Partitionnement multi-critères de graphes pour l'équilibrage de charge de simulations multi-physiques

Rémi Barat, Cédric Chevalier, François Pellegrini

Compas'2016 : Parallélisme / Architecture / Système
Lorient, France, du 5 au 8 juillet 2016

Partitionnement multi-critères de graphes pour l'équilibrage de charge de simulations multi-physiques

Rémi Barat[1], Cédric Chevalier[1] et François Pellegrini[2]

[1]CEA, DAM, DIF, F–91297 Arpajon, France, {remi.barat.ocre | cedric.chevalier}@cea.fr
[2]LaBRI & Inria Bordeaux Sud-Ouest, Université de Bordeaux, 33400 Talence, France, francois.pellegrini@labri.fr

---

Résumé
Les simulations numériques de grande taille nécessitent d'être effectuées en parallèle. L'équilibre de la charge entre les nœuds de calcul est alors primordial. Dans le cas de simulations multi-physiques, on modélise le problème d'équilibrage de charge par celui du partitionnement multi-critères de graphes, problème NP-Difficile pour lequel les outils existants reposent sur des heuristiques largement améliorables. Nous présentons ici une méthode de partitionnement multi-critères de graphe utilisant un schéma multi-niveaux adapté. Notamment, nous proposons un nouvel algorithme de partitionnement initial et nous démarquons de l'existant concernant la phase de raffinement, conçue pour respecter strictement les tolérances d'équilibre. En mono-critère, notre méthode est compétitive par rapport à des logiciels de référence comme Scotch et MeTiS. En multi-critères, contrairement à MeTiS, notre méthode retourne dans tous les cas des solutions valides par rapport aux tolérances prescrites.

Mots-clés : partitionnement, graphe, multi-critères, multi-niveaux, optimisation locale

---

1. Introduction

Les simulations numériques traitent des données de plus en plus volumineuses et sont de plus en plus complexes. Afin de les effectuer dans des temps raisonnables, on a recours au parallélisme. En mémoire distribuée, tirer pleinement partie du parallélisme nécessite d'équilibrer la charge de calcul entre les nœuds de calcul.
Pour les codes de simulation utilisant des maillages, on attribue à chaque maille un poids indiquant sa complexité de traitement, puis on cherche une partition du maillage équilibrant le poids de chaque partie. Le calcul d'une maille nécessitant les données de ses voisines, la partition doit par ailleurs minimiser le coût induit par les communications.
Certaines simulations dites multi-physiques couplent plusieurs modèles physico-numériques. Au sein d'une même boucle temporelle, plusieurs phases de calcul sont ainsi effectuées (voir Algorithme 1). Le poids d'une maille n'est alors plus scalaire mais vectoriel, avec une composante par phase de calcul. Il serait possible de calculer plusieurs partitionnements, correspondant à chaque phase de calcul, mais cela nécessiterait d'actualiser la distribution des données entre chaque phase de calcul, et donc d'augmenter les communications. La plupart du temps, il est plus rentable de trouver une partition dite multi-critères, équilibrant la charge de calcul pour toutes les phases en même temps.

Algorithme 1 Principe de fonctionnement d'une simulation numérique à deux phases de calculs, f et g, couplant les variables X et Y issues de deux modèles différents.

pour $t \in [0, t_{fin} - 1]$ **faire** # Boucle temporelle
  $X(t+1) \leftarrow f(t+1, X(t), Y(t))$ # Première phase de calcul
  $Y(t+1) \leftarrow g(t+1, X(t+1), Y(t))$ # Seconde phase de calcul
fin pour

Cependant, il existe actuellement peu de solutions pratiques pour résoudre le problème de partitionnement multi-critères de graphes. Il s'agit de la généralisation d'un problème NP-difficile, et la plupart des outils ont été conçus uniquement pour le mono-critère. Dans un premier temps, nous rappelons les principales heuristiques utilisées pour le partitionnement de graphes, puis nous proposons des modifications pour leur bon fonctionnement dans un contexte multi-critères. Enfin, nous présentons les résultats de cette nouvelle approche en les confrontant à deux partitionneurs couramment utilisés, MeTiS et Scotch.

