Contributions à la parallélisation de méthodes de type transport Monte-Carlo

Thomas Gonçalves

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Thomas Gonçalves

Thèse dirigée par **Jean-François Méhaut, Professeur, Université Grenoble Alpes**
et codirigée par **Frédéric Desprez, Directeur de recherche, Inria**

préparée au sein **du Laboratoire d'informatique de Grenoble**
et de **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Contributions à la parallélisation de méthodes de type transport Monte-Carlo

Thèse soutenue publiquement le **28 Septembre 2017**,
devant le jury composé de :

Monsieur Pierre Manneback
Professeur, Université de Mons, Rapporteur
Monsieur François Pellegrini
Professeur, Université de Bordeaux, Rapporteur
Monsieur William Jalby
Professeur, Université de Versailles Saint-Quentin-en-Yvelines, Président
Monsieur Jean-François Méhaut
Professeur, Université Grenoble Alpes, Directeur de thèse
Monsieur Frédéric Desprez
Directeur de recherche, Inria, Co-Directeur de thèse
Monsieur Marc Pérache
Ingénieur chercheur, C.E.A., Co-Directeur de thèse

Remerciements

Je remercie tout d'abord mes directeurs de thèse Jean-François Méhaut et Frédéric Desprez d'avoir accepté de diriger ma thèse. Je les remercie de m'avoir fait confiance. J'ai particulièrement apprécié leur aide précieuse pendant la phase de rédaction du manuscrit.

Je tiens à remercier très chaleureusement Marc Pérache, mon encadrant au sein du C.E.A.. Mes remerciements ne se limitent pas à son soutien scientifique et moral tout au long de cette thèse. Marc m'a suivi dès ma première année de master et il a grandement participé à mon évolution. Je le remercie sincèrement pour tout ce qu'il m'a apporté.

Je tiens à remercier tout particulièrement Pierre Manneback et François Pelligrini d'avoir accepté d'être rapporteurs de ma thèse et pour leurs retours sur mon manuscrit. Je tiens à remercier William Jalby pour m'avoir accueilli à Exascale avant ma thèse et pour avoir facilité la prolongation de mon contrat de thèse à l'UVSQ pour que je puisse finir la rédaction de mon manuscrit. Je remercie l'ensemble des membres de mon jury de thèse, Pierre Manneback, François Pelligrini, William Jalby, Jean-François Méhaut, Frédéric Desprez et Marc Pérache.

Je remercie le C.E.A. pour m'avoir accueilli durant ces trois années de thèse. J'adresse un remerciement tout particulier à Pierre Leca, Bruno Scheurer, Pierre-Franck Piserchia, Hervé Jourdren, Isabelle Visotto et Brigitte Sadoule. Je remercie également Thao Le et Denis Lubin pour leur accueil à Teratec. Je remercie Patrick Allal et Thao Le pour m'avoir aidé à préparer la salle de soutenance.

Je dis un grand merci à tous les thésards et stagiaires que j'ai côtoyé pendant mes trois années de thèse au C.E.A.. Je remercie tout particulièrement mes collègues thésards qui ont partagé mon bureau, Hugo Taboada avec qui j'ai partagé des soirées "craquage total de debug" au bureau, Hugo Brunie le thésard aux discussions politiques endiablées et Arthur Loussert, pour m'avoir initié aux échecs (le jeu bien sûr). J'adresse un sincère remerciement à Adrien Bernède pour avoir relu mon manuscrit et proposé des corrections. Je remercie également les membres du laboratoire Exascale pour leur accueil les derniers mois de ma thèse (pendant la phase de rédaction-craquage en plus). Je remercie tout particulièrement Ricardo Bispo-Vieira, Stéphane Bouhrour, Thomas Dionisi et Loïc Veyssière. Enfin, je remercie "bribri" la femme de ménage, qui m'a suivie du début jusqu'à la fin de ma thèse, pour sa bonne humeur et sa grande sympathie.

