés à l'objectif de minimisation de la taille de la coupe. Le choix du poids des arêtes fictives par rapport à celui des arêtes d'origine définit le rapport entre le coût de migration d'un sommet et le coût de communication classique représenté par les poids des arêtes correspondant aux interdépendances entre les calculs.
Ces techniques de *partitionnement biaisé* (*skewed graph partitioning* en anglais), déjà utilisées dans le contexte du placement statique [9, 15], combinent flexibilité et élégance. Elles permettent, avec très peu de

modifications par rapport aux algorithmes de partitionnement simple, d'optimiser à la fois l'équilibrage de la charge entre parties, la taille de la coupe et la quantité de données devant être redistribuées d'un processeur à un autre. Notre travail se situe dans ce cadre.

3. Adaptation au repartitionnement des algorithmes existants

Comme la majorité des outils actuels de partitionnement de graphes, le logiciel SCOTCH [17], qui nous a servi de plate-forme de développement, s'appuie sur un schéma multi-niveaux [1]. Cette méthode consiste à contracter le graphe original en construisant une famille de graphes de tailles de plus en plus petites, au moyen d'appariements de sommets et de fusions d'arêtes, jusqu'à obtenir un graphe possédant à peine plus de sommets que de parties souhaitées, à calculer une partition initiale sur ce petit graphe, puis à la prolonger de proche en proche sur les graphes de plus grande taille jusqu'à obtenir une partition du graphe original. Les méthodes multi-niveaux sont couramment combinées à des méthodes d'optimisation locale, appliquées à chaque niveau aux partitions prolongées, afin que la granularité de la partition finale soit celle du graphe original et non celle du graphe le plus contracté.
Jusqu'à présent, SCOTCH calculait les k-partitions par bipartitionnement récursif, chaque bipartitionnement nécessitant le recours au schéma multi-niveaux. L'un des bénéfices de notre travail sur le repartitionnement a été de réaliser un module de k-partitionnement direct, où le schéma multi-niveaux n'est utilisé qu'une seule fois, comme d'autres outils le faisaient déjà.
Comme expliqué en section précédente, l'adaptation de SCOTCH au repartitionnement est réalisée en ajoutant au graphe à repartitionner des arêtes fictives qui vont induire une augmentation de la taille de la coupe à chaque fois qu'un sommet sera migré à l'extérieur de sa partie d'origine. L'ajout de ces arêtes fictives est réalisé à partir de l'ancienne partition, que l'utilisateur fournit en entrée avec le coût de migration. Afin que ce coût puisse être inférieur au poids minimal des arêtes du graphe d'origine représentant les inter-dépendances entre les calculs, souvent égal à 1, ce coût est de type flottant. Comme SCOTCH n'accepte que des poids entiers sur ses sommets et arêtes, le poids des arêtes représentant le coût de migration est ramené à un entier, en définissant un coefficient multiplicateur du poids des arêtes d'origine et un poids entier unique pour toutes les arêtes de migration.
L'adaptation du schéma multi-niveaux au repartitionnement a été réalisée comme suit (cf. figure 1 (a)). Lors de la phase de contraction, il faut propager l'information relative à l'ancien partitionnement jusqu'au graphe le plus grossier. Ceci peut être réalisé de deux manières. La première consiste à ajouter les arêtes fictives au graphe au début de la phase de contraction. Ceci accroît de manière importante le nombre d'arêtes à manipuler, rend la phase de contraction plus lente, et oblige à traiter les sommets ancres de façon spécifique. La deuxième consiste à n'ajouter les arêtes fictives que lorsqu'on réalise effectivement les calculs de repartitionnement. Dans ce cas, il faut imposer que l'appariement ne puisse se réaliser qu'entre sommets appartenant à la même partie, afin de conserver sur le graphe contracté toutes les informations de l'ancien partitionnement. C'est cette deuxième solution que nous avons retenue.
L'ajout des arêtes fictives sur le graphe le plus grossier permet de réutiliser sans modification les algorithmes de k-partitionnement par dissection récursive pour calculer le k-repartitionnement initial. Lors de la phase d'expansion, la k-partition calculée au niveau précédent est prolongée sur le graphe plus fin (chaque couple de sommets appariés est affecté à la même partie que le sommet qui le représentait dans le graphe plus grossier), puis des algorithmes d'optimisation locale sont appelés afin de lisser les frontières de la partition prolongée. Pour ce faire, SCOTCH dispose de deux méthodes principales : une version k-aire de l'heuristique de Fiduccia-Mattheyses [7] et une méthode diffusive. La première méthode, utilisée classiquement dans de nombreux partitionneurs, est une méthode itérative qui déplace les sommets situés à la frontière entre deux parties ou plus, en cherchant à minimiser le nombre d'arêtes coupées. Elle s'applique directement au graphe auquel ont été ajoutées les arêtes fictives, que ce soit le graphe original ou un sous-ensemble de ce dernier (voir ci-dessous). L'inconvénient de cette méthode, outre qu'elle est intrinsèquement séquentielle et se parallélise donc mal, est qu'elle est locale. Elle conduit donc souvent à des frontières constituées de segments de coupe localement optimale, mais dont la juxtaposition n'est pas globalement optimale. C'est pour remédier à ce problème, et considérer de façon globale le problème de lissage des frontières, qu'une méthode diffusive, appelée de façon imagée le « tonneau des Danaïdes », a été mise en œuvre au sein du logiciel SCOTCH [16].
Le principe de cet algorithme consiste à représenter le graphe sous la forme d'un ensemble de barriques,



