Équilibrage de charge dynamique avec un nombre variable de processeurs basé sur des méthodes de partitionnement de graphe

Clément VUCHENER

Université de Bordeaux – INRIA HiePACS – LaBRI

7 février 2014

Plan

1. Introduction
2. État de l'art
   - Partitionnement
   - Repartitionnement
   - Problématique et positionnement
3. Repartitionnement $M \times N$
   - Étude préliminaire
   - Méthode de partitionnement biaisé
   - Méthode diffusive
4. Résultats expérimentaux
   - Influence du nombre de nouvelles parties
   - Comparaison sur des graphes complexes
5. Conclusion et Perspectives

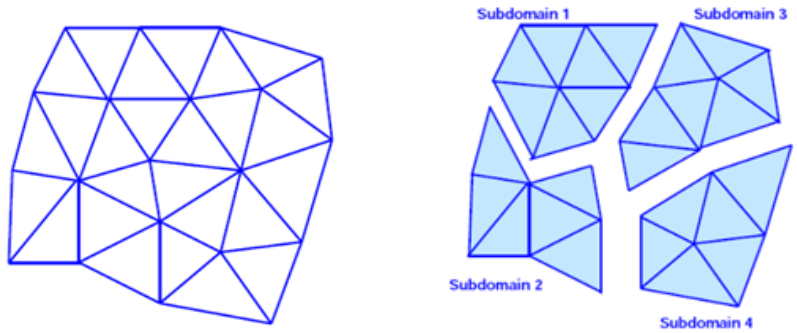
1. Introduction
2. État de l'art
   - Partitionnement
   - Repartitionnement
   - Problématique et positionnement
3. Repartitionnement $M \times N$
   - Étude préliminaire
   - Méthode de partitionnement biaisé
   - Méthode diffusive
4. Résultats expérimentaux
   - Influence du nombre de nouvelles parties
   - Comparaison sur des graphes complexes
5. Conclusion et Perspectives

Contexte

Simulation numérique

- Discrétisation en espace : maillages
- Discrétisation en temps : itérations

La parallélisation de ces applications implique une distribution des calculs.



Maillage Sous-domaines

source : [TDF06]

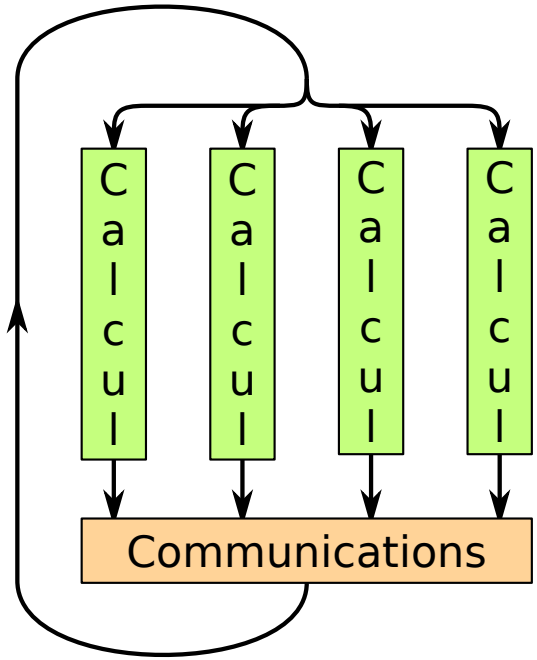
La qualité de la distribution conditionne les performances de l'application.

Équilibrage

$$T_{total} = \quad T_{calcul} + T_{comm}$$

On veut minimiser :

- Le temps de calcul T_{calcul} (équilibrage)
- Le temps de communication T_{comm}



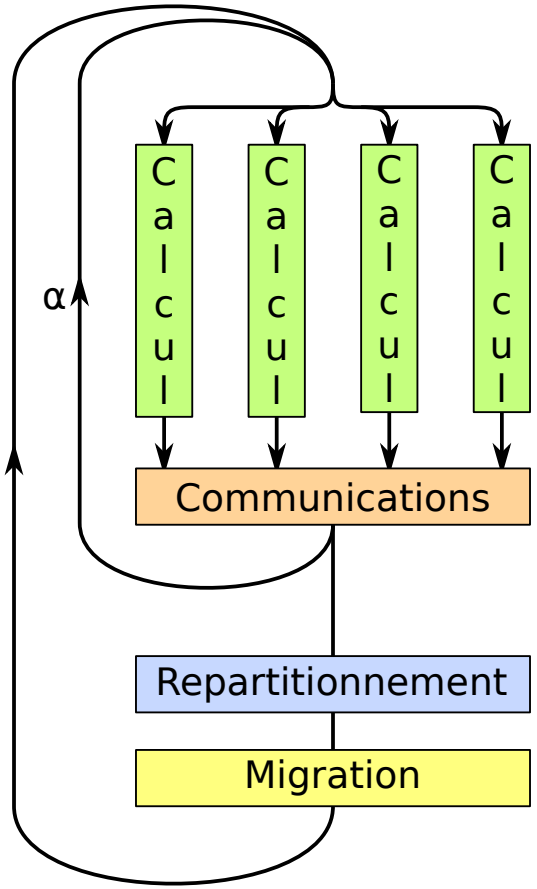
Rééquilibrage

La charge évolue et se déséquilibre au cours de la simulation.

$$T_{total} = \alpha \times (T_{calcul} + T_{comm}) + T_{repart} + T_{mig}$$

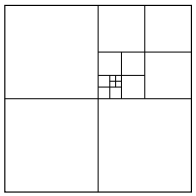
On veut minimiser :

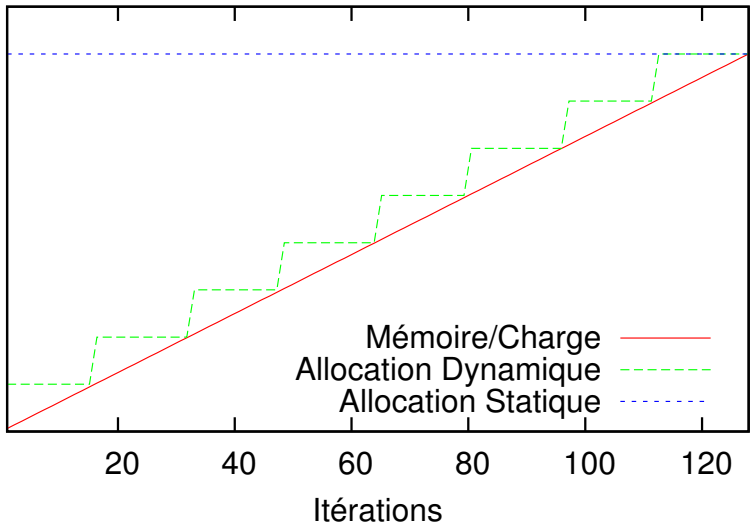
- Le temps de calcul T_{calcul} (équilibrage)
- Le temps de communication T_{comm}
- Le temps de calcul de la nouvelle partition T_{repart}
- Le temps de migration des données T_{mig}



Problématique

Certaines applications peuvent avoir une charge qui augmente de façon très importante (par exemple : AMR).





- **Allocation statique des processeurs**
  Efficacité faible au début
- **Allocation dynamique des processeurs**
  Efficacité stable

$\rightarrow$ Besoin de calculer une bonne redistribution en tenant compte de la variation du nombre de processeurs.

1 Introduction

2 État de l'art
- Partitionnement
- Repartitionnement
- Problématique et positionnement

3 Repartitionnement $M \times N$
- Étude préliminaire
- Méthode de partitionnement biaisé
- Méthode diffusive

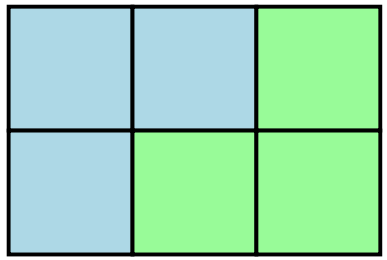
4 Résultats expérimentaux
- Influence du nombre de nouvelles parties
- Comparaison sur des graphes complexes

5 Conclusion et Perspectives

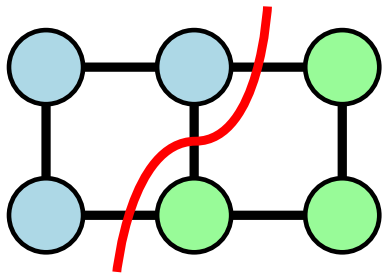
Partitionnement de graphe

Le maillage est modélisé par un graphe :

- Un sommet représente un élément du maillage
- Une arête représente une dépendance de calcul



Maillage



Graphe

Une partition doit donc :

- Être équilibrée à une tolérance près

  Le déséquilibre est le rapport entre la plus grande partie et la taille moyenne.

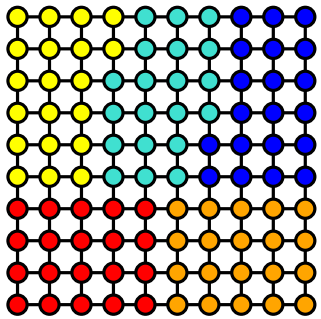
- Minimiser la coupe

  La coupe donne une approximation du volume de communication.

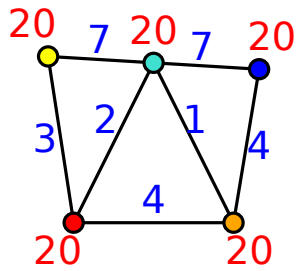
Graphe quotient

Définition

Le graphe quotient par rapport à une partition fusionne les sommets appartenant à une même partie. Le poids des sommets correspond à la taille des parties et celui des arêtes à la coupe entre les deux parties.



Partition d'un graphe en 5

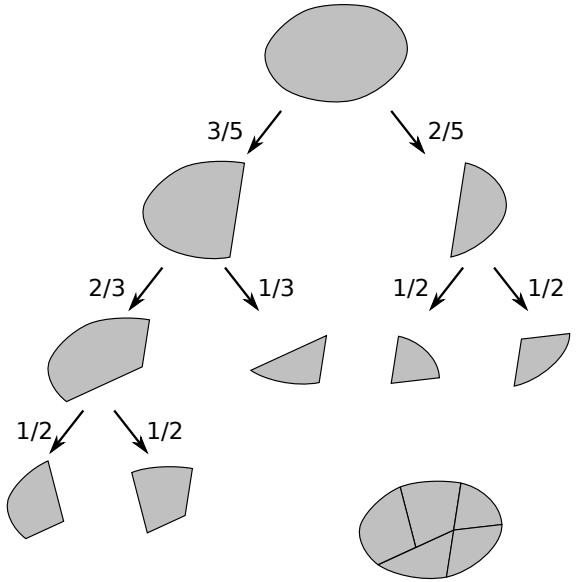


Graphe quotient associé à la partition

Le graphe quotient donne une vue globale de la partition.

Méthodes de partitionnement k-aire

- Méthode des bissections récursives
  - Coupe du graphe en 2 puis de chaque partie récursivement jusqu'à obtenir k parties
  - Basée sur des heuristiques de bipartitionnement nombreuses et maîtrisées *greedy growing*, *bubble growing*, spectral, ...
  - Pas de vision globale du problème
- Méthode de partitionnement k-aire direct
  - Théoriquement donne de meilleurs résultats
  - En pratique, les heuristiques sont de moins bonne qualité.



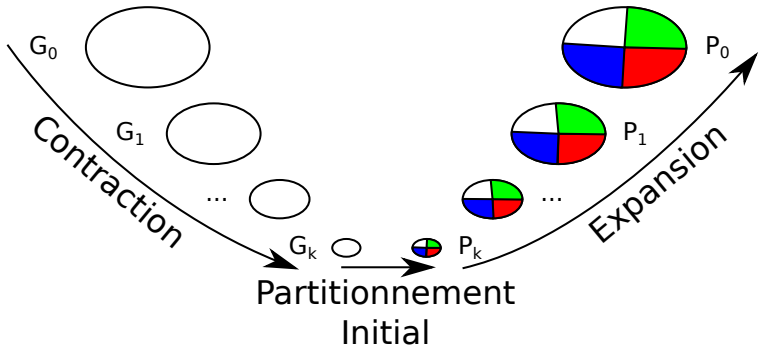
Arbre de bissections pour 5 parties

Partitionnement multi-niveaux

La partitionnement multi-niveaux permet de simplifier le problème de partitionnement

3 phases d'un cycle en V :

1. Phase de contraction
   Fusion de sommets : HEM, ...
2. Partitionnement initial
   Partitionnement en k parties
3. Phase d'expansion
   Projection et raffinement de la partition



Raffinement de partition

Algorithme de raffinement FM [FM82] :

- Déplace les sommets un par un suivant leur gain de coupe
- Tous les sommets sont déplacés exactement une fois par passe
- Garde la meilleure partition

Complexité : $O(|E|)$

-1	$+1$	-1	-2
-3	-2	$+2$	-1
-2	-1	$+1$	-2

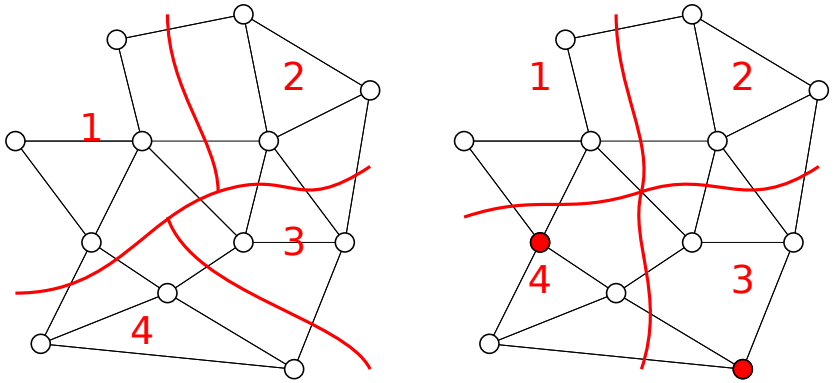
Coupe : 6

Exemple de raffinement FM d'une bipartition

Nombreuses autres méthodes [KL70, Pel07, MMS09]

Repartitionnement

Calcul d'une nouvelle partition à partir d'une partition déséquilibrée.