J'adresse un chaleureux remerciement à mes parents sans qui cela n'aurait jamais été possible. J'ai la chance d'avoir des parents formidables qui m'ont toujours soutenu dans mes choix parfois au prix de sacrifices personnels. J'espère pouvoir un jour leur rendre la pareille. Je remercie également le reste de ma famille et mes amis pour leur soutien.

Enfin, je remercie celle qui partage ma vie depuis plus de quatre années maintenant, Séverine Candau. Elle est même officiellement devenue ma femme en Novembre dernier. Je la remercie pour son soutien pendant les moments difficiles et pour son implication dans ma thèse également, j'imagine que ça n'a pas toujours été facile d'endurer les présentations de mes travaux et les relectures de mon manuscrit mais elle a toujours proposé son aide et je l'en remercie chaleureusement.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Dans de nombreux domaines tels que la physique, la chimie, la biologie ou encore la médecine, les simulations numériques permettent de résoudre sur ordinateur des problèmes complexes, à l'aide d'une modélisation. Dans le domaine de la simulation numérique, la précision des résultats de simulation est un enjeu important. Le modèle de simulation se résout par un schéma numérique. Le schéma numérique s'appuie, la plupart du temps, sur une discrétisation du domaine physique. L'augmentation du nombre d'éléments de simulation permet, en général, d'augmenter la précision de la simulation. Chaque élément de simulation est stocké dans l'espace mémoire de la plateforme de calcul. L'augmentation du nombre d'éléments implique alors une consommation mémoire élevée et un volume de calcul important. Le besoin de simulation précises nécessite l'exploitation de processeurs de calcul et de stockage mémoire distribué.

Les plateformes de calcul à mémoire distribuée se définissent, la plupart du temps, comme un ensemble de nœuds interconnectés par un réseau de communication. Chaque nœud possède ses cœurs de calcul et sa mémoire d'une capacité limitée. La parallélisation d'une simulation numérique sur de telles architectures consiste à ce que chaque cœur traite une partie du problème. Lorsqu'un cœur a besoin d'accéder à un élément de simulation stocké dans une mémoire distante, un mécanisme d'accès, appelé communication, est invoqué. Le calcul appliqué à un élément de simulation est plus efficace lorsque cet élément est stocké dans la mémoire du nœud de calcul auquel appartient le cœur (principe de localité). L'exécution parallèle sur de telles architectures nécessite alors la répartition de la charge de calcul et des données entre plusieurs ressources de calcul.

Lorsque plusieurs cœurs réalisent des calculs pendant que tous les autres cœurs sont en attente, la plateforme de calcul est sous-exploitée. Nous parlons dans ce cas d'une situation de déséquilibre de charge. Ces déséquilibres de charge peuvent apparaître lorsque les temps de traitements des éléments de simulation sont non-uniformes, par exemple lorsque les équations à résoudre sont différentes, ou lorsque la répartition des calculs entre les cœurs est non-uniforme. Lorsque les données de simulation ne sont pas équitablement réparties entre les nœuds de calcul, un déséquilibre de consommation mémoire apparaît. Nous parlons également de surconsommation mémoire. Cette surconsommation mémoire peut alors avoir pour conséquence l'arrêt total de la simulation, dû au dépassement de la capacité mémoire d'une ressource. Dans ce cas précis, l'application ne peut aller à son terme.

1.1 Motivations

Les applications de transport de particules Monte-Carlo [1][2] consistent à suivre les histoires de particules dans un domaine de simulation. Les applications de transport de particules Monte-

Carlo sont très impactées par des problématiques de répartition de charge de calcul et de donnée. En effet, la charge de calcul ne sont pas réparties de manière uniforme sur le domaine de simulation. De plus, cette répartition évolue dynamiquement au cours de la simulation.