(a) (b)

FIG. 1 – (a) Adaptation du schéma multi-niveaux au repartitionnement. (b) Méthode de raffinement multi-niveaux par bande, dans le cas de deux parties. Un graphe bande de faible largeur est créé autour du séparateur prolongé, tous les sommets restants de chaque partie étant contractés en un unique sommet ancre. Après qu'un algorithme d'optimisation locale a été appliqué, le séparateur bande raffiné est prolongé sur l'ensemble du graphe, puis le processus d'expansion se poursuit au niveau supérieur.

décrivant les sommets, reliés par des tuyaux, les arêtes, et à alimenter chaque partie avec des liquides de différentes couleurs tels que deux liquides différents se détruisent lorsqu'ils se rencontrent afin de déterminer quel liquide dominera dans chacune des barriques. Du fait que nous ne souhaitons utiliser cet algorithme que pour l'optimisation locale de la partition prolongée, seul le voisinage immédiat des frontières entre parties nous intéresse. Nous restreignons donc son usage à un *graphe bande*, défini par l'ensemble des sommets du graphe qui sont à une distance au plus d de la frontière et par l'ensemble des arêtes reliant ces sommets entre eux, auxquels on ajoute pour chaque partie un *sommet ancre* de poids égal à la somme des poids des sommets de la partie qui ne sont pas dans le graphe bande, et relié par des arêtes de poids 1 à chacun des sommets de sa partie qui sont à distance d de la frontière, appelés les sommets de dernier niveau (voir la figure 1 (b)). C'est à partir des sommets ancres que les liquides seront injectés dans le réseau de barriques.
Dans le cadre du repartitionnement, nous ajoutons à chaque sommet une nouvelle arête, de poids correspondant au coût de migration du sommet[3], le reliant à sa partie d'origine ; si cette arête existe déjà (c'est-à-dire que le sommet est un sommet de dernier niveau), le coût de migration est ajouté au poids de l'arête existante. Ainsi, tout sommet, qu'il soit à la frontière ou de dernier niveau, recevra une quantité de liquide modélisant le fait qu'il a une affinité plus grande avec sa partie d'origine. Grâce à l'usage du graphe bande, le nombre d'arêtes créées est bien plus faible que si l'on avait considéré le graphe entier.
L'adaptation de cet algorithme au repartitionnement fait apparaître un cas extrême lorsque le nouveau graphe comporte un sommet dont le poids est supérieur à deux fois la taille optimale d'une partie. Dans ce cas, l'algorithme de partitionnement initial du graphe le plus grossier, qui opère par bipartitionnement récursif, fournira un partitionnement laissant au moins un processeur inoccupé. Ceci n'est pas déraisonnable, car cela ne changera rien au temps d'exécution qui est dominé par le processeur hébergeant la charge de calcul du plus gros sommet, et permet de diminuer de manière importante la taille de la coupe par rapport au fait de répartir les sommets restants sur l'ensemble des processeurs. Dans le cas d'un tel scénario, est-il pertinent de créer des sommets ancres pour les parties n'apparaissant pas dans le nouveau partitionnement ? Si l'on choisit d'ajouter ces sommets sources, le raffinement par diffusion aura tendance à ramener quelques sommets de chaque partie disparue vers leur partie d'origine, ce qui aboutira à l'apparition sur le graphe bande de petits îlots qui impliqueront une mauvaise coupe pour un gain d'équilibrage de la charge très faible. Si l'on considère la méthode de diffusion comme une méthode de raffinement, elle ne doit pas remettre en cause le choix initial de limiter le nombre de processeurs utilisés. Nous avons donc fait en sorte de ne pas modéliser sur les graphes bandes les parties disparues, et donc de ne pas prendre en compte les coûts de migration des sommets ayant appartenu à