Ancienne partition Nouvelle partition

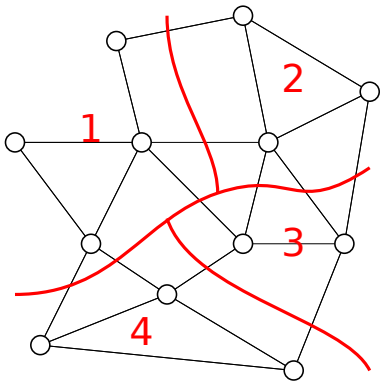
Objectifs :

- Équilibrer les tailles des parties
- Minimiser la coupe
- Optimiser la migration

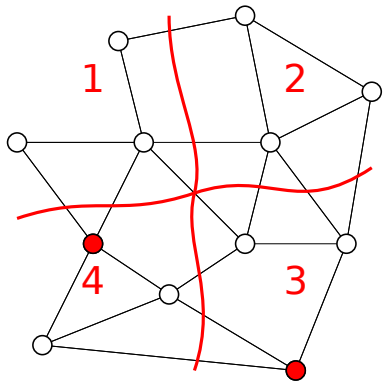
Métriques pour la migration

La migration peut être évaluée à l'aide de différentes métriques :

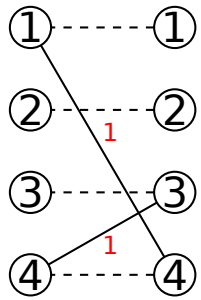
- TotalV le volume total de migration
- MaxV le volume maximum de migration par partie
- TotalZ le nombre total de messages de migration
- MaxZ le nombre maximum de messages par partie



Ancienne partition en 4



Nouvelle partition en 4



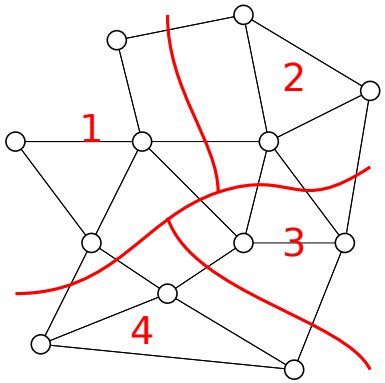
Communications

$\text{TotalV} = 2$
$\text{MaxV} = 2$
$\text{TotalZ} = 2$
$\text{MaxZ} = 2$

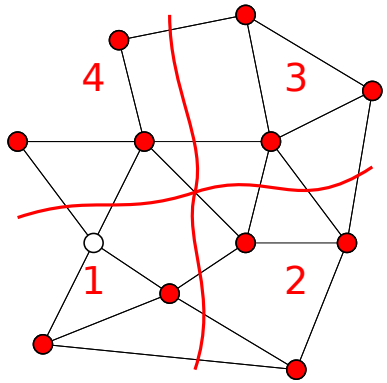
Scratch-Remap

La méthode de *Scratch-Remap* se décompose en deux étapes simples :

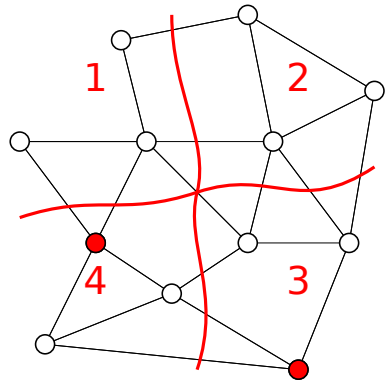
1. Calcule d'une nouvelle partition *from scratch*
2. Renumérotation des parties pour minimiser la migration



Partition initiale déséquilibrée



Nouvelle partition mal numérotée



Nouvelle partition renumérotée

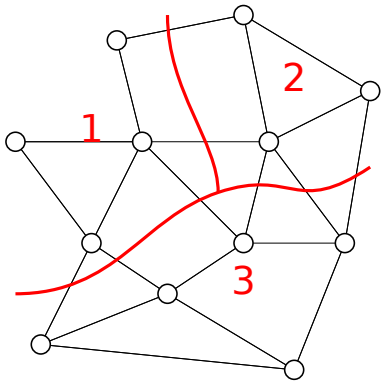
→ Avantage : Très bonne coupe

→ Inconvénient : Migration médiocre

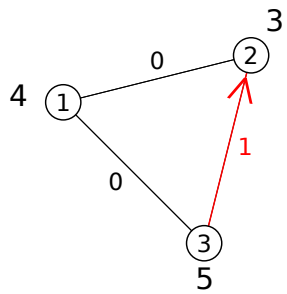
Diffusion

Le repartitionnement basé sur la diffusion comporte deux étapes :

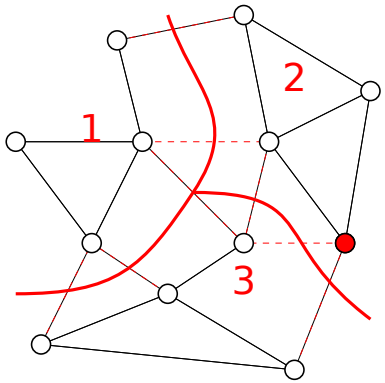
1. Calcule la migration optimale [OR94, HB95]. Les messages de migration sont autorisés seulement entre parties voisines.
2. Des sommets sont sélectionnés le long des frontières à l'aide d'une heuristique similaire à FM.



Partition initiale déséquilibrée



Calcul de la migration optimale



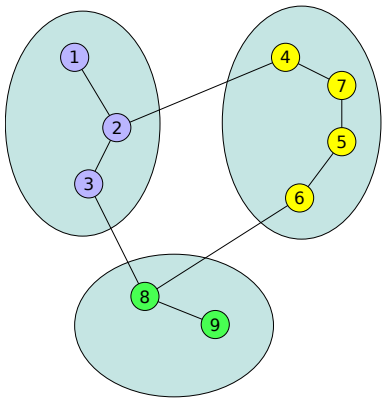
Échange des sommets

→ Avantage : Bonne migration

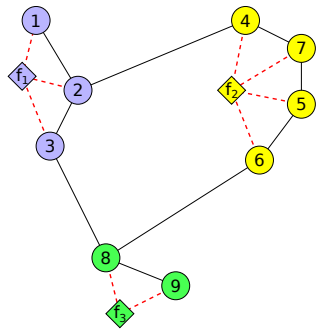
→ Inconvénient : Coupe détériorée en cas de migration importante

Partitionnement biaisé

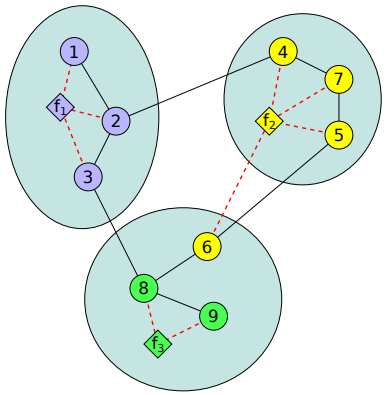
- Le graphe est modifié pour modéliser le problème de repartitionnement avec des sommets fixes et des arêtes de migration [cBD+07, ACFK07] (*Zoltan*).
- Les heuristiques sont modifiées directement [HLD97] (*Scotch*).



Partition initiale déséquilibrée



Graphe enrichi



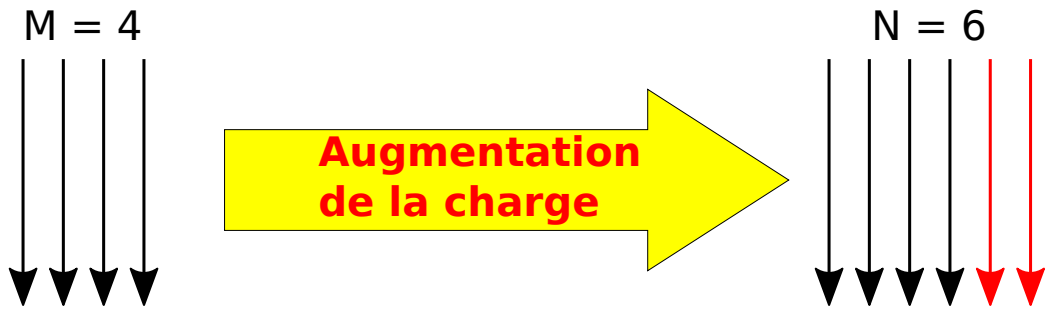
Graphe enrichi partitionné

→ Avantage : Permet un compromis (ajustable) entre coupe et migration.

Problématique et positionnement

Repartitionnement avec nombre variable de processeurs

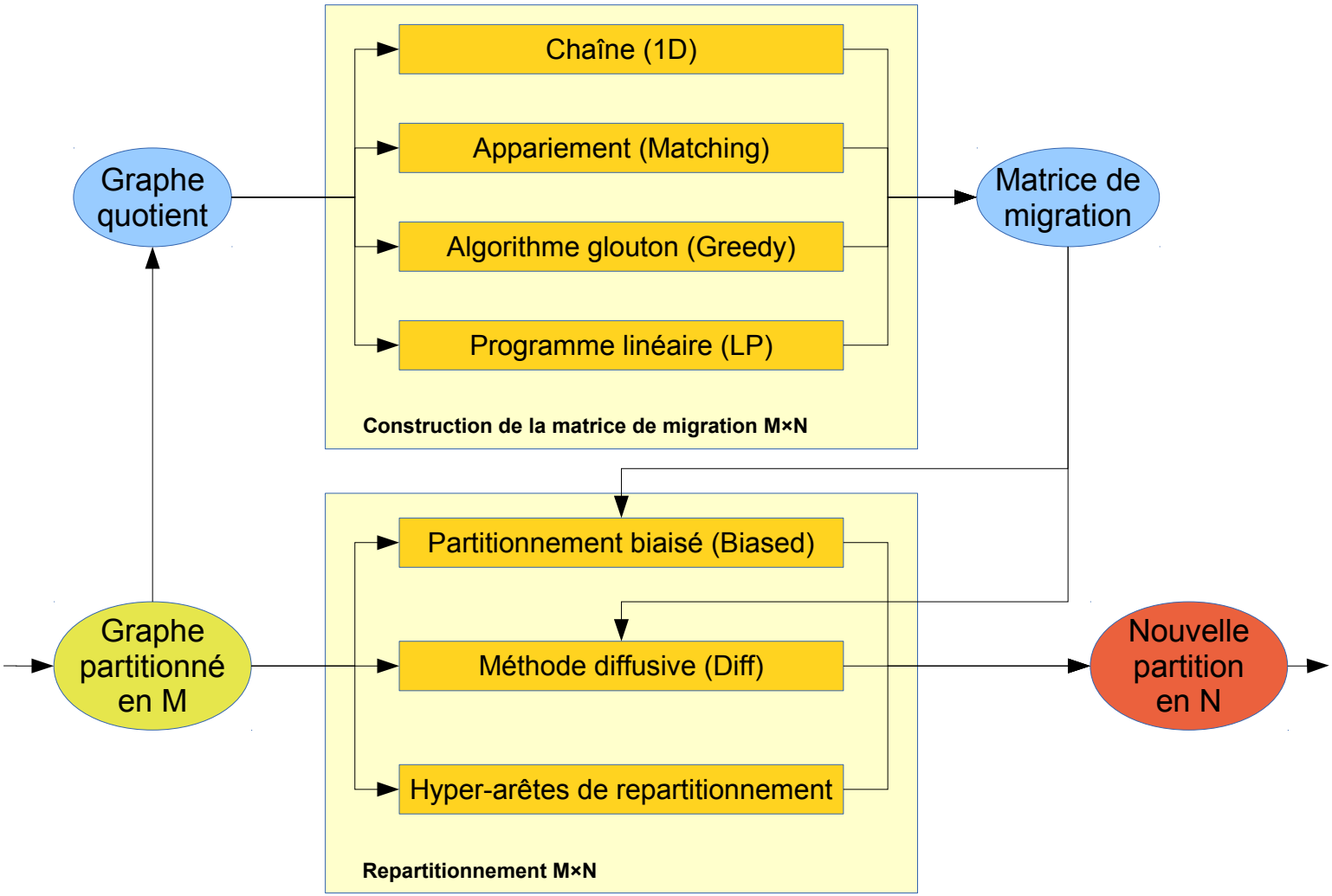
Ajuster le nombre de ressources en fonction de la consommation mémoire permet d'optimiser l'efficacité et la consommation pour des codes type AMR [IC05].



Le nombre de ressource augmente proportionnellement à la charge (+150 %).

Les méthodes actuelles sont limitées au cas où le nombre de processeurs est fixe.

Nos méthodes de repartitionnement $M \times N$

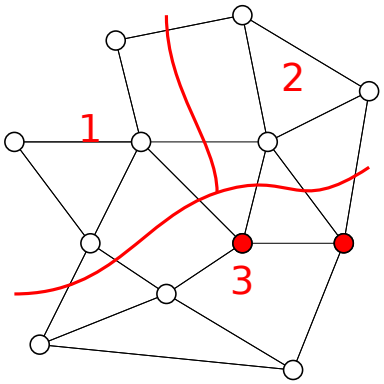


1. Introduction
2. État de l'art
   - Partitionnement
   - Repartitionnement
   - Problématique et positionnement
3. Repartitionnement $M \times N$
   - Étude préliminaire
   - Méthode de partitionnement biaisé
   - Méthode diffusive
4. Résultats expérimentaux
   - Influence du nombre de nouvelles parties
   - Comparaison sur des graphes complexes
5. Conclusion et Perspectives

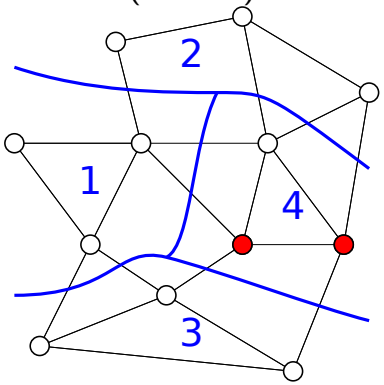
Matrice de migration

La matrice de migration décrit les échanges de données entre parties.

L'élément $m_{i,j}$ donne la quantité de données envoyées de l'ancienne partie i vers la nouvelle partie j.



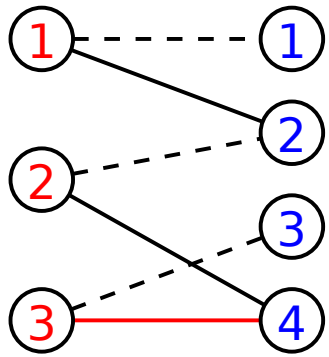
Ancienne partition ($M = 3$)



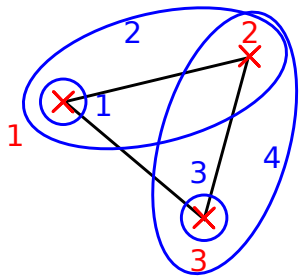
Nouvelle partition ($N = 4$)

$$\begin{bmatrix} 3 & 1 & 0 & 0 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & 3 & 2 \end{bmatrix}$$

Matrice de migration



Graphe biparti de repartitionnement



Hypergraphe de repartitionnement (superposé au graphe quotient)

Matrices de migration optimales (cas équilibré)

Définition

Une matrice de migration optimale donne à la fois un TotalV minimal et TotalZ minimal.

On étudie les matrices de migration optimales dans le cas où les anciennes et nouvelles partitions sont parfaitement équilibrées.

Volume minimal de migration

$$\text{TotalV}_{opt} = W\left(1 - \frac{\min(M,N)}{\max(M,N)}\right)$$

On maximise la quantité de données restant sur place.
Cas $N > M$: M fois la taille d'une nouvelle partie ($\frac{W}{N}$).

$$\begin{bmatrix} \frac{W}{N} & 0 & 0 & m_{1,4} \\ 0 & \frac{W}{N} & 0 & m_{2,4} \\ 0 & 0 & \frac{W}{N} & m_{3,4} \end{bmatrix}$$

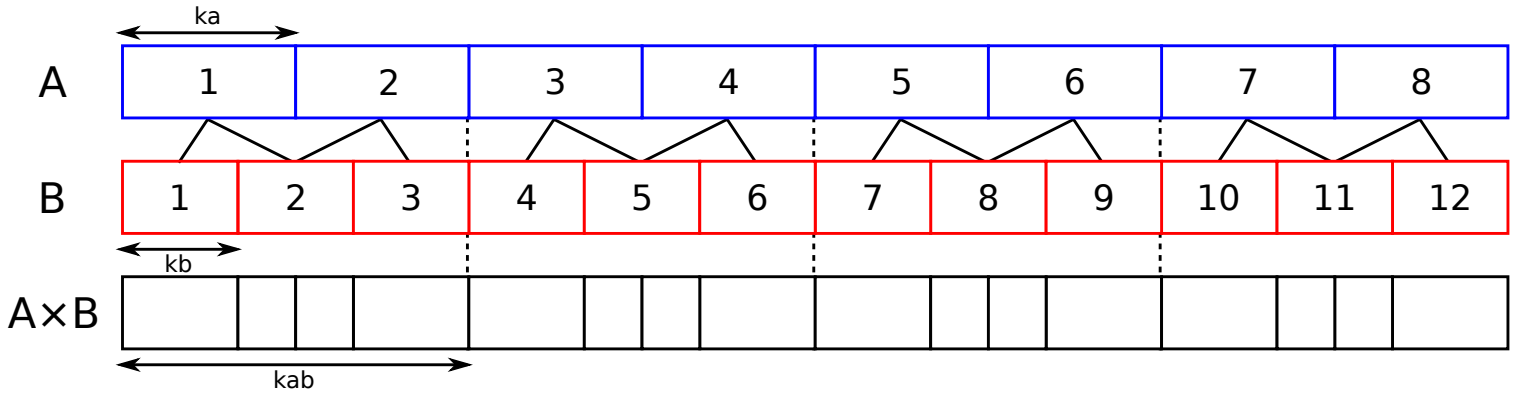
Nombre minimal de messages (cas équilibré)