Dans le domaine de la simulation numérique, plusieurs méthodes de parallélisation ont été définies et utilisées, principalement pour des applications régulières ou faiblement irrégulières. La répartition inégale des charges de calcul et des données rend les méthodes classiques de parallélisation inefficaces pour des applications trop irrégulières, comme le transport de particules Monte-Carlo. Le comportement des applications de transport de particules Monte-Carlo est très fluctuant au cours de la simulation. Par conséquent, l'exécution efficace nécessite la mise en place de mécanismes dynamiques de régulation de charge et de donnée.

Ces mécanismes dynamiques de régulation de charge et de donnée amènent plusieurs problématiques. La première consiste à détecter et exprimer les situations de déséquilibre. La seconde consiste à définir des actions de correction pour améliorer la régulation de charge et de donnée. Ces mécanismes de régulation de charge et de donnée sont basés sur des communications. Ces communications ont un coût qui devra être compensé par une meilleure régulation de charge et donc une meilleure performance. La question qui se pose alors est la suivante :

Comment garantir une bonne répartition dynamique des charges de calcul et des données, sur des plateformes massivement parallèles, en minimisant le coût de communication ?

1.2 Contributions

Pour résoudre la problématique de régulation de charge et de donnée du transport de particules Monte-Carlo en contexte massivement parallèle, nous avons procédé en plusieurs étapes. La première a consisté à identifier les difficultés de parallélisation de ce type d'applications. Une approche, reposant sur des techniques de partitionnement, a ensuite été proposée. Enfin, nous avons défini une approche dynamique plus adaptée à la problématique que nous avons posée.

Identification des limitations de méthodes classiques de parallélisation : Dans le domaine du calcul haute performance, des méthodes de parallélisation sont couramment utilisées. Appliquées au transport de particules Monte-Carlo, ces méthodes ne parviennent pas à répondre efficacement à la problématique de répartition de charge de calcul et de donnée. Nous proposerons alors une analyse théorique et expérimentale de ces méthodes permettant d'identifier et de caractériser leurs limitations. L'analyse théorique consistera à étudier la répartition de la charge de calcul et des données, en fonction du nombre de ressources, des méthodes de parallélisation que nous aurons introduites. Cette analyse sera validée par une étude expérimentale sur un ensemble de cas test représentatifs.

Régulation de charge et de donnée par partitionnement de graphe : Les techniques de partitionnement permettent de répondre à des problématiques de répartition d'éléments selon un ensemble de critères. Nous proposerons d'abord un modèle de graphe permettant de représenter la charge de calcul et les données. Nous proposons ensuite une méthode, permettant d'exploiter les techniques de partitionnement, afin de réaliser une régulation dynamique des charges de calcul et des données. Cette méthode s'appuiera sur le modèle de graphe introduit précédemment.

Régulation dynamique de charge et de donnée : La minimisation des coûts de régulation et la gestion de la forte dynamicité du transport de particules Monte-Carlo nous amènera à considérer de

manière dynamique et locale la répartition de la charge de calcul et des données. Nous proposerons donc un nouveau modèle de graphe dynamique et local. Nous proposerons alors une approche de régulation dynamique de charge et de donnée reposant sur ce modèle de graphe.

1.3 Plan du manuscrit

Le manuscrit s'organise de la façon suivante : Le contexte scientifique et applicatif sera présenté au chapitre 2. Ce chapitre présentera, en particulier, les enjeux principaux du calcul haute performance et la méthode du transport de particules Monte-Carlo. L'analyse théorique et expérimentale des méthodes classiques de parallélisation sera détaillée au chapitre 3. Nous conclurons ce chapitre par un bilan des forces et des faiblesses des différentes méthodes. La méthode par partitionnement et le modèle de graphe associé seront présentés au chapitre 4. Le chapitre 5 introduira l'approche dynamique incluant la présentation du modèle de graphe dynamique et des mécanismes de régulation dynamique de charge et de donnée. S'en suivra une analyse expérimentale de cette approche au chapitre 6. Le chapitre 7 permettra de présenter les différents travaux qui se sont intéressés à la problématique de régulation de charge de calcul et de donnée, en particulier dans le domaine du transport de particules Monte-Carlo. Une comparaison avec notre approche sera réalisée. Enfin, au chapitre 8 nous conclurons et explorerons des perspectives.