[3] Le coût de migration d'un sommet est égal au coût de migration fourni par l'utilisateur multiplié par le nombre de sommets qui ont été fusionnés lors de la phase contraction pour obtenir le sommet courant.

TAB. 1 – Description des graphes de test que nous avons utilisés. Ces derniers correspondent à des problèmes 3D. |V| et |E| sont respectivement les poids totaux des sommets et des arêtes, en milliers.

Graphe	Taille ($\times 10^3$)		Degré moyen	Graphe	Taille ($\times 10^3$)		Degré moyen
	|V|	|E|			|V|	|E|	
`oilpan`	74	1762	47.77	`audikw1`	944	38354	81.28
`fcondp2`	202	5546	54.96	`ldoor`	952	22785	47.86
`troll`	213	5886	55.15	`conesphere1m`	1055	8023	15.21
`bmw32`	227	5531	48.65				

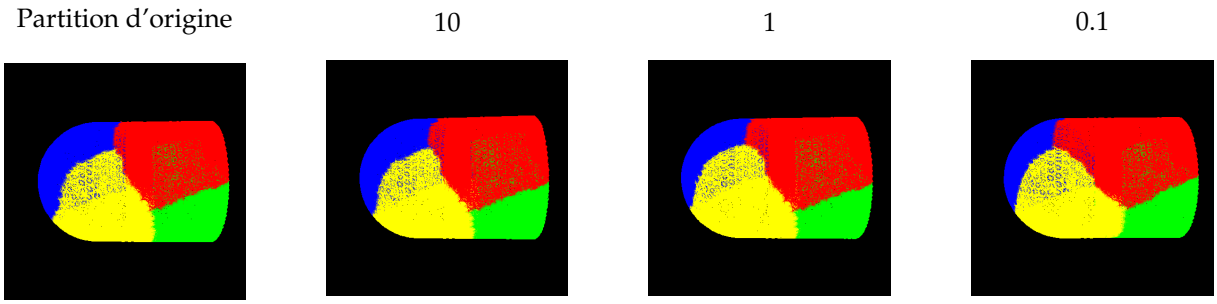


FIG. 2 – Le graphe `altr4` repartitionné avec SCOTCH en prenant comme coût de migration, les valeurs 0.1, 1 et 10.

ces dernières.