2. Contexte

Il existe différentes manières de modéliser le problème de l'équilibrage de charge d'un code de simulation basé sur un maillage [3, 6, 7]. Pour prendre en compte explicitement la minimisation du coût de communication, le problème peut être formulé comme un problème de partitionnement de graphe. Ce problème est NP-difficile, mais des heuristiques efficaces ont été proposées, notamment celles basées sur une approche multi-niveaux, utilisée par la plupart des outils.
L'algorithme multi-niveaux [1] est une méthode en trois étapes, contraction, partitionnement initial et expansion. La phase de contraction permet de réduire la taille du graphe considéré en conservant ses propriétés topologiques, par fusion de sommets voisins. On réalise autant de niveaux de contraction successifs que nécessaire à ce que le graphe atteigne une taille suffisamment faible (par exemple : moins de 120 sommets dans Scotch [11]), puis on partitionne le graphe le plus grossier à l'aide d'une méthode de partitionnement initial. Ce partitionnement est successivement prolongé au niveau supérieur puis raffiné par une méthode d'optimisation locale, et ainsi à chaque niveau jusqu'à revenir au graphe original (voir Figure 1).

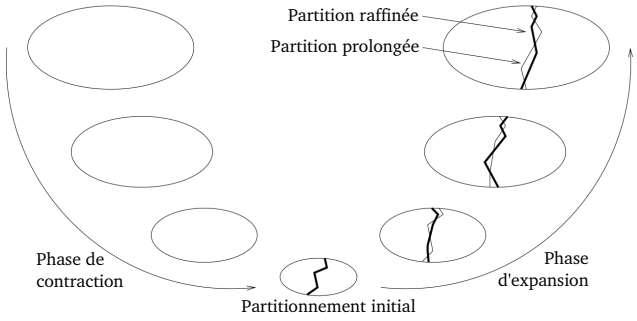


FIGURE 1 – Schéma global du multi-niveaux, illustrant les trois phases : contraction, partitionnement initial et expansion.

Une méthode de partitionnement initial classique en mono-critère est le partitionnement par expansion de région [8]. Elle vise à obtenir des parties connexes et équilibrées par agrégation

de sommets, à partir de graines choisies aléatoirement. Cependant, nos expérimentations nous ont montré qu'en multi-critères l'équilibrage est très difficile à obtenir par cette méthode. En effet, il est possible qu'il soit plus difficile de trouver des partitions connexes et équilibrées en multi-critères.
Le partitionnement initial permet donc de partitionner le graphe contracté le plus grossier. Néanmoins, cette partition n'est pas optimale et peut facilement être améliorée par une méthode d'optimisation locale.
Le principe d'une méthode d'optimisation locale est, à partir d'un partitionnement initial, de changer successivement de partie un sommet dès lors qu'on se rapproche de l'objectif (ici réduire le volume de communication). Un changement de partie s'appelle un *mouvement*, et on appelle *gain* le coût des communications évitées grâce à ce mouvement. Les différentes heuristiques d'optimisation locales se différencient par :

- le fait d'autoriser ou non des mouvements qui ne respectent pas les contraintes (ici, cela signifie dépasser le seuil de tolérance de déséquilibre pour au moins un critère) ;
- le fait d'autoriser ou non des mouvements de gain nul, voire négatif.

L'algorithme 2 détaille la méthode d'optimisation locale la plus utilisée en partitionnement de graphe : l'algorithme de Fiduccia-Mattheyses (FM) [4].

Algorithme 2 L'algorithme de Fiduccia-Mattheyses

```
Entrée : p partitionnement équilibré du graphe considéré
  répéter                                                  # Effectuer une passe
    Déverrouiller tous les sommets
    Calculer les gains pour tous les mouvements
    tant que il reste des mouvements possibles faire
      Effectuer le meilleur mouvement possible
      Verrouiller le sommet déplacé
      Enregistrer la configuration et son coût
      Mettre à jour les gains des sommets voisins du sommet déplacé
    fin tant que
    Restaurer l'état de coût minimal trouvé dans la précédente boucle
  jusqu'à la passe ne fait pas décroître le coût
  retourner La partition de l'état courant
```

L'algorithme FM effectue des passes (boucle externe), bouclant tant que la passe précédente réussit à améliorer la qualité du partitionnement. Une passe consiste en une suite de mouvements (boucle interne). Une fois qu'un sommet a été changé de partie, on le verrouille jusqu'à la fin de la passe : cela évite des mouvements contradictoires au sein d'une même passe. Lorsqu'il n'y a plus de mouvements possibles (soit parce que tous les sommets sont verrouillés, soit parce que les mouvements restants ne respectent pas les contraintes), on revient à l'état où le volume de communications était le plus faible, avant de recommencer une nouvelle passe.

3. Méthodes de partitionnement multi-critères

Notre méthode de partitionnement multi-critères utilise une approche multi-niveaux munie d'une méthode de contraction et d'un algorithme de Fiduccia-Mattheyses adapté. Concernant le partitionnement initial, nous proposons une méthode se focalisant sur l'obtention d'une partition équilibrée. En effet, les méthodes de partitionnement initial multi-critères existantes [9]

dérivent des méthodes mono-critère et peinent à obtenir des solutions équilibrées.