Chapitre 2

Contexte

Dans ce chapitre, nous commencerons par présenter le cas d'étude en section 2.1. Nous présenterons le transport de particules Monte-Carlo [1][2]. Nous définirons ainsi notre cadre d'étude et en particulier nous situerons la problématique dans le processus du transport de particules Monte-Carlo. Les applications de transport de particules Monte-Carlo nécessitent une grande capacité de stockage mémoire et une puissance de calcul conséquente. Les architectures parallèles à mémoire distribuée sont de bonnes candidates pour ce type d'applications. Nous ferons une présentation des architectures parallèles en section 2.2. Nous déterminerons alors les possibilités qu'elles offrent mais aussi les contraintes qu'elles imposent. L'aspect logiciel, en particulier les interfaces de programmation parallèle, fera l'objet d'une attention particulière. Après avoir donné une définition des différents types d'extensibilités parallèles, nous terminerons cette section par une définition des méthodes de parallélisation dites classiques. Nous exprimerons alors les spécificités de chacune de ces méthodes. Enfin, en section 2.3, nous nous intéresserons aux techniques de partitionnement. Ces techniques feront l'objet d'une présentation détaillée permettant de se familiariser avec leurs problématiques. Nous terminerons ce chapitre par une présentation de la méthodologie d'expérimentation utilisée pendant cette thèse, en section 2.4.

2.1 Le transport de particules Monte-Carlo

Le transport de particules est un problème connu en physique des particules [3]. Il consiste à étudier le comportement des particules dans un domaine physique et en particulier leurs interactions avec la matière. Le transport de particules est utilisé dans de nombreux domaines tels que la physique nucléaire, le médical, etc. Nous distinguons différents types de particules selon les domaines : proton, neutron, électron, photon, etc. Ces particules se déplacent et subissent des interactions avec la matière. Les applications de transport de particules se distinguent en deux catégories : les approches déterministes et les approches statistiques. Nous nous intéressons aux approches statistiques.

2.1.1 Introduction

Les approches statistiques consistent à étudier le comportement d'un grand nombre de particules indépendantes entre elles. À chaque particule est assignée une graine aléatoire unique permettant de déterminer le comportement de la particule. Les interactions subies par chaque particule sont soumises à des lois de probabilité. Les lois de probabilité peuvent varier d'une zone du domaine physique à l'autre, impliquant des interactions différentes. Chaque particule peut subir différents

types d'interactions : collision, absorption, fuite du domaine. La collision décrit le phénomène qui se produit lorsqu'une particule interagit avec la matière, la particule change alors de direction et de vitesse. L'absorption intervient quand une particule devient trop faible en masse ou en énergie, elle est alors absorbée par le domaine physique. La fuite du domaine désigne, comme son nom l'indique, le fait qu'une particule sorte du domaine de simulation. Chaque interaction est aussi appelée évènement. L'ensemble des interactions ou évènements d'une particule est appelé histoire d'une particule. La méthode de transport Monte-Carlo consiste alors à étudier les histoires d'un ensemble de particules dans un domaine physique dans le but d'exprimer un comportement réel, approché, du système.

La complexité des applications de transport de particules Monte-Carlo est conditionnée par le modèle physique simulé. Les interactions particules-matière simulées dépendent en particulier du type de particules utilisé ainsi que du milieu dans lequel elles se propagent. De plus, les applications peuvent être décomposées en plusieurs parties. Par exemple, une application donnée peut être scindée en deux phases distinctes : une phase de transport de particules et une phase hydrodynamique utilisant les données calculées par la phase de transport. Nous nous sommes focalisés sur la phase de transport de particules. Nous avons étudié une application prototype de transport de particules Monte-Carlo uniquement focalisée sur la phase de transport. Cette application est représentative des problématiques d'équilibrage de charge. Cette application met en œuvre l'étude de particules, insérées en début de simulation, se déplaçant dans un domaine physique. Les particules et le domaine physique sont modélisés par des éléments de simulation que nous allons maintenant détailler.