4. Résultats expérimentaux

Les résultats de cet article ont été calculés sur les nœuds du cluster PLAFRIM, comprenant chacun deux processeurs quad-core Nehalem Intel® Xeon® X5550 à 2,66 GHz et 24 Go de mémoire.
Un ensemble représentatif des graphes que nous avons utilisés au cours de nos expérimentations est présenté dans le tableau 1. Notre méthodologie de test du repartitionnement est la suivante. Ces graphes, dont tous les sommets et arêtes ont un poids initial égal à 1, ont été partitionnés en 128 parties au moyen de la stratégie actuelle de partitionnement de SCOTCH, à savoir le bipartitionnement récursif utilisant le schéma multi-niveaux. Puis, les graphes ont été modifiés en attribuant au sommet 0 un nouveau poids égal à $\frac{|V|-1}{2\times(128-1)}$, c'est-à-dire la moitié du poids qui serait optimal pour les 127 autres parties si le sommet avait ce poids ; cette perturbation élémentaire induit un déséquilibre moyen par partie de l'ordre d'un pourcent, par rapport à la nouvelle taille optimale de ces dernières si l'on considère le partitionnement réalisé précédemment. Les graphes ont ensuite été repartitionnés au moyen de la routine `SCOTCH_graphRemap`, avec une stratégie combinant les algorithmes décrits précédemment, à savoir le schéma multi-niveau k-aire utilisant comme raffinement local le raffinement par diffusion sur graphe de bande suivi de l'heuristique de Fiduccia-Mattheyses k-aire. La figure 2 montre le type de résultats obtenus pour quatre parties sur le graphe `altr4`.
Pour les graphes testés selon la méthode ci-dessus, nous mesurons la qualité des partitions en termes de taille de la coupe résultante, de taux de déséquilibre de charge entre parties, et de pourcentage d'arêtes migrées, pour des coûts de migration variant de 0,1 à 50[4] et un déséquilibre résiduel souhaité de 1 %.
Afin d'évaluer la qualité de nos méthodes, les heuristiques de repartitionnement de SCOTCH 5.3 (BETA) sont comparées à celles du logiciel concurrent PARMETIS 3.1.1 [12], utilisé sur deux processeurs, la version séquentielle du logiciel METIS ne disposant pas de ces fonctionnalités. Pour effectuer ces comparaisons, nous avons utilisé la routine `ParMETIS_V3_AdaptiveRepart`. Le coût de migration m de

[4] Nous avons fait croître ce coût avec un pas de 0,01 lorsqu'il est inférieur à 1, et avec un pas de 1 ensuite.

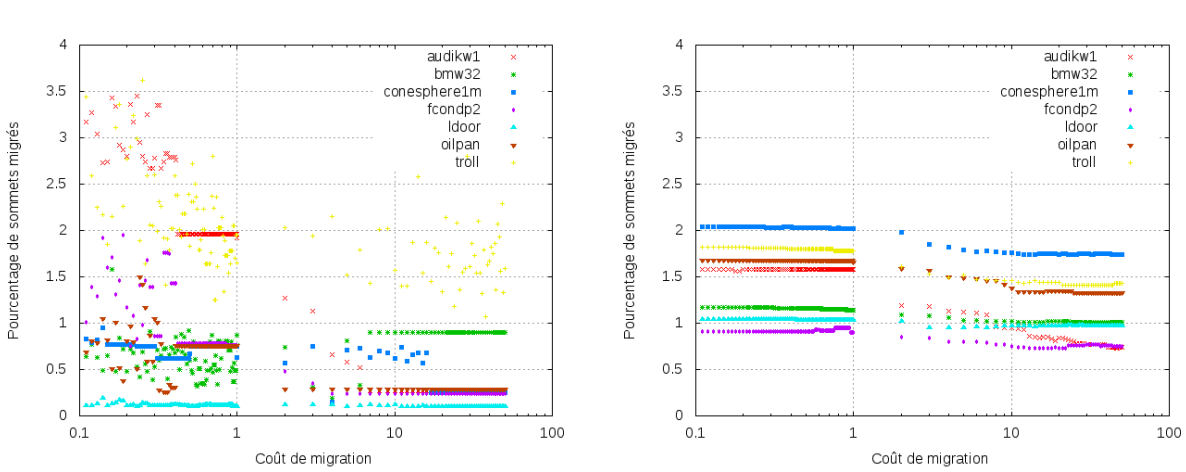


FIG. 3 – Pourcentage de sommets migrés en fonction du coût de migration pour SCOTCH et PARMETIS.