Nombre minimal de messages

$$\text{TOTALZ}_{opt} = \max(M, N) - \text{pgcd}(M, N)$$

Exemple du partitionnement de la chaine :

- $\text{pgcd}(M, N) - 1$ frontières communes
- $M + N - \text{pgcd}(M, N)$ intersections



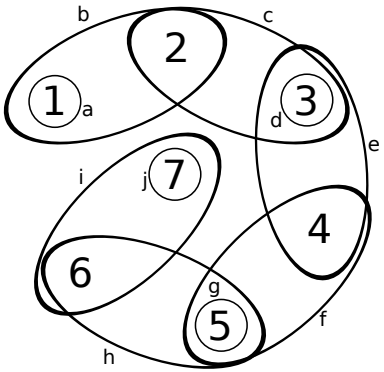
Exemples de matrices de migration (cas équilibré)

Repartitionnement 7×10

Matrice en escalier permutée

$\textsc{TotalV} = 24$

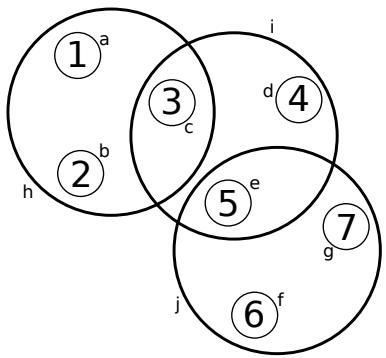
$$\begin{bmatrix} 7 & & & & & & & 3 & & \\ & 6 & & & & & & 4 & & \\ & 1 & 7 & 2 & & & & & & \\ & & & 5 & & & & & 5 & \\ & & & & 7 & 1 & & & 2 & \\ & & & & & 6 & & & & 4 \\ & & & & & & 7 & & & 3 \end{bmatrix}$$



Matrice optimale basé sur une sous matrice en escalier

$\textsc{TotalV} = 21$

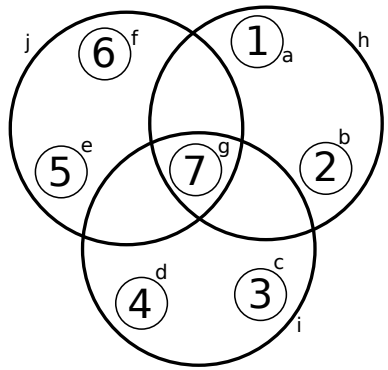
$$\begin{bmatrix} 7 & & & & & & & 3 & & \\ & 7 & & & & & & 3 & & \\ & & 7 & & & & & 1 & 2 & \\ & & & 7 & & & & & 3 & \\ & & & & 7 & & & & 2 & 1 \\ & & & & & 7 & & & & 3 \\ & & & & & & 7 & & & 3 \end{bmatrix}$$



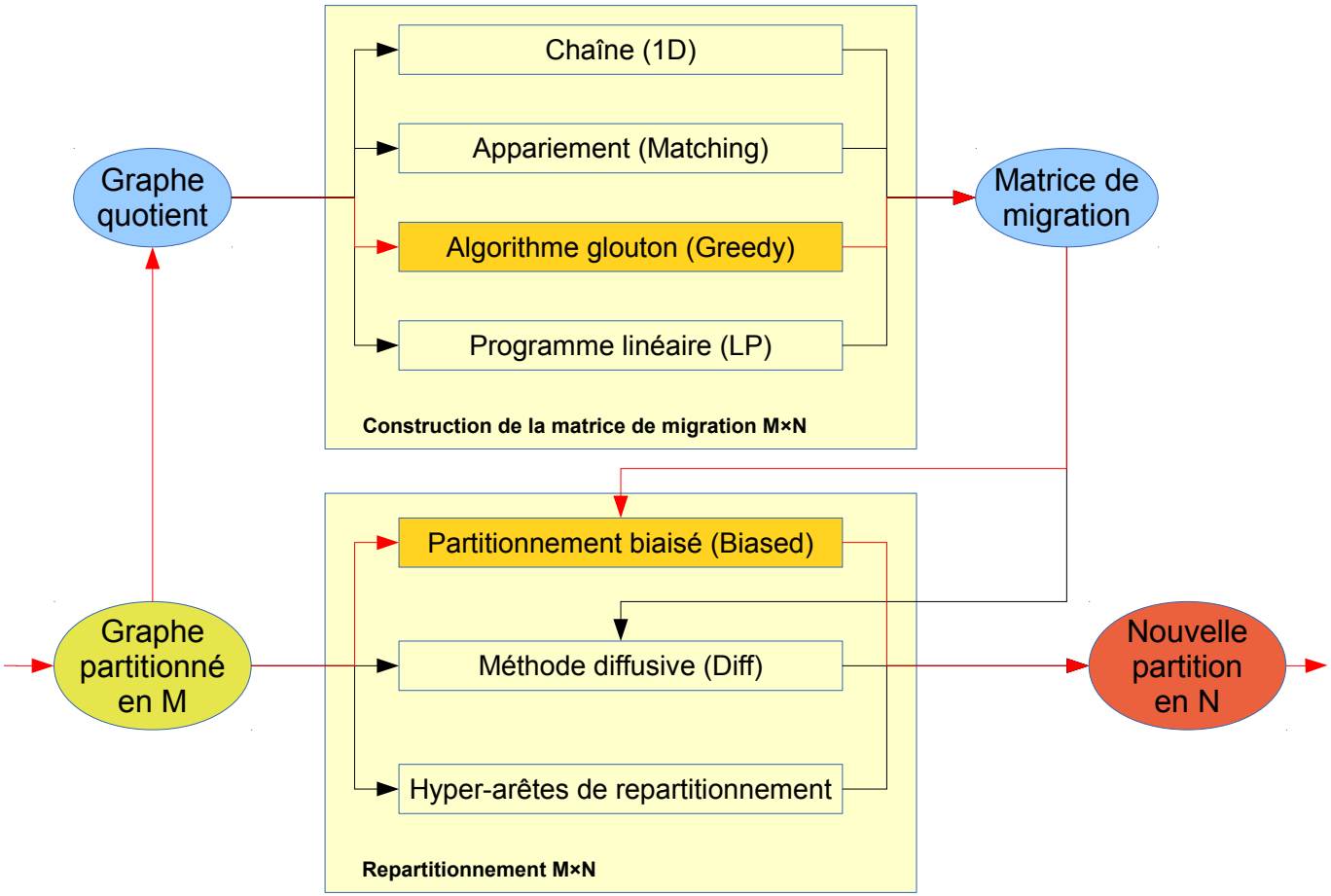
Autre exemple de matrice optimale

$\textsc{TotalV} = 21$

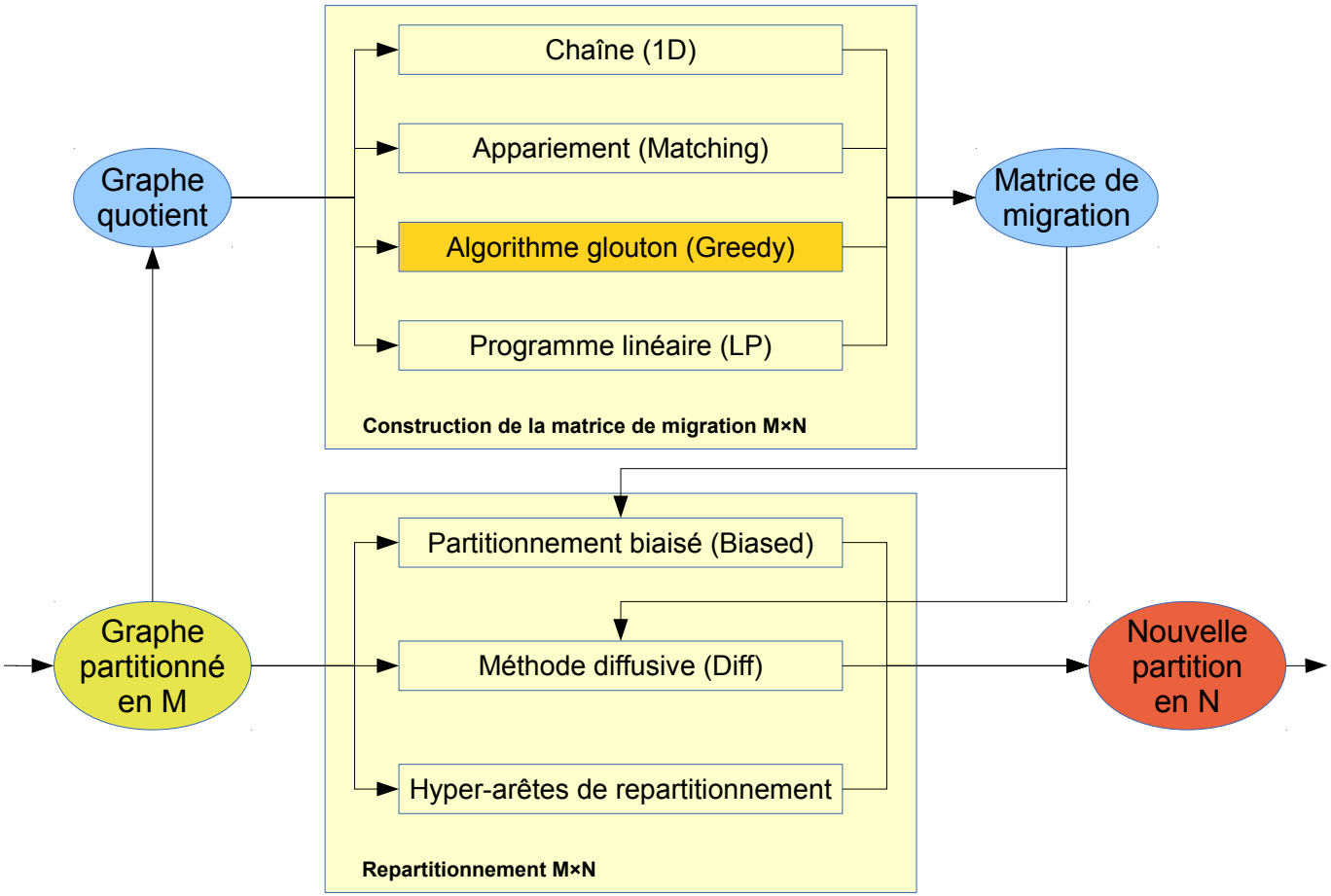
$$\begin{bmatrix} 7 & & & & & & & 3 & & \\ & 7 & & & & & & & 3 & \\ & & 7 & & & & & & & 3 \\ & & & 7 & & & & 3 & & \\ & & & & 7 & & & & 3 & \\ & & & & & 7 & & & & 3 \\ & & & & & & 7 & 1 & 1 & 1 \end{bmatrix}$$



Repartitionnement $M \times N$: partitionnement biaisé



Algorithme glouton



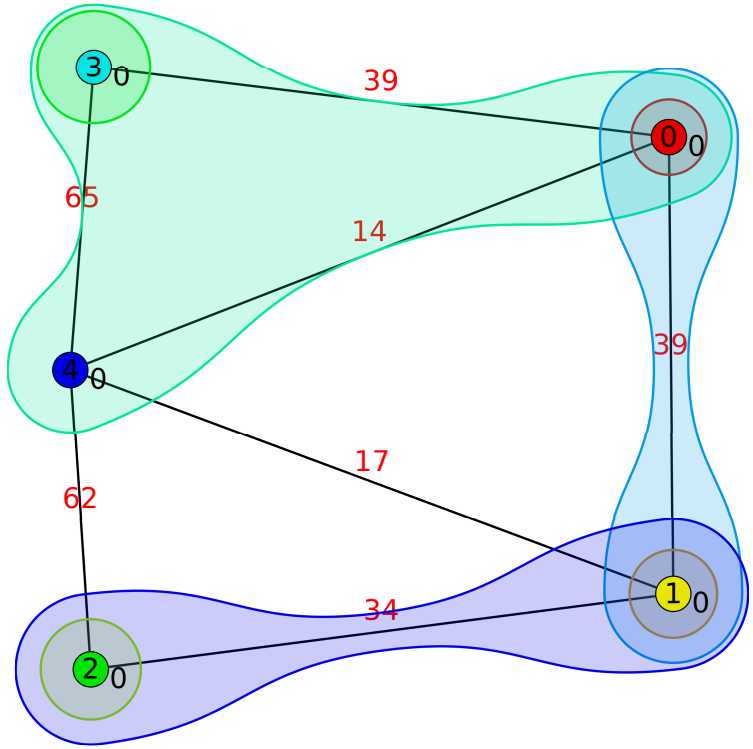
Algorithme glouton

1. Remplir la diagonale (si possible)
2. Construire les nouvelles parties restantes à partir d'anciennes parties proches

Repartitionnement 5×7 (déséquilibre initial 26 %)

$$\frac{W}{N} = \frac{210}{7} = 30$$

30	0	0	0	3	20	0	53
0	30	0	0	0	10	10	50
0	0	30	0	0	0	20	50
0	0	0	30	5	0	0	35
0	0	0	0	22	0	0	22
30	30	30	30	30	30	30	



$\textsc{TotalV} = 68$ et $\textsc{TotalZ} = 6$.

Réduction du nombre de messages

Remarque

Pour un repartitionnement $M \times N$, l'algorithme ajoute $M + N - 1$ éléments non-nuls dans la matrice.

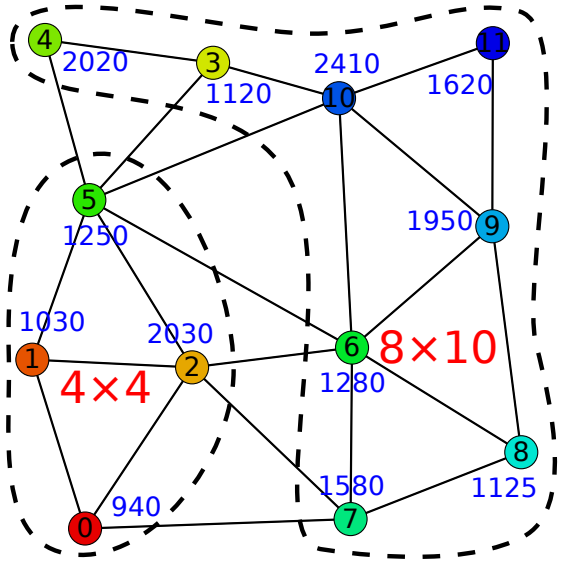
Réduction de TotalZ

Découper ce problème en K sous-problèmes de repartitionnement $M_i \times N_i$ s'appliquant sur un sous-graphe regroupant M_i anciennes parties permet de réduire le nombre d'éléments non-nuls à $M + N - K$.

Contraintes :

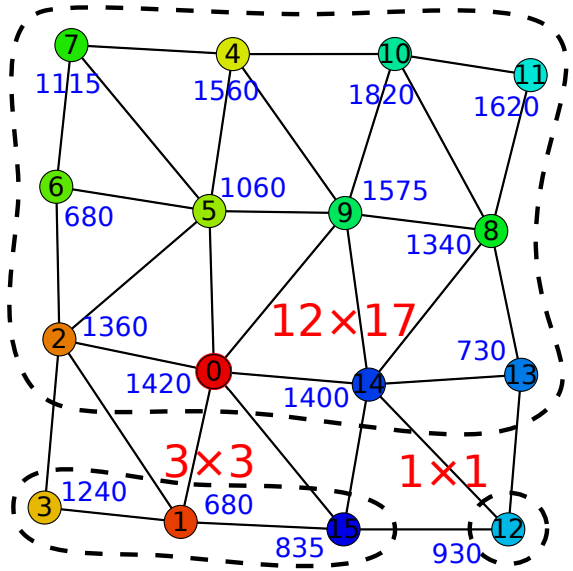
- Le poids du sous-graphe doit permettre de créer N_i nouvelles parties équilibrées.
- Il faut regrouper M_i anciennes parties proches.