2.1.2 Définitions

Le domaine physique que nous souhaitons modéliser est continu en espace. Une modélisation continue nécessite une capacité de stockage mémoire hors de portée des machines actuelles. Pour contourner cette difficulté, les simulations numériques utilisent des éléments de simulation appelés maillages. Le maillage est une représentation discrétisée d'un espace continu. Il est existe plusieurs types de maillages : réguliers, irréguliers, cartésiens, etc. Le maillage est constitué d'un ensemble de mailles représentant chacune une partie du domaine physique simulé.

Nous souhaitons simuler les interactions subies par les particules dans le domaine physique. Ainsi, lorsqu'une particule traverse une maille, cette particule est susceptible de subir des interactions. Les probabilités diffèrent d'une maille à l'autre. Ces probabilités sont calculées à partir de donnée géométriques décrivant la composition du domaine de simulation. Nous pouvons également attribuer à chaque type d'interaction une loi de probabilité différente. Nous assignons à chaque maille m, σ_m, une interactivité définissant les lois de probabilité régissant le comportement des particules traversant cette maille. Les lois de probabilités sont aussi appelées sections efficaces.

À chaque maille, est assignée une liste de particules correspondant aux particules présentes sur la maille à un instant donné. Nous notons ρ_m le nombre de particules présentes. Chaque maille possède également un ensemble de donnée statistiques décrivant les interactions ayant eu lieu dans cette maille. Nous définissons C_m, le nombre de collisions subies par les particules ayant traversé la maille. Chaque particule présente sur une maille peut être absorbée par la maille. Dans notre cas, chaque particule dépose systématiquement une masse dans chaque maille qu'elle traverse. Nous appelons contribution la masse déposée dans une maille. Nous notons Ω_m, la masse totale absorbée par la maille m.

Chaque particule possède une vitesse et une direction qui lui sont propres. Nous notons ν_p et $\vec{d}_p$ respectivement la vitesse et la direction d'une particule p. Notons que $\vec{d}_p$ correspond à un vecteur

normalisé. L'ensemble des coordonnées de la particule définissant sa position dans le domaine est noté x_p. Une particule p est dotée d'une masse, notée ω_p. Lorsque cette masse devient trop faible la particule est absorbée. Nous notons $\underline{\omega}$ la masse minimale en dessous de laquelle une particule est absorbée. La masse initiale d'une particule est notée $\overline{\omega}$. Enfin, les tirages aléatoires permettant de déterminer les interactions subies par une particule sont réalisés à l'aide de graines aléatoires. Nous nommons λ_p, la graine aléatoire assignée à la particule p.

Le transport de particules Monte-Carlo consiste à étudier l'histoire des particules. Les différentes propriétés des mailles et des particules peuvent être modifiées au fil des évènements qui se produisent. L'aspect évolutif des propriétés des éléments de simulation est modélisé par une approche itérative.

2.1.3 Modèle itératif

Nous souhaitons étudier l'histoire d'un ensemble de particules. L'étude des histoires des particules nécessite un processus itératif. Le nombre d'itérations dépend de plusieurs paramètres. En particulier, il dépend de la condition d'arrêt. Cette condition d'arrêt dépend de la physique simulée. Dans le cas de notre application, la simulation s'arrête lorsque toutes les particules ont fui le domaine de simulation ou ont été absorbées par le domaine de simulation.

L'algorithme 1 présente le modèle itératif du transport de particules Monte-Carlo. La boucle de la ligne 2 correspond aux itérations Monte-Carlo. À chaque itération, nous procédons de la même façon. Les évènements survenus au cours de l'itération sont d'abord calculés (ligne 3). Ensuite, nous mettons à jour la liste des particules de chaque maille, des particules ayant potentiellement migrées d'une maille à l'autre (ligne 4). Enfin, nous évaluons la condition d'arrêt en ligne 5.