SCOTCH a été converti pour correspondre au paramètre `itr` pris en entrée par PARMETIS, selon la formule $\texttt{itr} = \frac{1}{m}$; de même, le taux de déséquilibre souhaité de 0,01 fourni à SCOTCH s'est traduit par le remplissage du tableau `ubvec` (contenant les déséquilibres souhaités par partie) avec la valeur 1,01.
La figure 3 présente, pour SCOTCH à gauche et PARMETIS à droite, le pourcentage de sommet migrés pour chaque graphe et chaque coût de migration. Il ressort de ces courbes que SCOTCH est plus sensible à la variation du coût de migration que PARMETIS, qui présente un effet de plateau très net et ne descend pas en dessous de 0,75 %, alors que SCOTCH peut descendre sous les 0,25 %. On remarque également la plus grande dispersion des résultats obtenus avec SCOTCH pour des coûts de migration proches. Les oscillations des résultats pour les faibles valeurs de coût s'expliquent par le fait que SCOTCH utilise une combinaison d'algorithmes explorant chacun de manière différente l'espace des solutions. Alors que l'algorithme de diffusion a comme fonction principale la minimisation de la taille de la coupe, l'algorithme de Fiduccia-Mattheyses cherchera pour sa part à réduire le déséquilibre de la charge. Pour de faibles valeurs du coût de migration, cette combinaison se trouve dans un état méta-stable, où une modification minime de la frontière peut entraîner l'algorithme de Fiduccia-Mattheyses à vouloir se comporter comme un algorithme *scratch-remap* qui migrera alors beaucoup de sommets, la pénalité de migration étant trop faible pour l'en empêcher.
La figure 4 représente la relation entre l'évolution du déséquilibre (à gauche) et de taille de la coupe des partitions (à droite) en fonction du pourcentage de sommets migrés. Chaque nuage de points représente les mesures effectuées pour un graphe donné avec les différentes valeurs de coût de migration. Chaque taux, en abscisse comme en ordonnée, est calculé à partir des valeurs obtenues pour SCOTCH (x_s) et PARMETIS (x_p), selon la formule : $(\frac{x_s}{x_p} - 1) \times 100$. Remarquons tout d'abord qu'en accord avec ce que nous avons exposé précédemment, SCOTCH peut migrer jusqu'à 1,2 fois plus de sommets que PARMETIS. Alors que ce dernier tend à fournir des partitions plus équilibrées, quel que soit le nombre de sommets migrés, SCOTCH obtient en moyenne une coupe de taille 6 % plus petite. Pour la majorité des graphes, on remarque que les nuages de points se regroupent autour de segments obliques dirigés vers les bas droits des figures. Cela nous indique que plus SCOTCH migre un pourcentage important de sommets par rapport à PARMETIS, plus il tend à avoir un meilleur équilibrage et une meilleure coupe que ce dernier. Pour certains graphes, on peut avoir deux droites de pente différente ; ces dernières correspondent à des points consécutifs et le changement de pente s'effectue lorsque le coût de migration est de l'ordre des poids des sommets du graphe[5].
Le temps d'exécution moyen de SCOTCH en séquentiel est de 9,4 s, et celui de PARMETIS est de 2,6 s sur

[5] L'explication des comportements obtenus avec les graphes `ldoor` (droite verticale) et `troll` (déséquilibre important) ne semble pas découler des caractéristiques de ces graphes, et nécessite une étude plus approfondie.