Exemples de résultats



Repartitionnement 12×14

Nouvelle partie de taille 1311

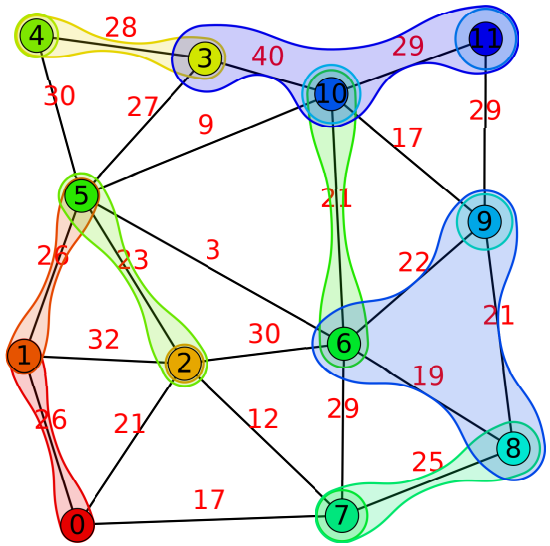


Repartitionnement 16×21

Nouvelle partie de taille 922

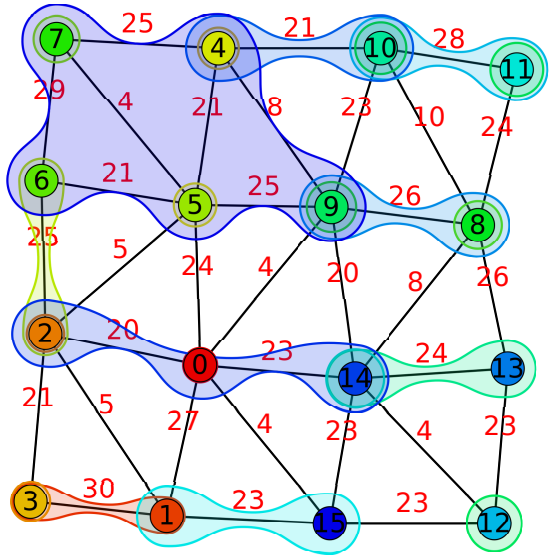
- Avantages : TotalV et TotalZ très bas
- Inconvénients : Certaines nouvelles parties sont réparties sur de trop nombreuses anciennes parties.

Exemples de résultats



Repartitionnement 12×14

Nouvelle partie de taille 1311

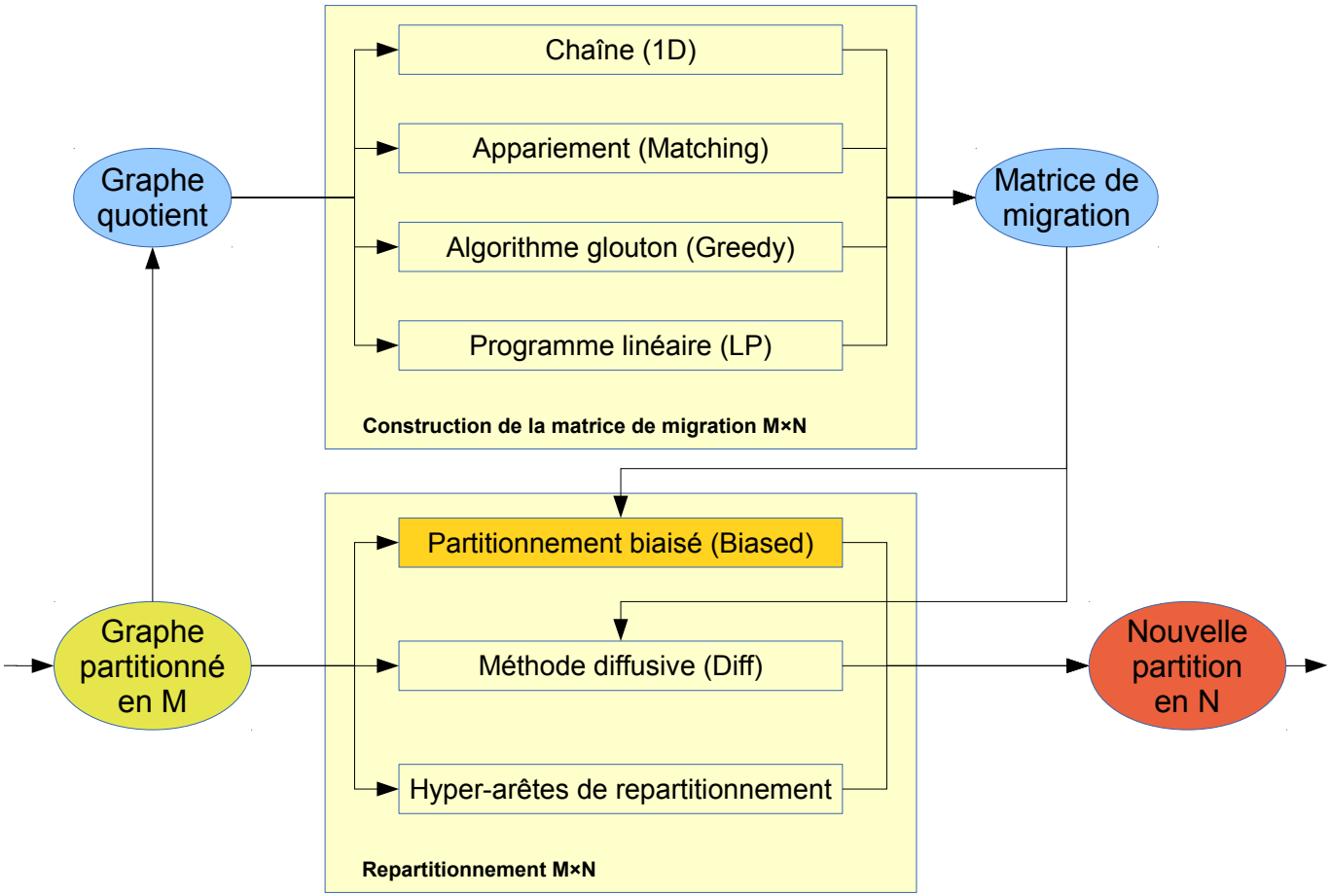


Repartitionnement 16×21

Nouvelle partie de taille 922

- Avantages : TotalV et TotalZ très bas
- Inconvénients : Certaines nouvelles parties sont réparties sur de trop nombreuses anciennes parties.

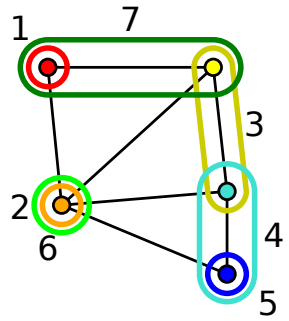
Repartitionnement biaisé

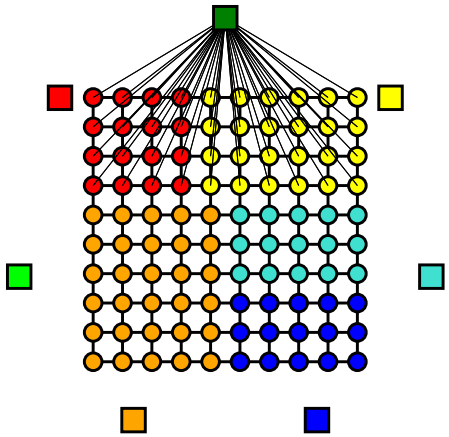


Algorithme de repartitionnement biaisé avec sommets fixes

1. Un graphe avec une ancienne partition en M parties est donnée.
2. Une matrice de migration $M \times N$ est calculée.
3. N sommets fixes de poids nuls sont ajoutés.
4. Les sommets fixes sont connectés aux sommets du graphe d'après l'hypergraphe de repartitionnement.
5. Le graphe enrichi est partitionné en N parties (KGGGP).
6. La partition est restreinte au graphe original.

Exemple de repartitionnement 5×7

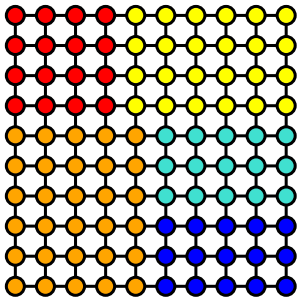


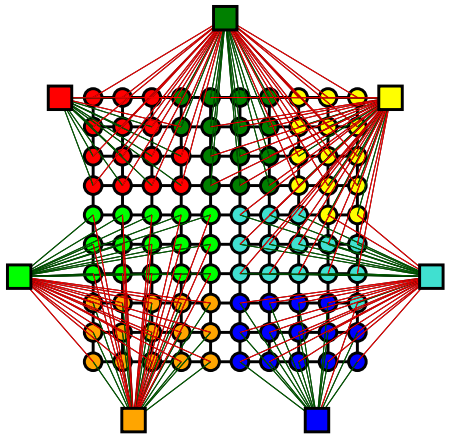


Algorithme de repartitionnement biaisé avec sommets fixes

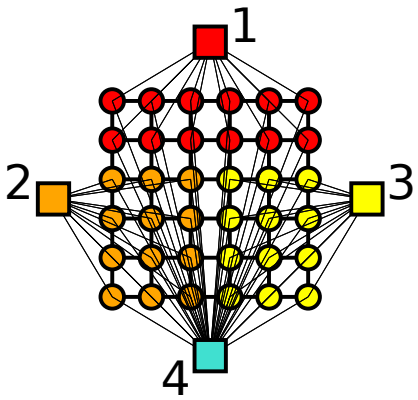
1. Un graphe avec une ancienne partition en M parties est donnée.
2. Une matrice de migration $M \times N$ est calculée.
3. N sommets fixes de poids nuls sont ajoutés.
4. Les sommets fixes sont connectés aux sommets du graphe d'après l'hypergraphe de repartitionnement.
5. Le graphe enrichi est partitionné en N parties (KGGGP).
6. La partition est restreinte au graphe original.

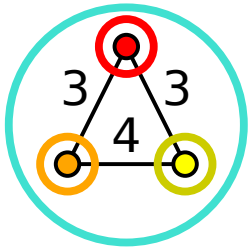
Exemple de repartitionnement 5×7

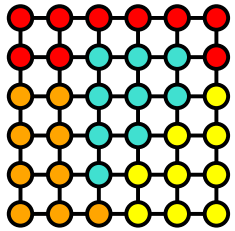




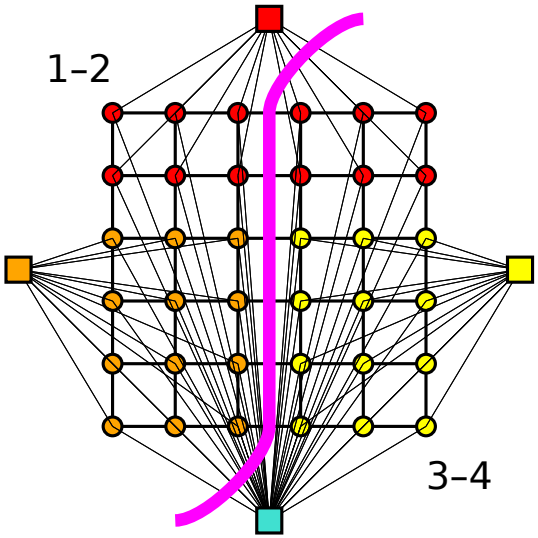
Limitations des bissections récursives

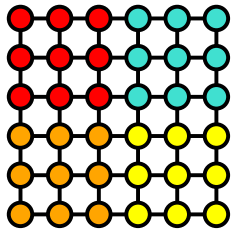






Partitionnement k-aire :
36 arêtes de migration coupées





Bissections récursives :
39 arêtes de migration coupées

k-way Greedy Graph Growing Partitioning (KGGGP)

Notre partitionnement biaisé nécessite un partitionnement k-aire direct.
Algorithme :

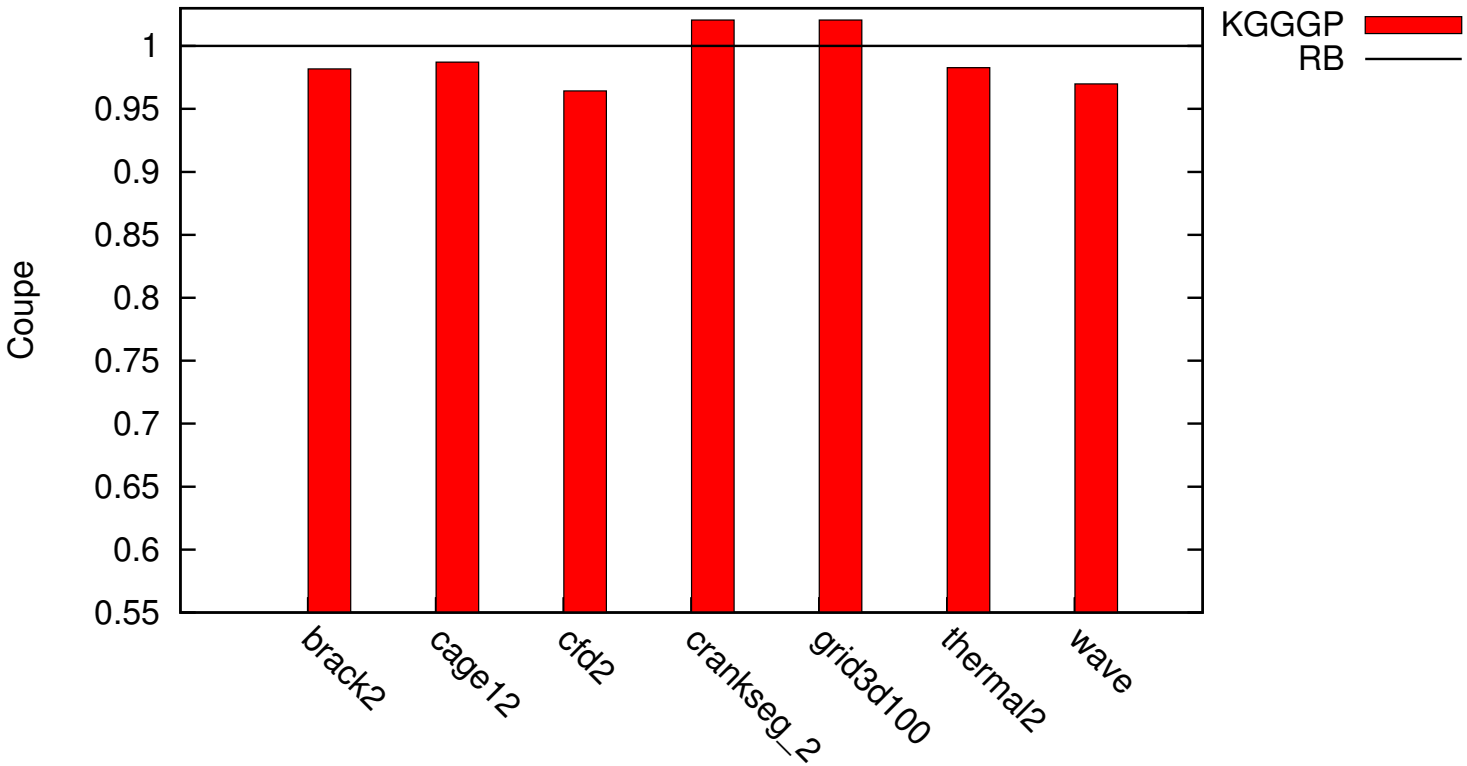
- Les sommets ne sont initialement dans aucune partie.
- Une heuristique inspirée de FM déplace les sommets dans les parties suivant le gain de coupe.

  Il faut calculer les gains pour chacune des k parties.
- Un sommet ne peut pas être déplacé dans une partie qui deviendrait trop grande.