Algorithme 1 : Modèle itératif du transport de particules Monte-Carlo.

```
1 fin_application ← Faux
  /* L'ensemble des itérations de la simulation                    */
2 tant que !fin_application faire
      /* Contributions des particules, interactions, etc.          */
3     Traitement_particules( Maillage, Particules )
      /* Mise à jour de la liste des particules                    */
4     Mise_a_jour_listes( Maillage, Particules )
      /* Fin de la simulation ?                                    */
5     fin_application ← Evaluation_fin_simulation( Maillage, Particules )
      /* iteration t ← t + 1                                       */
6 fin
```

Les évènements d'une itération influent sur les évènements des itération suivantes. En effet, les évènements survenus au cours d'une itération peuvent modifier la répartition des particules dans le domaine physique. Nous allons maintenant décrire le déroulement d'une itération Monte-Carlo et en particulier des mécanismes événementiels.

2.1.4 Itération de transport de particules Monte-Carlo

Chaque particule du système est traitée à chaque itération. Une itération Monte-Carlo se déroule en plusieurs étapes. La première consiste à calculer la contribution de la particule à la maille à laquelle elle appartient (dépôt de masse). La deuxième étape consiste à déterminer si une interaction a eu lieu entre la particule et la matière représentée par la maille, puis de calculer de nouvelles directions et vitesse le cas échéant. Enfin, la particule se déplace en fonction de sa vitesse et de sa

direction. Ses nouvelles coordonnées permettront de déterminer dans quelle maille elle se trouvera à l'itération suivante.

L'algorithme 2 détaille les différentes étapes de l'itération Monte-Carlo. Pour chaque particule, nous commençons par calculer la contribution de la particule (ligne 3). La fonction $f_\omega \in [0, 1]$ définit la proportion de masse que chaque particule dépose dans une maille. La masse totale absorbée par la maille et la masse de la particule sont mises à jour (lignes 4 et 5). Nous évaluons, en ligne 6, si la masse de la particule est suffisante pour continuer à la suivre. Si ce n'est pas le cas, la particule est absorbée et nous passons à la particule suivante. Ensuite, nous évaluons les interactions particule matière. Nous commençons par calculer la probabilité d'interaction $p_{int} \in [0, 1]$ (ligne 11). La fonction $f_i \in [0, 1]$ définit la probabilité d'interaction pour une maille donnée. Nous calculons une valeur aléatoire à l'aide de la graine aléatoire de la particule (ligne 12). La fonction $f_a \in [0, 1]$ renvoie un réel aléatoire en fonction d'une graine passée en paramètre. Nous testons alors si une interaction s'est produite (ligne 13). En cas d'interaction, de nouvelles composantes de direction et de vitesse sont tirées aléatoirement. Enfin, nous effectuons le déplacement de la particule en fonction de sa direction et de sa vitesse (ligne 18). Soit x_D définissant l'ensemble des coordonnées du domaine de simulation. Nous évaluons si la particule demeure dans le domaine de simulation (ligne 19). Si la particule a fui le domaine de simulation, elle n'est plus suivie.

L'ensemble des évènements subis par une particule définit son histoire. Les histoires des particules sont différentes les unes par rapport aux autres. Premièrement, les interactions sont aléatoires. Deuxièmement, les particules n'ont pas nécessairement les mêmes vitesses et directions initiales. Troisièmement, elles ne sont pas forcément insérées au même endroit du domaine. Elles ne rencontrent alors pas les mêmes mailles. Les propriétés des mailles qu'une particule rencontre influent considérablement sur l'histoire de la particule. Nous proposons d'illustrer par un exemple l'influence des propriétés d'interaction des mailles sur les histoires des particules.