3.1. Contraction

La phase de contraction consiste à agréger des sommets du graphe. Nous n'avons pas fait de modification de cette phase par rapport au partitionnement mono-critère, c'est-à-dire que nous utilisons une variante du *Heavy Edge Matching*.

3.2. Partitionnement initial

L'objectif principal du partitionnement initial est de trouver une solution qui respecte les contraintes d'équilibre. Ce problème est analogue à celui du partitionnement de nombres [10, 2], qui est NP-complet [5]. Notre méthode de résolution part d'une répartition aléatoire des sommets et utilise une politique d'optimisation locale afin d'obtenir une partition équilibrée. Ainsi, le gain d'un mouvement est déterminé par le gain qu'il apporte à l'équilibre. Notamment, si d_{max} est le déséquilibre maximal relatif, alors le gain d'un mouvement est la réduction de d_{max} qu'il entraîne (voir Figure 2). Un sommet est changé de partie dès lors que son gain est strictement positif (c'est-à-dire que ce mouvement permet de réduire d_{max}). On s'arrête dès lors qu'aucun sommet ne vérifie cette condition.

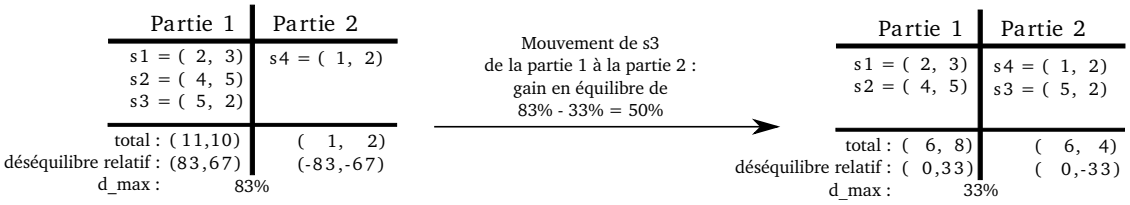


FIGURE 2 – Un exemple du calcul de gain dans la recherche d'un partitionnement initial. Le mouvement de s3 de la partie 1 à 2 réduit le déséquilibre maximal de 83% sur le premier critère à 33% sur le second critère. Ce changement du critère le moins bien équilibré est représentatif de la difficulté du problème.

d_{max} diminuant strictement à chaque mouvement, l'algorithme est garanti de terminer et de renvoyer une solution de déséquilibre au pire égal à celui du partitionnement donné en entrée. Si l'algorithme est bloqué (plus de mouvement de gain strictement positif) avant d'avoir atteint une solution respectant les contraintes d'équilibre, alors il est relancé à partir d'une autre répartition aléatoire des sommets. En pratique notre algorithme permet de trouver facilement et rapidement un partitionnement de déséquilibre inférieur à la tolérance fixée par l'utilisateur. Sa qualité sera améliorée par un raffinement.