Complexité : $O(k|E|)$

Évaluation du KGGGP

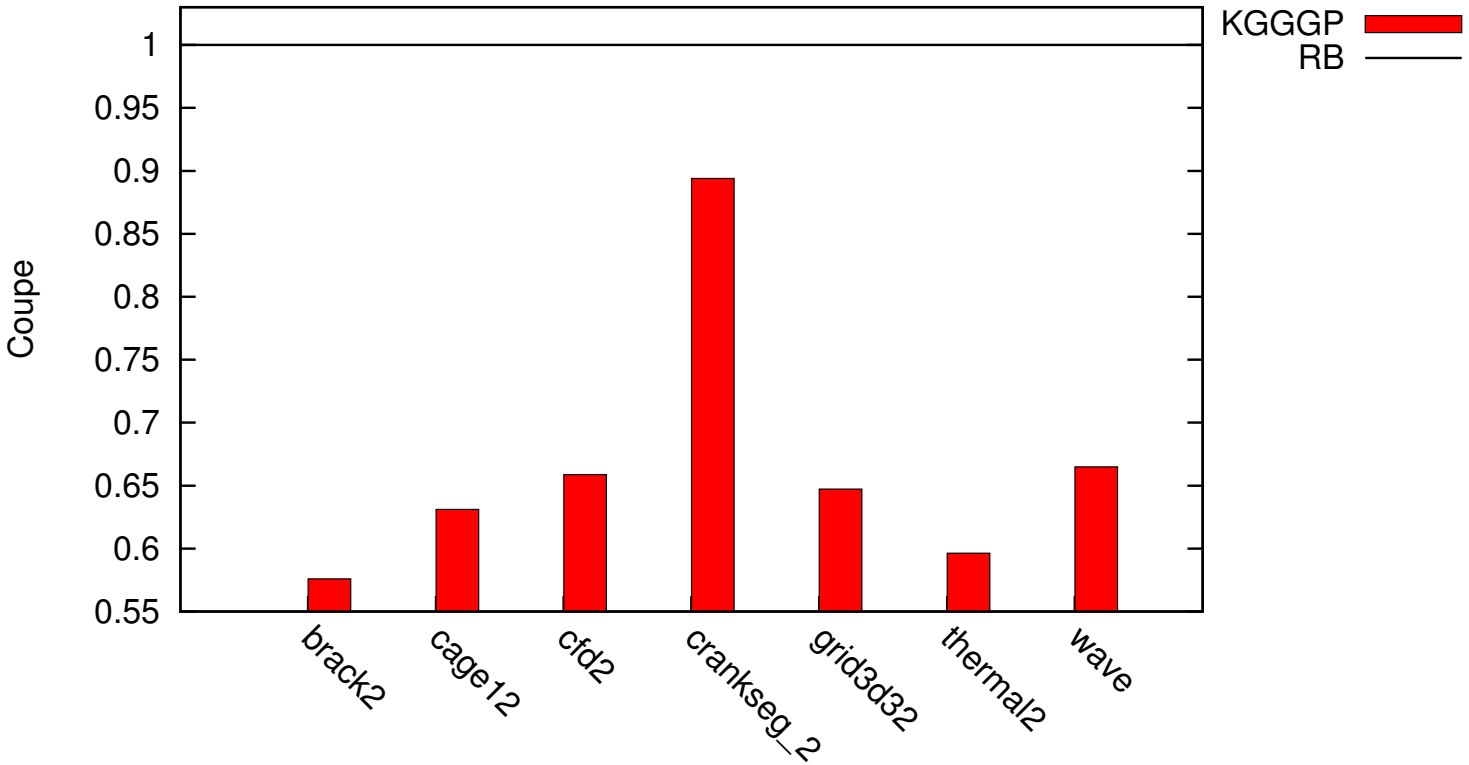
Comparaison relativement aux bissections récursives.



Partition en 32 parties multi-niveaux g-ref-4p

Évaluation du KGGGP

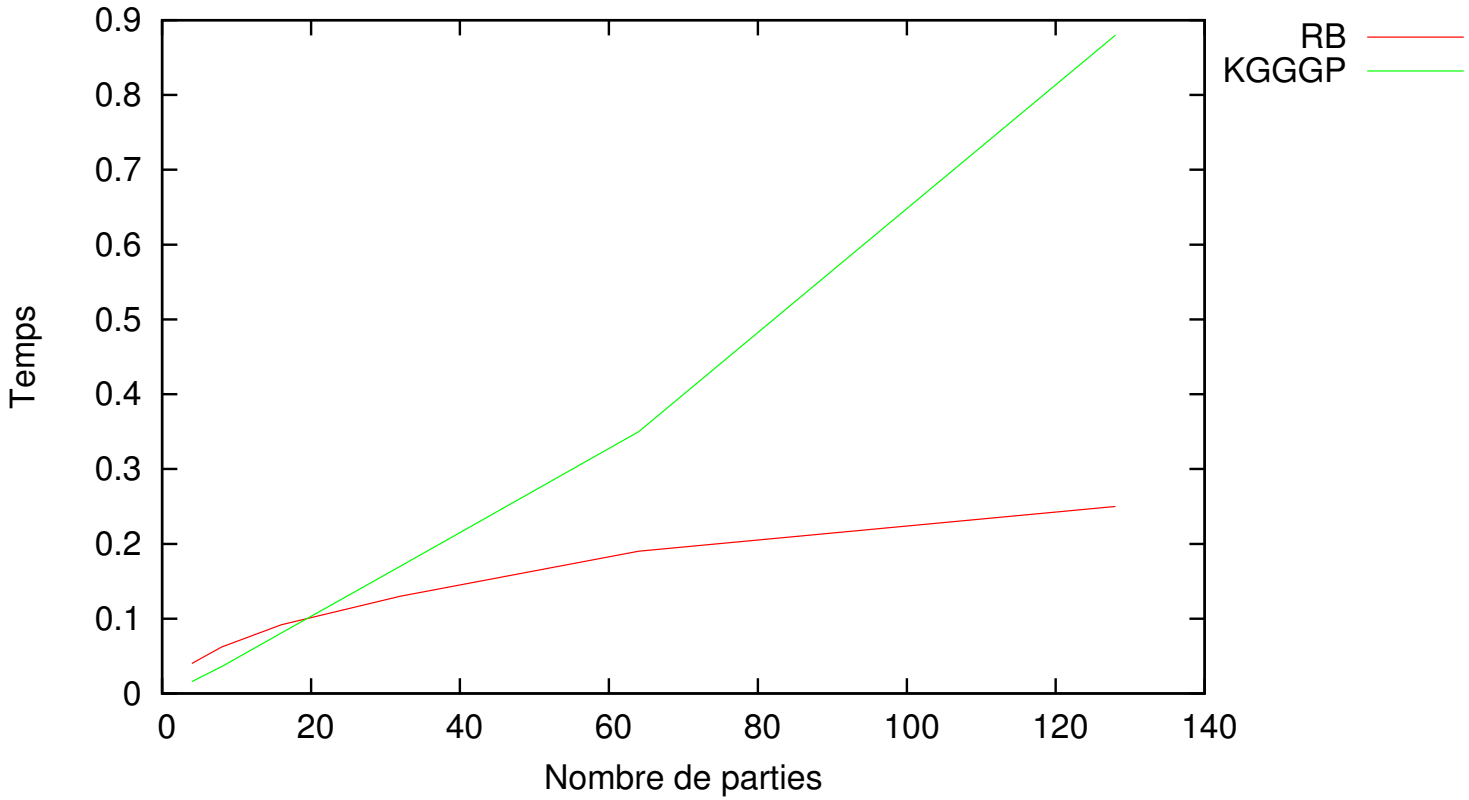
Comparaison relativement aux bissections récursives sur des graphes enrichis avec des sommets fixes (méthode *BIASED*).



Partition en 21 parties
Graphes enrichis pour un repartitionnement 16×21 multi-niveaux g-ref-4p

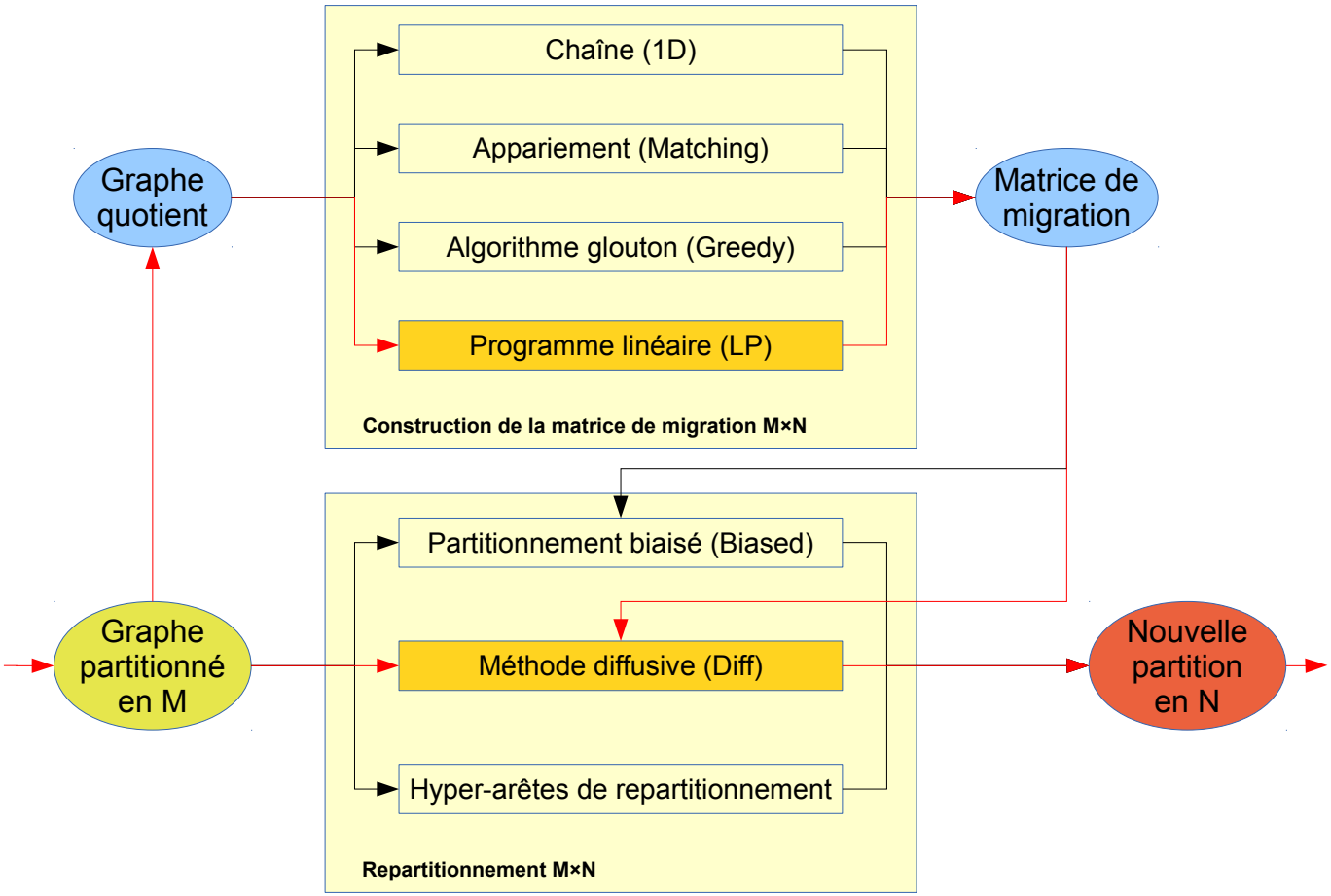
Évaluation du KGGGP

Évaluation de la complexité par rapport au nombre de parties

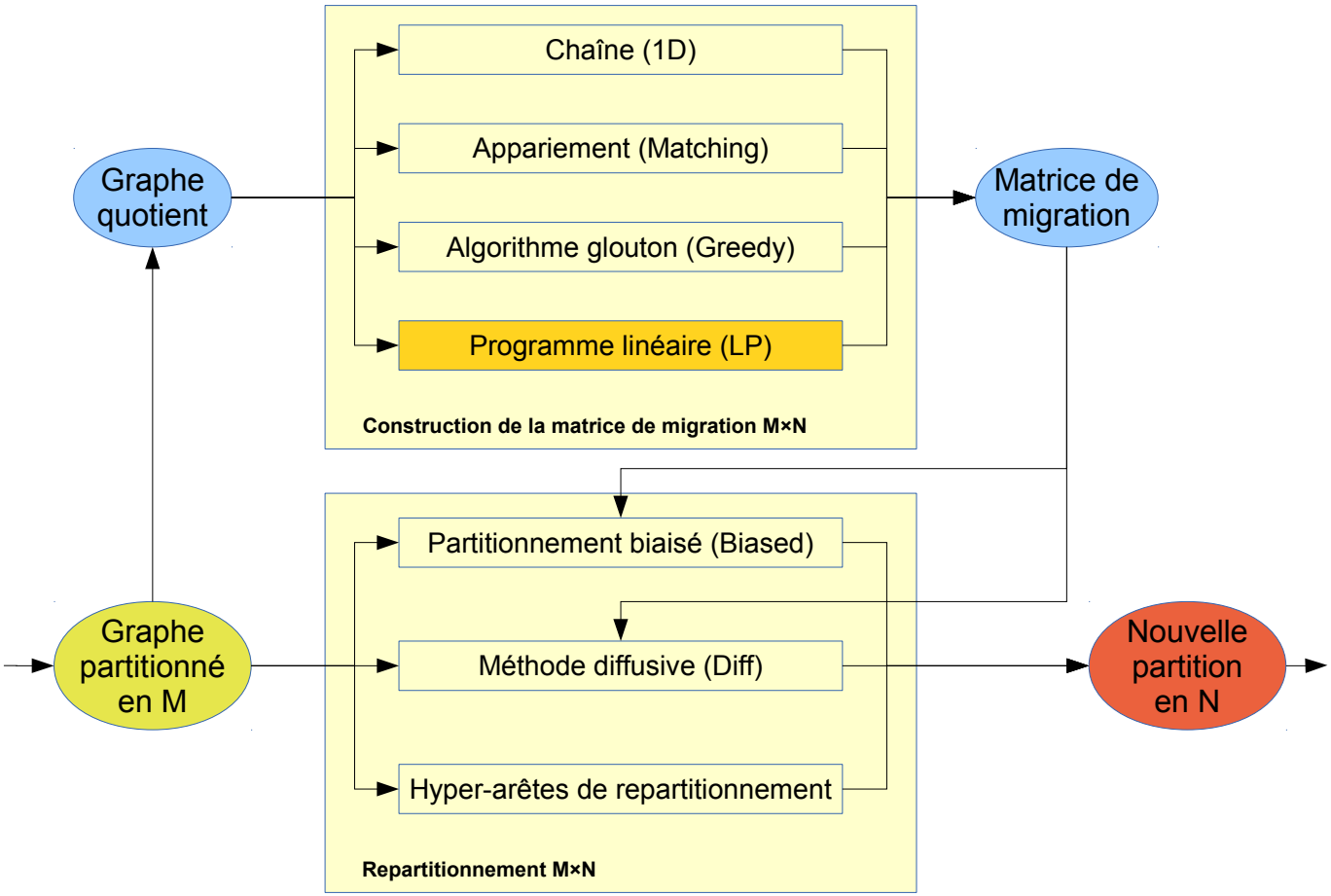


Partitionnement d'une grille $32 \times 32 \times 32$ sans multi-niveaux g-noref-1p

Repartitionnement $M \times N$: méthode diffusive

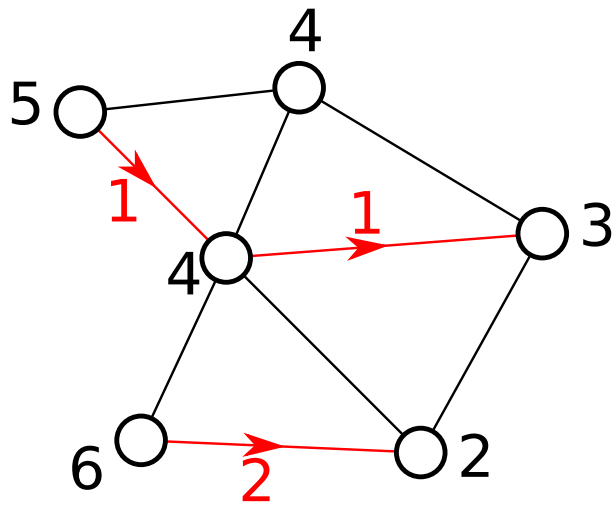


Programme linéaire



Rappels sur les méthodes diffusives

- Les méthodes diffusives calculent les messages permettant d'optimiser la migration.
- Les messages possibles sont donnés par le graphe quotient (les sommets seront migrés le long des frontières).
- Les messages peuvent être calculés en résolvant un système linéaire [HB95] ou un programme linéaire [OR94].