2.1.5 Exemple

La figure 2.1 présente un domaine découpé en quatre zones avec des interactivités différentes. En commençant par la gauche, la première zone possède une interactivité relativement faible. La deuxième zone possède en revanche une interactivité plutôt élevée. La troisième zone est celle ayant l'interactivité la plus faible. A l'inverse, la dernière zone possède l'interactivité la plus élevée. Nous avons inséré 8 particules sur le bord gauche du domaine. Nous leur avons donné les mêmes directions et masses initiales mais des vitesses aléatoires. Chaque particule possède sa propre graine aléatoire.

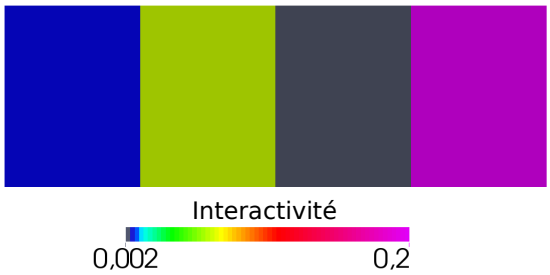


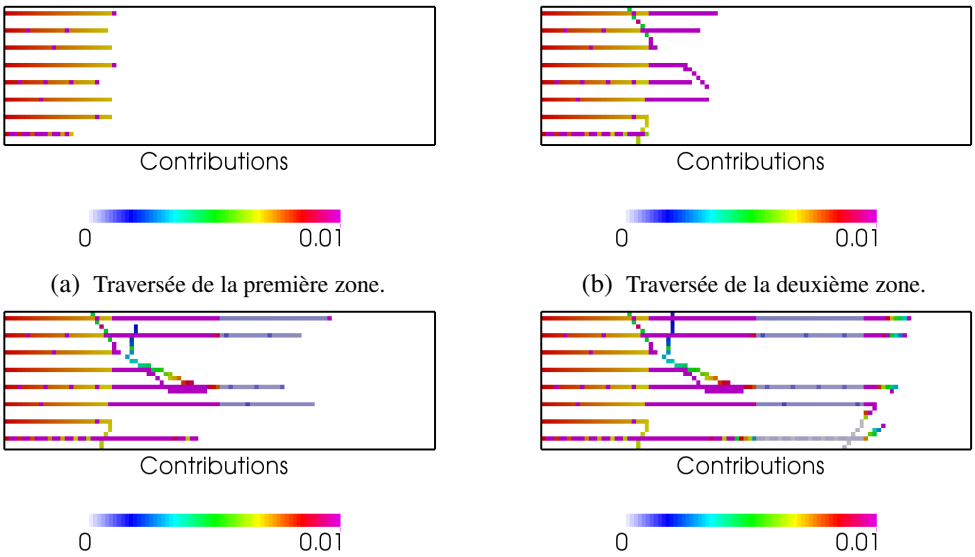
FIGURE 2.1 – Interactivité du domaine.

La figure 2.2 illustre les histoires des particules en représentant les contributions des particules sur les mailles à 4 itérations différentes (ordre chronologique). La sous-figure 2.2a correspond aux contributions des particules pendant la traversée de la première zone. Nous pouvons constater que,

Algorithme 2 : Déroulement d'une itération Monte-Carlo.

```
   /* Pour chaque maille à l'itération t                                  */
1  pour chaque maille m^t faire
       /* Pour chaque particule de la maille à l'itération t              */
2      pour chaque p^t ∈ m^t faire
           /* Contribution de la particule                                */
3          ω ← f_ω(σ_m) ω_p^t
4          Ω_m ← Ω_m + omega
5          ω_p^{t+1} ← ω_p^t − ω
           /* Masse suffisante ?                                          */
6          si ω_p^{t+1} < ω̲ alors
               /* La particule n'est plus suivie, la maille absorbe la
                  particule                                                */
7              Ω_m ← Ω_m + ω_p^{t+1}
8              ω_p^{t+1} ← 0
               /* Particule suivante                                      */
9              continue
10         fin
           /* Évaluation des interactions                                 */
11         p_int ← f_i(σ_m)
12         α ← f_a(λ_p)
           /* Interaction particule matière ?                             */
13         si α < p_int alors
14             C_m ← C_m + 1
               /* Nouvelle vitesse et nouvelle direction aléatoires       */
15             d⃗_p ← f_d(λ_p)
16             ν_p ← f_ν(λ_p)
17         fin
           /* Déplacement de la particule                                 */
18         x_p ← x_p + d⃗_p ν_p
           /* Toujours dans le domaine ?                                  */
19         si x_p ∉ x_D alors
               /* La particule n'est plus suivie                          */
20             ω_p^{t+1} ← 0
21         fin
22     fin
23 fin
```