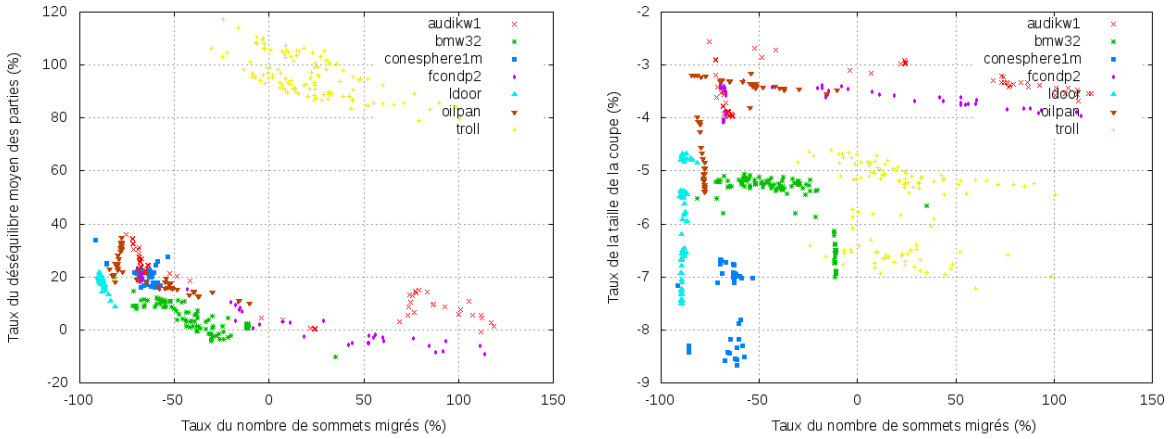


FIG. 4 – Mise en regard des gains apportés par SCOTCH par rapport à PARMETIS en termes de déséquilibre moyen des parties (figure de gauche) et de taille de la coupe (figure de droite) par rapport au nombre de sommets migrés.

deux processeurs. Ainsi, dans les conditions expérimentales qui sont celles de cette étude, SCOTCH est en moyenne 4,5 fois plus lent que PARMETIS (avec un écart-type de 1,7). Ce surcoût tient tant au gain apporté par le parallélisme[6] qu'au surcoût induit par l'utilisation conjointe de l'algorithme de diffusion, servant à lisser globalement la frontière, et de l'algorithme de Fiduccia-Mattheyses k-aire, servant à lisser localement les artefacts.

Pour résumer, nous avons constaté que SCOTCH est plus sensible aux variations du coût de migration, et qu'il profite des faibles valeurs de ce taux pour migrer plus de sommets afin d'améliorer le déséquilibre et la taille de la coupe. Alors que PARMETIS a tendance à calculer des partitions mieux équilibrées en un temps plus petit, l'utilisation de SCOTCH permet d'obtenir une meilleure taille de coupe.

5. Conclusion

Dans cet article, nous avons montré comment biaiser différents algorithmes de partitionnement de graphes afin de modéliser le problème du repartitionnement. En particulier, nous avons montré comment adapter à cette problématique un algorithme d'optimisation globale par diffusion, utilisé dans un contexte local par son confinement au sein de graphes bandes.

Cet algorithme présente de nombreuses qualités, qui en font, comme pour le partitionnement simple, un bon candidat à la parallélisation, alors que les algorithmes classiques d'optimisation locale de type Fiduccia-Mattheyses se parallélisent mal (mais peuvent être utilisés séquentiellement lors des premiers niveaux de l'expansion, lorsque les graphes contractés sont de tailles suffisamment petites). Les résultats produits ont permis de mettre en avant la bonne sensibilité de nos algorithmes au coût de migration et leur tendance, en comparaison avec PARMETIS, à privilégier l'optimisation de la taille de la coupe plutôt que la réduction du déséquilibre.

La transposition au cas parallèle de nos travaux est en cours, au sein du logiciel PT-SCOTCH.

Remerciements

Les résultats présentés dans cet article ont été obtenus sur la plate-forme de test PLAFRIM mise en place dans le cadre de l'*action de développement INRIA PlaFRIM* avec le soutien du LaBRI, de l'IMB et des entités suivantes : le Conseil Régional d'Aquitaine, le FeDER, l'Université de Bordeaux et le CNRS (voir `https://plafrim.bordeaux.inria.fr/`).

[6] Il ne serait pas honnête de multiplier par deux le temps de PARMETIS pour le normaliser sur un processeur, car cela supposerait une scalabilité idéale qui nous avantagerait. Il est regrettable que PARMETIS ne puisse s'exécuter sur un unique processeur.