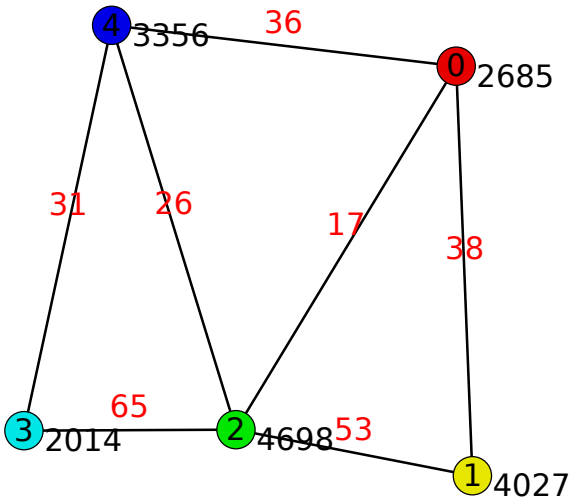


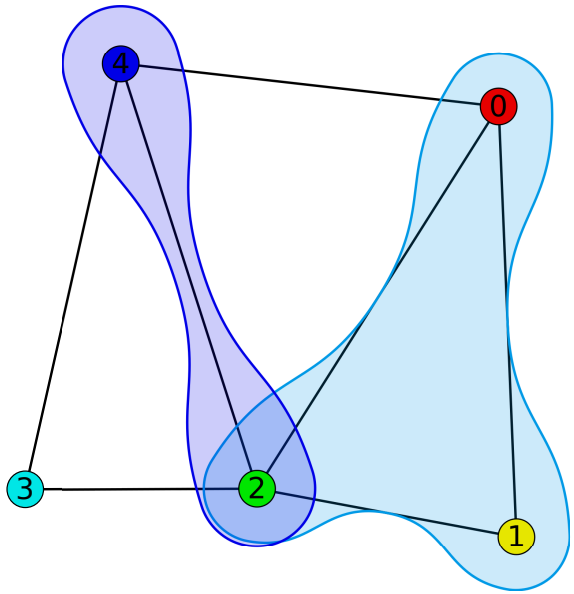
Extension du graphe quotient

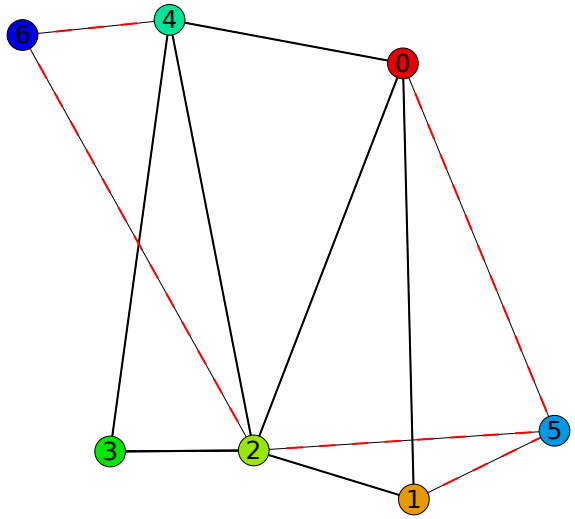
On s'intéresse uniquement au cas $M < N$.

- Le graphe quotient ne contient pas le bon nombre de sommets ($M \neq N$).
- On choisit un emplacement pour les parties supplémentaires.
- Le graphe quotient est enrichi en conséquence.

Exemple de repartitionnement 5×7 :







Programme linéaire pour optimiser TotalV

Programme linéaire

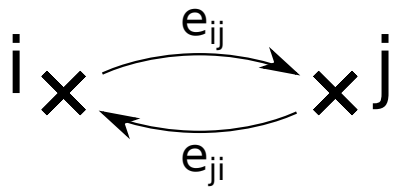
$$\text{minimiser} \quad \sum_{i,j} e_{ij}$$

Contraintes linéaires :

$$\forall i \in V, \quad v_i = \sum_j (e_{ji} - e_{ij})$$

Bornes :

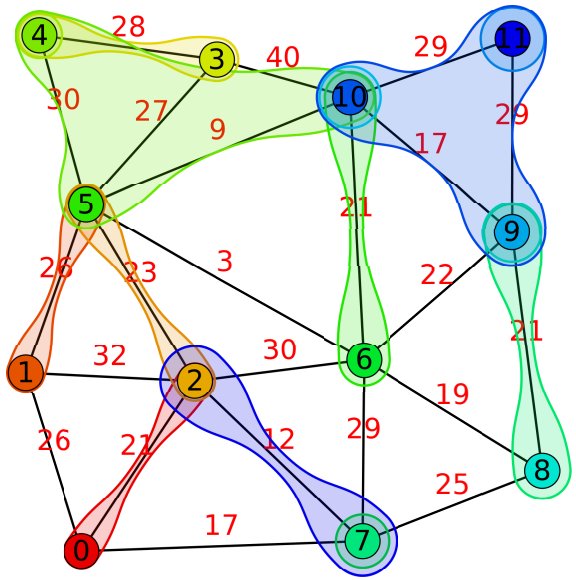
$$\forall i \in V, \quad v_i \leq d_i + ub$$
$$\forall (i,j) \in E, \quad e_{ij} \geq 0$$



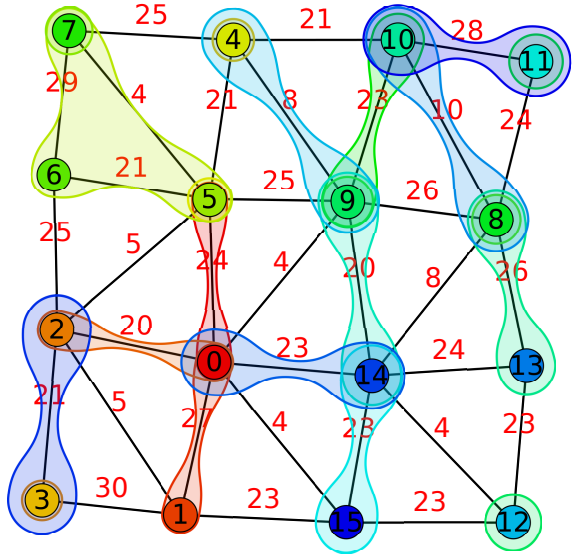
- d_i : variation de charge **souhaitée** pour la partie i
- v_i : variation de charge **calculée** pour la partie i
- ub : tolérance au déséquilibre

Le programme linéaire peut être étendu aux autres métriques MaxV, TotalZ et MaxZ ou à une combinaison linéaire de celles-ci.

Exemples de résultats



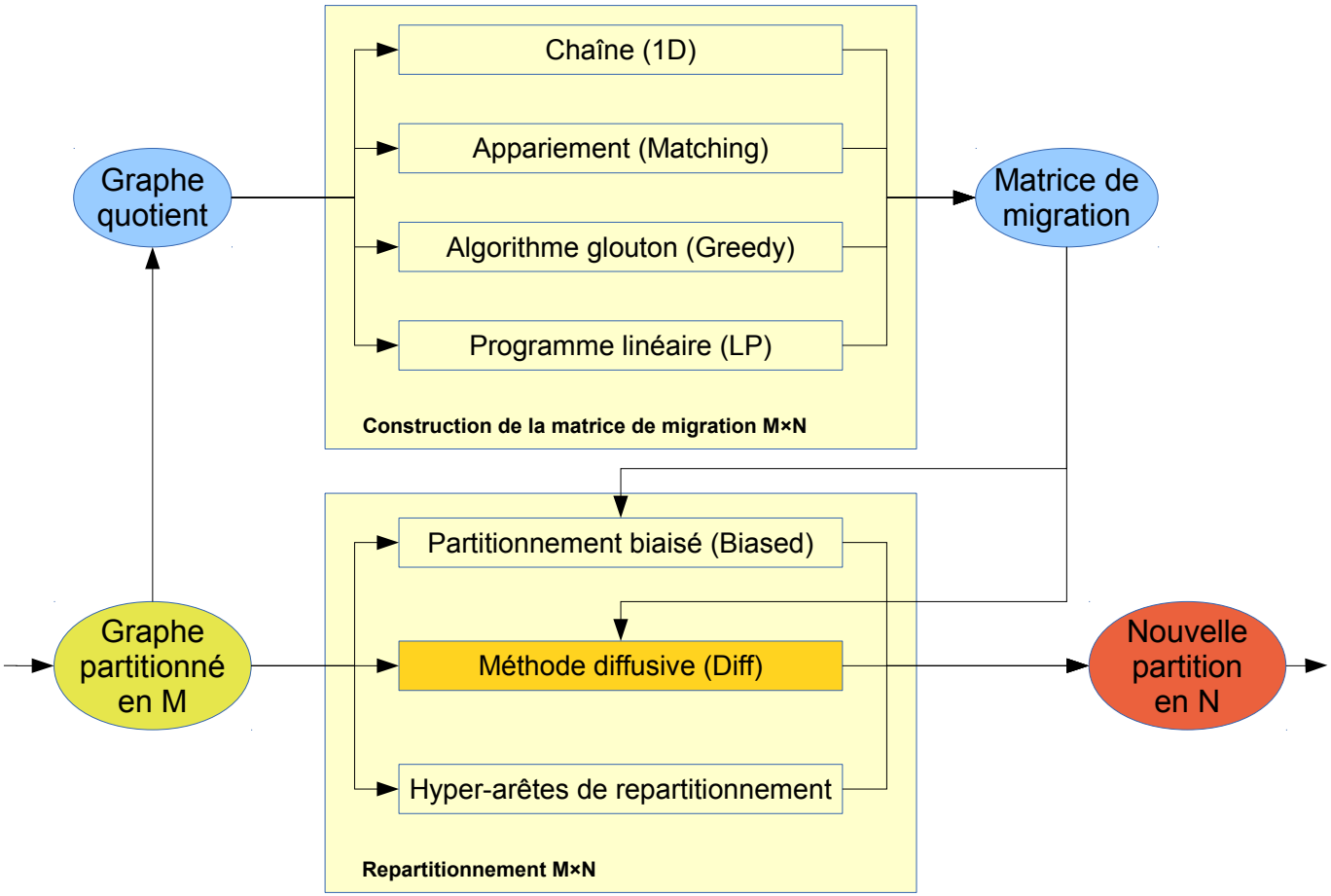
Repartitionnement 12×14



Repartitionnement 16×21

- Minimise les métriques de migration
- MaxZ reste bas grâce au graphe quotient enrichi

Repartitionnement diffusif



Repartitionnement diffusif

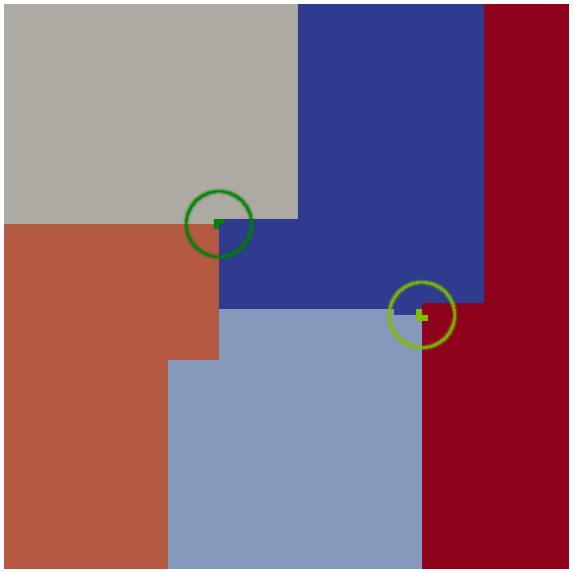
Pour chaque élément non nul de la matrice de migration, des sommets sont échangés le long de la frontière entre les parties concernées jusqu'à avoir migré le poids souhaité.

- Une heuristique de type FM est utilisée pour choisir les sommets à migrer.
- Ces migrations peuvent être effectuées :
  - par paires de parties (un ordonnancement est nécessaire),
  - globalement.

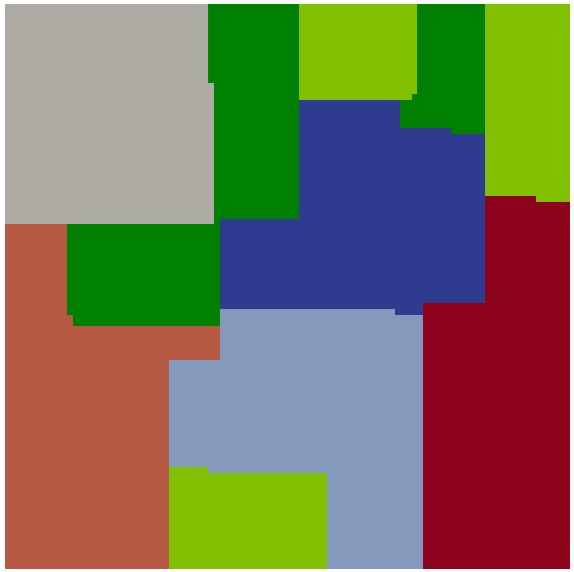
Problème

Dans le cas du repartitionnement $M \times N$, des parties peuvent ne pas avoir de frontière (elles n'existent pas dans l'ancienne partition).

Utilisation de graines



Diffusion avec graines



Diffusion sans graine

Solution

Ajouter des graines pour les parties initialement vides permet de créer des frontières à partir desquelles on fait grandir ces parties.