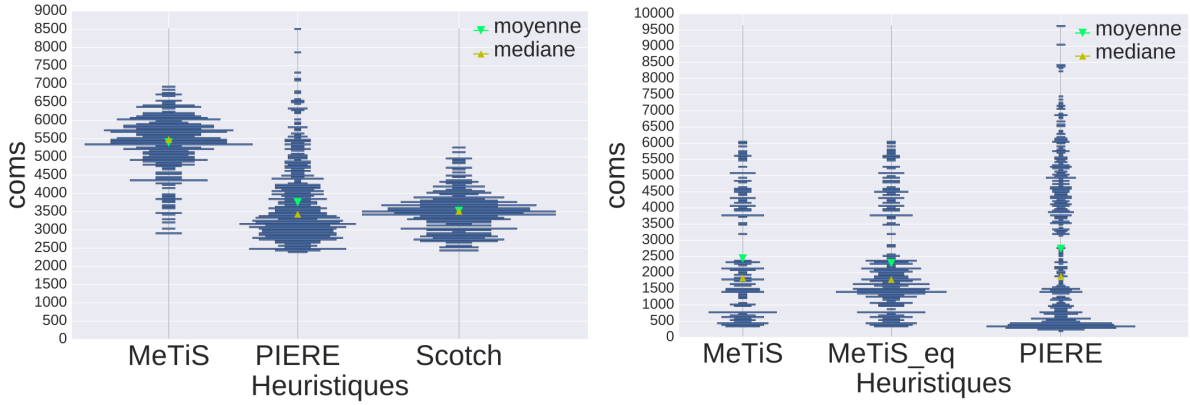
3.3. Raffinement

Lors de la phase d'expansion, notre algorithme d'optimisation locale en multi-critères est une adaptation de l'algorithme de Fidduccia-Mattheyses décrit en Section 2. Sa spécificité est de ne jamais autoriser de mouvement entraînant un déséquilibre qui dépasserait le seuil de tolérance fixé par l'utilisateur, et ce pour tous les critères. Ainsi, à l'issue de la phase de raffinement, le partitionnement obtenu respecte toujours les contraintes d'équilibre, ce qui constitue une différence fondamentale avec les algorithmes de raffinement ayant besoin d'une phase de rééquilibrage, comme celui de MeTiS [9]. En effet, notre heuristique de partitionnement initial fournissant une solution respectant les contraintes d'équilibre et notre algorithme d'optimisation locale ne permettant pas de violer ces contraintes, notre solution finale est garantie de les satisfaire.

4. Expérimentations et résultats

Nous avons implanté notre algorithme présenté dans la section précédente par un prototype en langage python appelé « PIERE » (Partitionnement Initial Équilibré et Raffinement Équilibré), et l'avons comparé à deux logiciels de partitionnement de graphes, Scotch et MeTiS.
Nous présentons ici les résultats que nous avons obtenus sur un maillage de 3584 triangles [1].
Nous avons généré deux répartitions de poids : une mono-critère et une avec trois critères. Leurs distributions correspondent à l'exécution de code monte carlo de simulation de physiques des particules.
Sur chacune des deux instances, chaque algorithme a été lancé 500 fois, à chaque fois en numérotant les sommets du graphe aléatoirement. En effet, changer l'ordre de parcours des sommets peut mener à une solution différente, car lors de la phase de contraction les agrégats de sommets seront modifiés, et lors de la phase de raffinement l'ordre dans lequel les mouvements de même gain sont effectués peut varier.
Pour chaque exécution, lorsque la solution renvoyée respecte les contraintes d'équilibre, nous enregistrons sa qualité. Nous ne comparons pas les temps de calcul, notre algorithme PIERE n'étant codé que sous la forme d'un prototype python alors que les Scotch et Metis sont des outils optimisés écrits en C. De plus, les complexités en temps des algorithmes étant semblables, une implantation efficace de PIERE dans Scotch devrait être réalisée prochainement.
Les résultats obtenus sont présentés en Figure 3. À chaque instance correspond un ensemble d'histogrammes. Un histogramme présente les résultats d'une heuristique pour cette instance. En ordonnée figure le volume de communications. La longueur de la barre à l'ordonnée y pour l'algorithme A est proportionelle au nombre de fois que le partitionnement trouvé par A engendre y communications. Pour chaque algorithme, nous faisons également figurer la moyenne (représentée par un triangle vert orienté vers le bas) et la médiane (triangle jaune orienté vers le haut). Cette représentation des données permet de visualiser facilement les différences de comportement des partitionneurs, en analysant la distribution de la qualité des partitions calculées.
Pour les deux tests et pour chaque algorithme, on remarque que les résultats varient grandement suivant les exécutions (rappelons que la seule différence entre deux exécutions est la numérotation des sommets du graphe), et ce sur un maillage relativement petit et topologiquement simple. Par exemple, l'écart entre les meilleures et les pires solutions trouvées par Scotch varie du simple au double. Ceci rappelle d'abord la complexité du problème : si de nombreuses solutions valides existent, nos algorithmes peinent à toujours renvoyer celles qui sont proches de l'optimum. Ensuite, ces variations montrent l'importance de la méthode de comparaison des algorithmes. Notamment, la moyenne n'est pas le seul facteur à prendre en compte : il est essentiel de faire apparaître la dispersion d'une heuristique, ainsi que la propension qu'elle a à trouver une solution de bonne qualité ou, au contraire, à renvoyer une solution de mauvaise qualité.
Ainsi, sur l'instance mono-critère, on remarque que MeTiS ne parvient pas à obtenir de partitionnement d'aussi bonne qualité que PIERE et Scotch. PIERE permet de trouver les meilleures solutions, cependant il obtient aussi des solutions de mauvaise qualité. Scotch a moins de dispersion, vraisemblablement dû au fait qu'il effectue par défaut deux calculs multi-niveaux et garde le meilleur résultat.
Sur l'instance multi-critères, nous avons fait figurer MeTiS mais avons remarqué que les résultats retournés ne respectaient pas forcément les contraintes d'équilibre que nous avions im-

1. disponible sur `http://www.i2m.univ-amu.fr/latp_numerique/?q=node/6` (« mesh1_4 »)