Bibliographie

1. S. T. Barnard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency : Practice and Experience*, 6(2) :101–117, 1994.
2. Umit Catalyurek, Erik G. Boman, Karen D. Devine, Doruk Bozdag, Robert Heaphy, and Lee Ann Riesen. A repartitioning hypergraph model for dynamic load balancing. Tech. Report SAND2008-2304J, Sandia National Laboratories, Albuquerque, NM, 2008.
3. Umit Catalyurek, Doruk Bozdag, Erik G. Boman, Karen D. Devine, Robert Heaphy, and Lee Ann Riesen. Hypergraph-based dynamic partitioning and load balancing. Tech. Report SAND2007-0043P, Sandia National Laboratories, Albuquerque, NM, 2007.
4. U.V. Catalyurek, E.G. Boman, K.D. Devine, D. Bozdag, R.T. Heaphy, and L.A. Riesen. Hypergraph-based dynamic load balancing for adaptive scientific computations. In *Proc. of 21st International Parallel and Distributed Processing Symposium (IPDPS'07)*. IEEE, 2007.
5. C. Chevalier and F. Pellegrini. PT-SCOTCH : A tool for efficient parallel graph ordering. *Parallel Computing*, 34 :318–331, 2008.
6. G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.*, 7 :279–301, October 1989.
7. C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. 19th Design Automat. Conf.*, pages 175–181. IEEE, 1982.
8. M. R. Garey and D. S. Johnson. *Computers and Intractablility : A Guide to the Theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.
9. B. Hendrickson, R. Leland, and R. Van Driessche. Skewed graph partitioning. In *Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing*. IEEE, March 1997.
10. Y. F. Hu, R. J. Blake, and D. R. Emerson. An optimal migration algorithm for dynamic load balancing. concurrency : Practice and experience, 1998.
11. G. Karypis and V. Kumar. PARMETIS *Parallel Graph Partitioning and Sparse Matrix Ordering Library*. U. Minnesota, C.S.& E. Dept., AHPCRC, August 2003.
12. METIS : Family of multilevel partitioning algorithms. `http ://glaros.dtc.umn.edu/gkhome/views/metis`.
13. Henning Meyerhenke, Burkhard Monien, and Thomas Sauerwald. A new diffusion-based multilevel algorithm for computing graph partitions. *JPDC*, 69 :750–761, September 2009.
14. Leonid Oliker and Rupak Biswas. Plum : Parallel load balancing for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 52 :150–177, 1998.
15. F. Pellegrini. Static mapping by dual recursive bipartitioning of process and architecture graphs. In *Proc. SHPCC'94*, pages 486–493. IEEE, May 1994.
16. F. Pellegrini. A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries. In *Proc. Euro-Par'07, LNCS 4641*, pages 191–200, August 2007.
17. SCOTCH : Static mapping, graph partitioning, and sparse matrix block ordering package. `http ://www.labri.fr/~pelegrin/scotch/`.
18. Kirk Schloegel, George Karypis, and Vipin Kumar. Parallel multilevel diffusion algorithms for repartitioning of adaptive meshes, 1997.
19. Kirk Schloegel, George Karypis, and Vipin Kumar. Wavefront diffusion and lmsr : Algorithms for dynamic repartitioning of adaptive meshes. *Trans. Parallel Distrib. Syst.*, 12 :451–466, May 2001.
20. C. Walshaw. Variable partition inertia : graph repartitioning and load-balancing for adaptive meshes. In S. Chandra M. Parashar and X. Li, editors, *Advanced Computational Infrastructures for Parallel and Distributed Adaptive Applications*. Wiley, New York, 2010. (Invited chapter).
21. C. Walshaw and M. Cross. Parallel optimisation algorithms for multilevel mesh partitioning. *Parallel Computing*, 26(12) :1635–1660, 2000.
22. C. Walshaw and M. Cross. Dynamic Mesh Partitioning and Load-Balancing for Parallel Computational Mechanics Codes. In B. H. V. Topping, editor, *Computational Mechanics Using High Performance Computing*, pages 79–94. Saxe-Coburg Publications, Stirling, 2002. (Invited chapter).
23. C. Walshaw, M. Cross, and M. G. Everett. Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes. *J. Parallel Distrib. Comput.*, 47(2) :102–108, 1997.