Algorithme de repartitionnement basé sur la diffusion

1. Ancienne partition en M parties et matrice de migration
2. Ajout de graines pour les nouvelles parties initialement vides
3. Ordonnancement des étapes de migration

Exemple de repartitionnement 5×7 d'une grille régulière 100×100 initialement déséquilibrée de 40 %

$$C = \begin{bmatrix} 2517 & 0 & 0 & 0 & 0 & \mathbf{168} & 0 \\ 0 & 2516 & 0 & 0 & 0 & \mathbf{1511} & 0 \\ 0 & 0 & 2397 & \mathbf{264} & 0 & \mathbf{599} & \mathbf{1438} \\ 0 & 0 & 0 & 2014 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2516 & 0 & \mathbf{840} \end{bmatrix}$$

1 Introduction

2 État de l'art
- Partitionnement
- Repartitionnement
- Problématique et positionnement

3 Repartitionnement $M \times N$
- Étude préliminaire
- Méthode de partitionnement biaisé
- Méthode diffusive

4 Résultats expérimentaux
- Influence du nombre de nouvelles parties
- Comparaison sur des graphes complexes

5 Conclusion et Perspectives

Méthodologie expérimentale

Méthodes de repartitionnement $M \times N$

- Méthode de partitionnement biaisé (*BiasedGreedyD*)
  arêtes de migration à 10
- Méthode de diffusion (*DiffTV*)

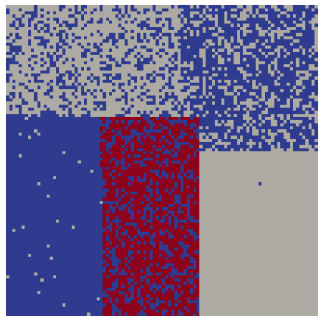
Comparaison avec des repartitionneurs actuels

- Scratch-Remap
- Zoltan
- ParMetis
- Scotch

Paramètres : tolérance au déséquilibre de 1 %, rapport coupe/migration à 1

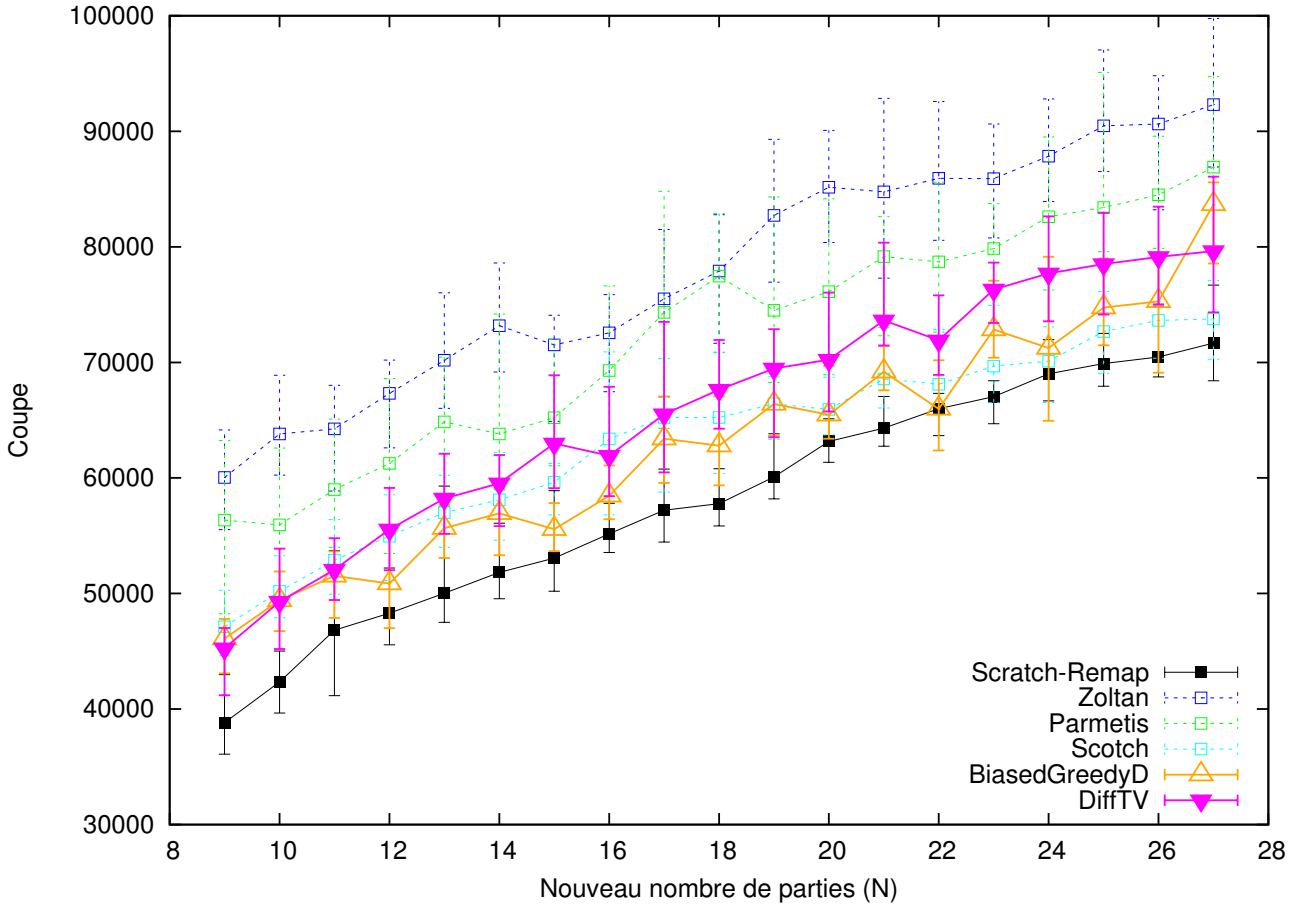
Les anciennes partitions sont déséquilibrées

- Changement du poids de certains sommets sélectionnés aléatoirement.
- Augmentation de la charge proportionnelle au nombre de ressources.



Répétées 10 fois

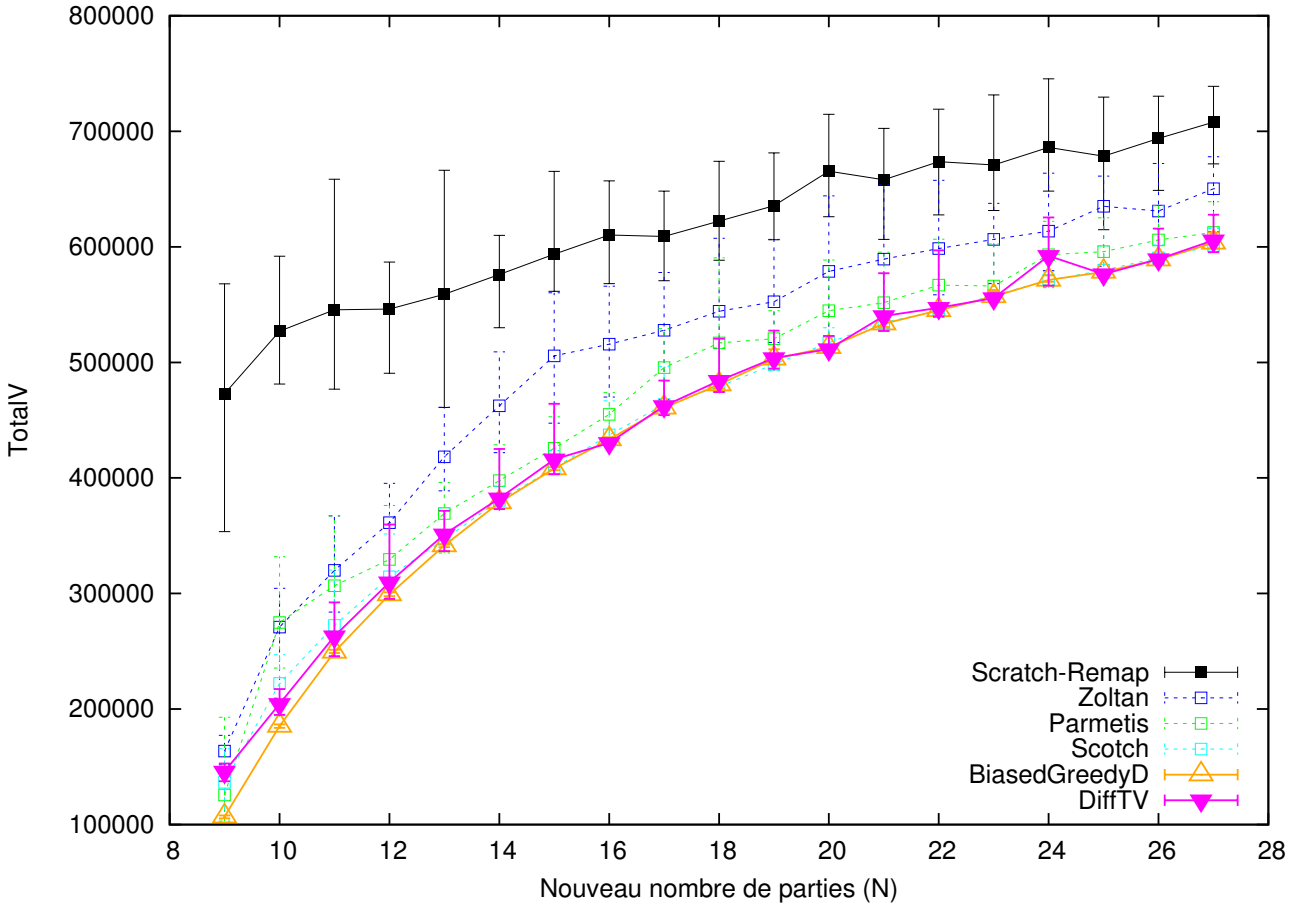
Influence du nombre de nouvelles parties



Grille 3D 100^3
$M = 8$
charge $+\frac{N}{8}$

$\rightarrow$ Coupe légèrement plus élevée que *Scratch-Remap*

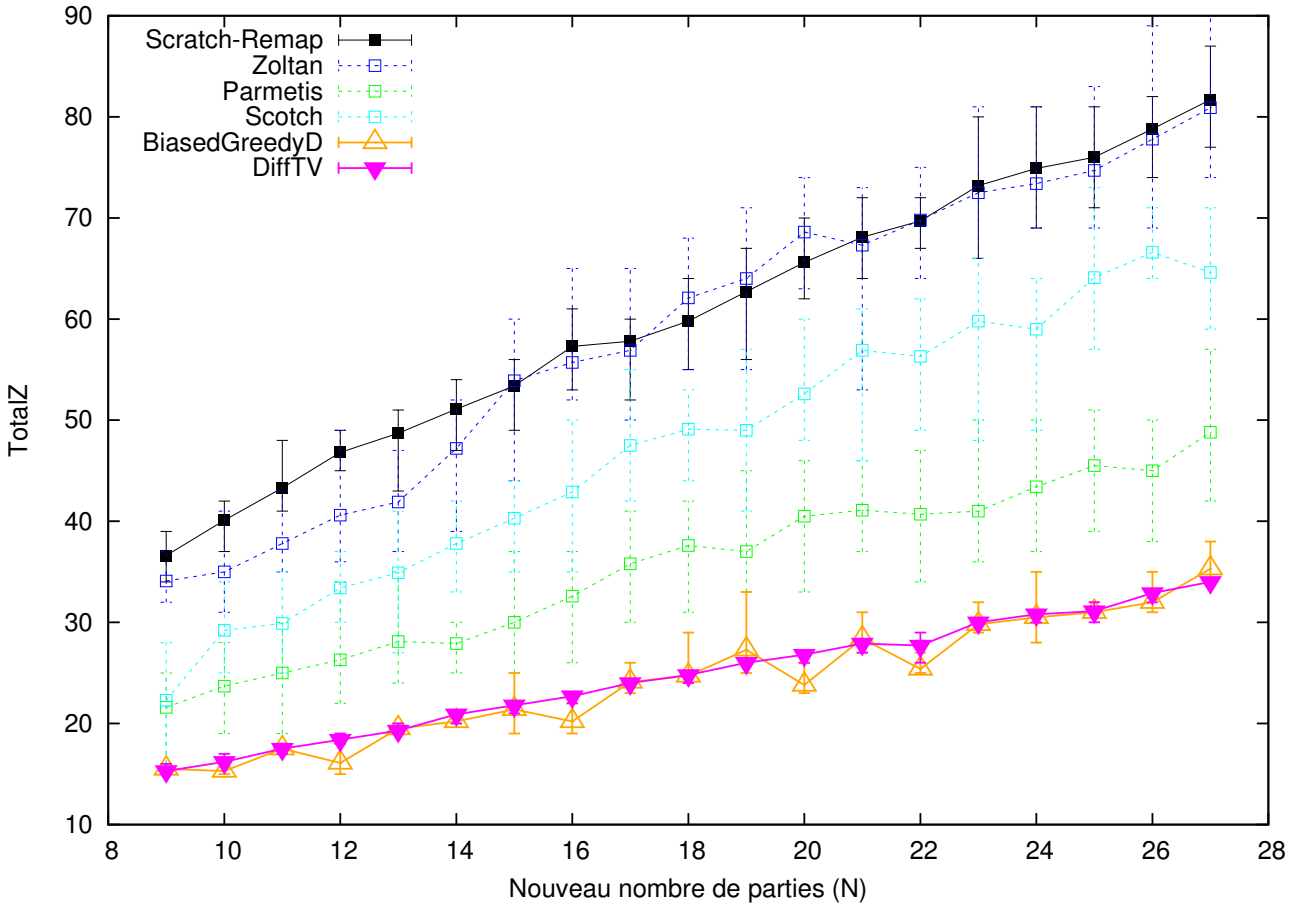
Influence du nombre de nouvelles parties



Grille 3D 100^3
$M = 8$
charge $+\frac{N}{8}$

$\rightarrow$ TotalV très bas mais se rapproche avec $N >> M$

Influence du nombre de nouvelles parties



Grille 3D 100^3
$M = 8$
charge $+\frac{N}{8}$

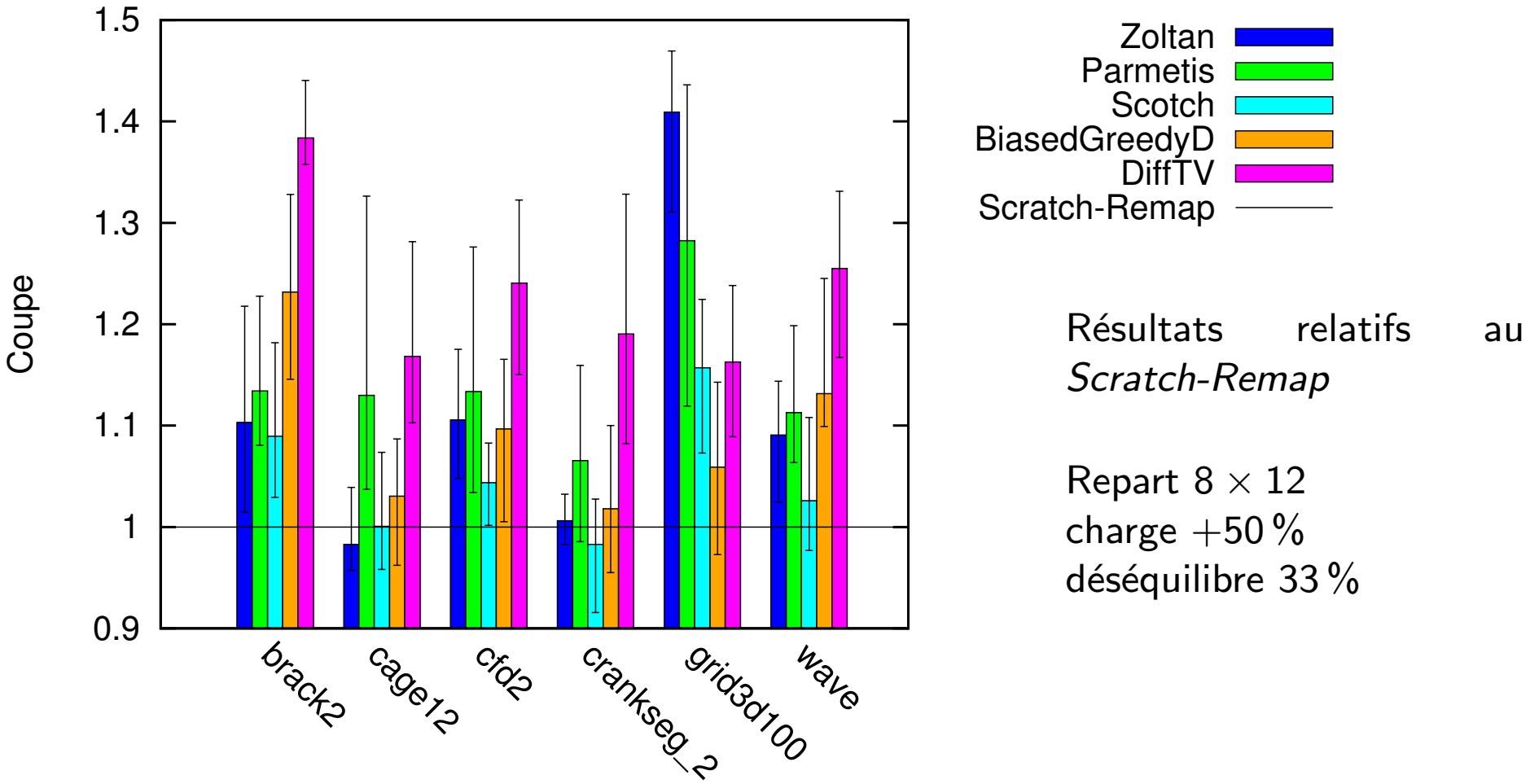
$\rightarrow$ TotalZ reste bas

Comparaison sur des graphes complexes

Repartitionnement 8×12 de graphes issus de la collection de matrices creuses de l'Université de Floride.