(a) Partitionnement mono-critère

(b) Partitionnement multi-critères

FIGURE 3 – Histogrammes présentant les distributions du volume de communications des partitionnement trouvés par MeTiS, PIERE et Scotch. On observe l'influence de la numérotation du graphe sur les algorithmes et on observe une grande variabilité dans la qualité des solutions. Dans l'absolu, PIERE parvient à trouver les meilleurs résultats pour nos deux cas tests. En moyenne, PIERE et Scotch sont assez semblables en mono-critère, tout comme PIERE et Metis en multi-critères.

posées (qui correspondent à un déséquilibre supérieur à 5%). Ce phénomène se produisant de manière très fréquente (298 fois pour les 500 expériences), nous montrons également les résultats d'une version plus contrainte, que nous nommons MeTiS_eq. Dans cette dernière version, nous relançons les calculs non acceptables en demandant un déséquilibre maximal de $2,5\%$ et nous obtenons en pratique un déséquilibre inférieur à 5%, donc acceptable. Contrairement à MeTiS, PIERE permet d'obtenir dans tous les cas une solution valide, preuve de l'intérêt d'équilibrer dès le partitionnement initial et d'avoir des algorithmes de raffinement ne détruisant pas cet équilibre. De plus, dans de nombreux cas, PIERE trouve de meilleures solutions que MeTiS, donc le fait d'imposer le respect de la contrainte d'équilibre tout au long du calcul ne constitue pas une gêne à l'obtention d'un résultat de bonne qualité.
Néanmoins, à de nombreuses reprises la solution obtenue est de mauvaise qualité. Ce phénomène peut s'expliquer par le fait que lors du raffinement nous départageons les mouvements de gains identiques en utilisant la numérotation du problème. Nous travaillons actuellement à des améliorations pour obtenir plus fréquemment les meilleures réponses.

5. Conclusion

Nous avons proposé des évolutions aux techniques multi-niveaux classiques en mono-critère pour nous permettre de résoudre le problème de partitionnement multi-critères de graphes.
Nous avons effectué des expérimentations en mono et multi-critères où nous avons montré que notre prototype PIERE pouvait trouver des partitions de meilleure qualité que Scotch et surtout MeTiS. De plus, notre approche se démarque nettement de l'existant en multi-critères en garantissant le respect des contraintes d'équilibre.
Nos futurs travaux consisteront à améliorer notre comportement vis-à-vis de la numérotation du problème et aussi à implanter nos algorithmes de manière pérenne et efficace dans Scotch, afin de proposer un outil robuste pour le partitionnement multi-critères de graphe.

Bibliographie

1. Barnard (S. T.) et Simon (H. D.). – A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency : Practice and Experience*, vol. 6, n2, 1994, pp. 101–117.
2. Chekuri (C.) et Khanna (S.). – On multi-dimensional packing problems. – In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, *SODA '99*, SODA '99, pp. 185–194, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.
3. Deveci (M.), Kaya (K.), Uçar (B.) et Ümit V. Çatalyürek. – Hypergraph partitioning for multiple communication cost metrics : Model and methods. *Journal of Parallel and Distributed Computing*, vol. 77, 2015, pp. 69 – 83.
4. Fiduccia (C. M.) et Mattheyses (R. M.). – A linear-time heuristic for improving network partitions. – In *Proceedings of the 19th Design Automation Conference*, pp. 175–181. IEEE, 1982.
5. Garey (M. R.) et Johnson (D. S.). – *Computers and Intractability : A Guide to the Theory of NP-Completeness*. – New York, NY, USA, W. H. Freeman & Co., 1979.
6. Hendrickson (B.). – Graph partitioning and parallel solvers : Has the emperor no clothes ? – In *In Proc. Irregular'98*, pp. 218–225. Springer-Verlag, 1998.
7. Hendrickson (B.) et Kolda (T. G.). – Graph partitioning models for parallel computing. *Parallel Comput.*, vol. 26, n12, novembre 2000, pp. 1519–1534.
8. Karypis (G.) et Kumar (V.). – A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, vol. 20, n1, décembre 1998, pp. 359–392.
9. Karypis (G.) et Kumar (V.). – Multilevel algorithms for multi-constraint graph partitioning. – In *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, *SC '98*, SC '98, pp. 1–13, Washington, DC, USA, 1998. IEEE Computer Society.
10. Leinberger (W.), Karypis (G.) et Kumar (V.). – Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. – In *Parallel Processing, 1999. Proceedings. 1999 International Conference on*, pp. 404–412, 1999.
11. Pellegrini (F.). – Scotch and libScotch 5.1 User's Guide, août 2008. user's manual.