Graphe	Description	$\|V\|$	$\|E\|$	d
grid3d100	Grille 3D régulière	1 000 000	2 970 000	5,94
cfd2	Mécanique des fluides	123 440	1 482 229	24,02
crankseg_2	Problème structurel	63 838	7 042 510	220,64
brack2	Problème 2D/3D	62 631	366 559	11,71
wave	Problème 2D/3D	156 317	1 059 331	13,55
cage12	Électrophorèse d'ADN	130 228	951 154	14,61

Comparaison sur des graphes complexes



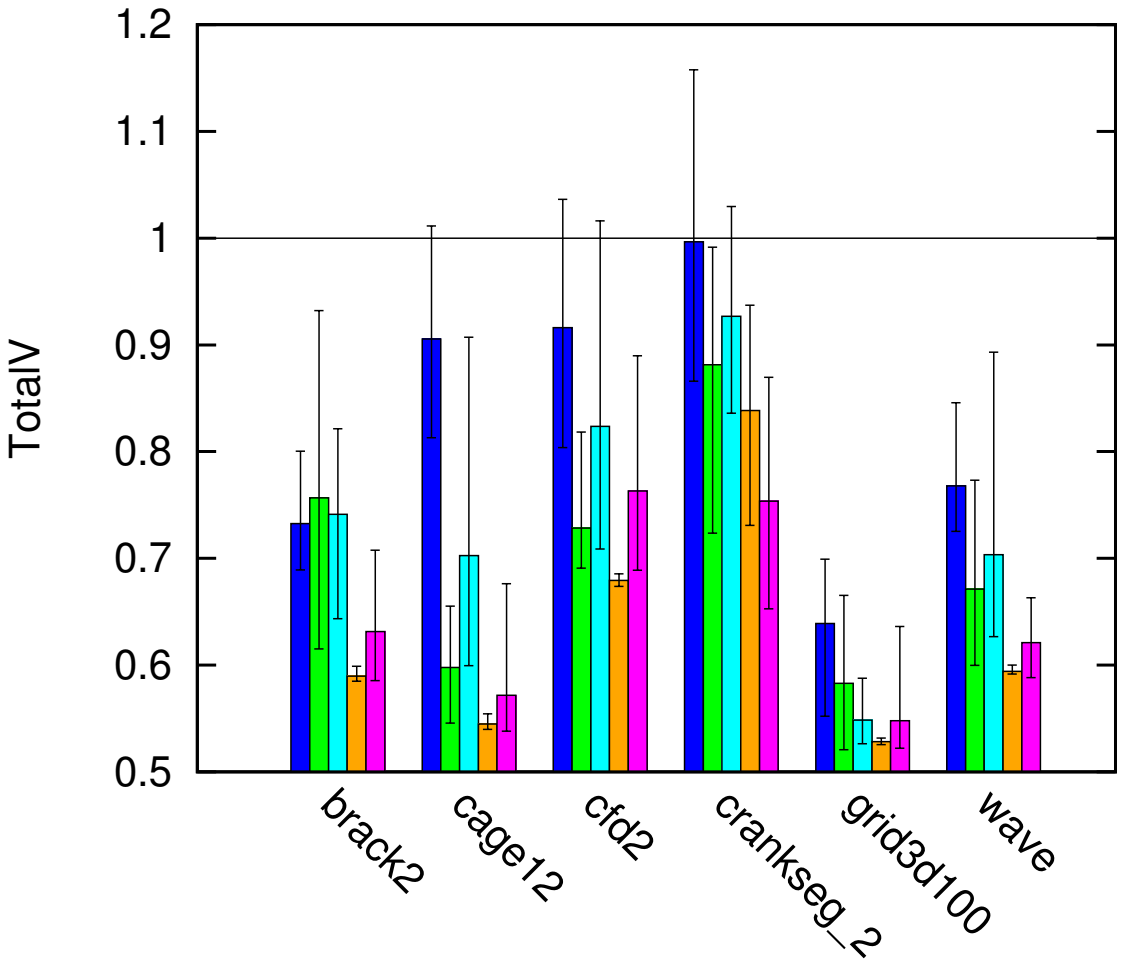
Résultats relatifs au *Scratch-Remap*

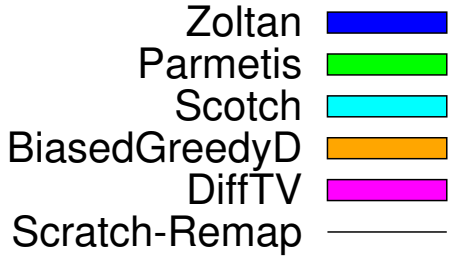
Repart 8×12
charge $+50\,\%$
déséquilibre $33\,\%$

$\rightarrow$ DiffTV donne une coupe élevée

$\rightarrow$ BiasedGreedyD au même niveau que les autres repartitionneurs

Comparaison sur des graphes complexes



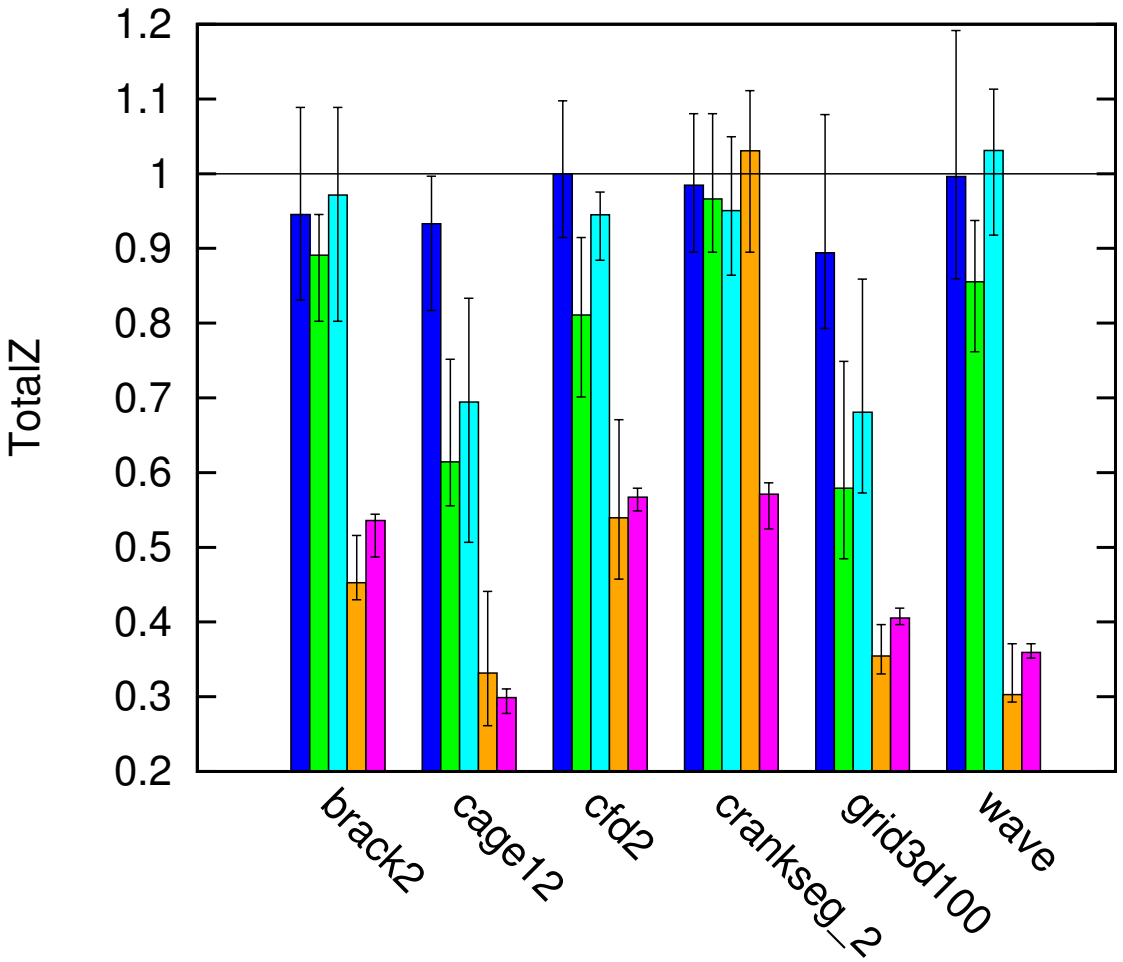


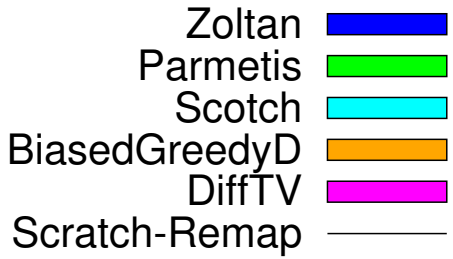
Résultats relatifs au *Scratch-Remap*

Repart 8×12
charge $+50\,\%$
déséquilibre $33\,\%$

$\rightarrow$ TotalV toujours très bas
$\rightarrow$ BiasedGreedyD meilleur que DiffTV

Comparaison sur des graphes complexes





Résultats relatifs au *Scratch-Remap*

Repart 8×12
charge $+50\,\%$
déséquilibre $33\,\%$

$\rightarrow$ TotalZ toujours très bas

Synthèse

- Coupe légèrement plus élevée que *Scratch-Remap* mais comparable aux autres repartitionneurs
- Migration bien optimisée
  mais de façon moins importante quand $N >> M$

Nos méthodes de repartitionnement $M \times N$ sont plus intéressantes quand le nombre de processeurs varie peu.

- BiasedGreedyD meilleur que DiffTV (coupe et migration)
- DiffTV plus stable (crankseg_2)

Temps de repartitionnement

- BiasedGreedyD utilise KGGGP (complexité $O(k|E|)$)
- Implémentation préliminaire de la diffusion trop lente (utilise le graphe entier)

1. Introduction
2. État de l'art
   - Partitionnement
   - Repartitionnement
   - Problématique et positionnement
3. Repartitionnement $M \times N$
   - Étude préliminaire
   - Méthode de partitionnement biaisé
   - Méthode diffusive
4. Résultats expérimentaux
   - Influence du nombre de nouvelles parties
   - Comparaison sur des graphes complexes
5. Conclusion et Perspectives

Conclusion

Deux méthodes de repartitionnement $M \times N$ présentées :

- Algorithme glouton et partitionnement biaisé
- Diffusion avec un programme linéaire optimisant TotalV

Objectifs :

- Nouvelle partition équilibrée
- Coupe légèrement plus élevée que *Scratch-Remap*
- Migration bien optimisée

  Surtout si M et N sont proches
- Temps de partitionnement assez lent
  - Complexité du KGGGP
  - Diffusion effectuée sur le graphe complet

Code disponible sur la forge INRIA

`https://gforge.inria.fr/frs/download.php/33277/lbc2-r1844.tar.gz`

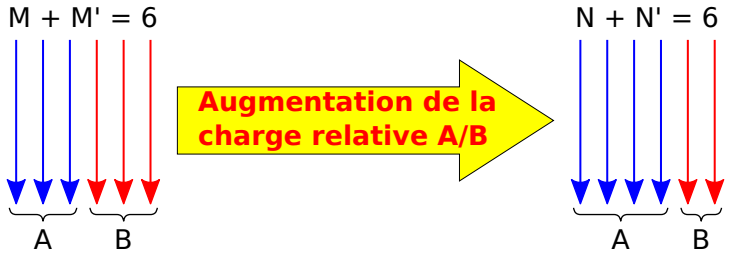
Perspectives I

- Parallélisation des méthodes
- Repartitionnement $M \times N$ avec partitionnement biaisé
  - Améliorer la complexité du KGGGP
    Considérer moins de possiblités de parties par sommet
  - Biaiser les heuristiques au lieu d'enrichir le graphe (à la *Scotch*)
- Repartitionnement $M \times N$ avec diffusion
  - Amélioration de l'enrichissement du graphe quotient
    - Pour diminuer $\textsc{TotalV}$
    - Traiter le cas $N < M$
  - Étudier la minimisation d'une combinaison des différentes métriques
  - Intégrer la diffusion dans un partitionnement multi-niveaux (à la *ParMetis*)

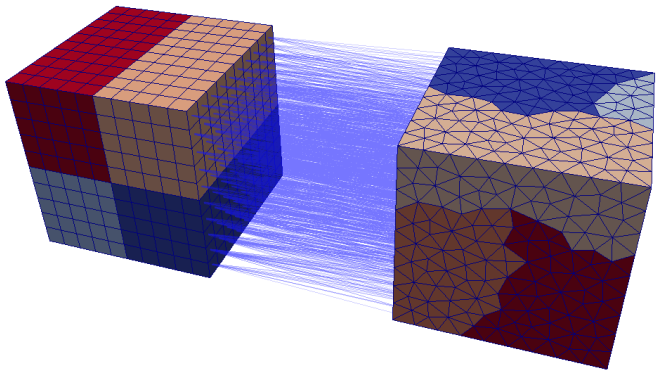
Perspectives II

Équilibrage de codes couplés

- Équilibrage des codes l'un par rapport à l'autre



- Optimisation des communications de couplages
  - Le volume de communication est fixe (pas d'optimisation de TotalV).
  - Optimisation du schéma de communication : TotalZ, MaxZ et MaxV



1. Introduction
2. État de l'art
   - Partitionnement
   - Repartitionnement
   - Problématique et positionnement
3. Repartitionnement $M \times N$
   - Étude préliminaire
   - Méthode de partitionnement biaisé
   - Méthode diffusive
4. Résultats expérimentaux
   - Influence du nombre de nouvelles parties
   - Comparaison sur des graphes complexes
5. Conclusion et Perspectives

Bibliographie I

Cevdet Aykanat, B. Barla Cambazoglu, Ferit Findik et Tahsin Kurc :
Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids.
J. Parallel Distrib. Comput., 67:77–99, janvier 2007.

U. V. Çatalyürek, E. G. Boman, K. D. Devine, D. Bozdağ, R. Heaphy, et L. A. Fisk :
Hypergraph-based dynamic load balancing for adaptive scientific computations.
Proceedings of 21st International Parallel and Distributed Processing Sympsosium (IPDPS), 2007.

Bibliographie II

C. M. Fiduccia et R. M. Mattheyses :
A linear-time heuristic for improving network partitions.
19th Design Automation Conference, pages 175–181, 1982.

Y. F. Hu et R. J. Blake :
An optimal dynamic load balancing algorithm.
Rapport technique, Daresbury Laboratory, 1995.

Bruce Hendrickson, Robert W. Leland et Rafael Van Driessche :
Skewed graph partitioning.
In Eighth SIAM Conf. Parallel Processing for Scientific Computing, 1997.

Bibliographie III

Saeed Iqbal et Graham F. Carey :
Performance analysis of dynamic load balancing algorithms with variable number of processors.
Journal of Parallel and Distributed Computing, 65(8):934 – 948, 2005.

B. W. Kernighan et S. Lin :
An efficient heuristic procedure for partitioning graphs.
Bell System Technical Journal, 49:291–307, février 1970.

Henning Meyerhenke, Burkhard Monien et Stefan Schamberger :
Graph partitioning and disturbed diffusion.
Parallel Comput., 35(10-11):544–569, octobre 2009.

Bibliographie IV

Chao-Wei Ou et Sanjay Ranka :
Parallel incremental graph partitioning using linear programming.
In Proceedings Supercomputing '94, pages 458–467, 1994.

François Pellegrini :
A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries.
In T. Priol A.-M. Kermarrec, L. Bougé, éditeur : *Euro-Par 2007 Parallel Processing*, volume 4641 de *Lecture Notes in Computer Science*, pages 195–204, Rennes, France, août 2007. Springer.

Bibliographie V

James D. Teresco, Karen D. Devine et Joseph E. Flaherty : Partitioning and dynamic load balancing for the numerical solution of partial differential equations.
In Timothy J. Barth, Michael Griebel, David E. Keyes, Risto M. Nieminen, Dirk Roose, Tamar Schlick, Are Magnus Bruaset et Aslak Tveito, éditeurs : *Numerical Solution of Partial Differential Equations on Parallel Computers*, volume 51 de *Lecture Notes in Computational Science and Engineering*, pages 55–88. Springer Berlin Heidelberg, 2006.