en raison de vitesses choisies aléatoirement, les particules n'ont pas effectué la même distance. En revanche, aucune particule n'a changé de direction. La sous-figure 2.2b présente les contributions des particules au cours de la traversée de la deuxième zone. Nous observons, en premier lieu, que trois particules ont changé de direction, deux d'entre elles ont même fui le domaine de simulation. Ensuite, nous observons une hausse des contributions des particules. L'interactivité plutôt élevée de cette zone a donc eu un impact sur les histoires des particules. La traversée de la troisième zone est représentée par la sous-figure 2.2c. Nous remarquons que les particules traversent cette zone sans changer de direction tout en ayant de faibles contributions. Ce constat met en lumière la faible interactivité de cette zone. Enfin, la sous-figure 2.2d présente les contributions des particules pendant la traversée de la dernière zone. Premièrement, nous observons qu'aucune particule n'a réussi à traverser. En effet, leurs masses étaient réduites à leur arrivée dans cette zone et la forte interactivité de la zone a eu pour conséquence une absorption des particules. Deuxièmement, nous remarquons que 3 particules sur les 5 ayant atteint cette zone ont interagi. Ce pourcentage élevé montre encore un peu plus l'interactivité de cette ultime zone du domaine.



(a) Traversée de la première zone.

(b) Traversée de la deuxième zone.

(c) Traversée de la troisième zone.

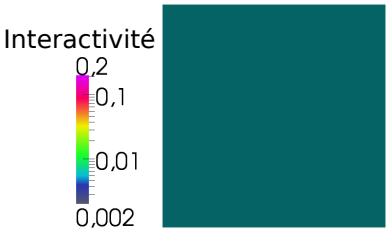
(d) Traversée de la dernière zone.

FIGURE 2.2 – Histoires des particules.

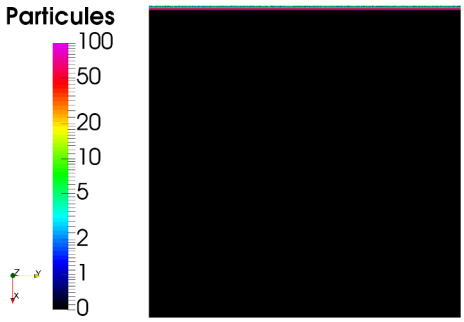
2.1.6 Cas test

Pour mener notre étude, nous nous appuierons sur un ensemble de cas test. Les domaines physiques simulés sont en deux dimensions. Les particules sont insérées à l'initialisation du système, aucune particule n'est réinsérée pendant la simulation. Nous avons choisi trois cas test ayant chacun des propriétés différentes.

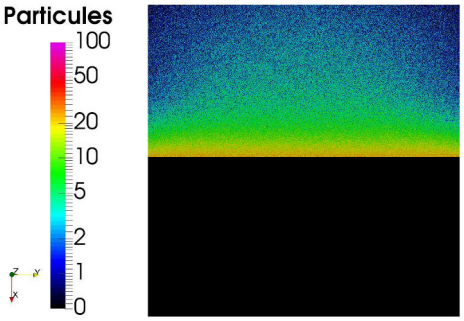
Notre premier cas test est un cas dit homogène. Ce cas test est présenté en figure 2.3. L'intérêt de ce cas étant la parfaite homogénéité du domaine. Les interactions particules matière suivent donc les mêmes lois de probabilité peu importe la localisation de la particule. Ici, le problème d'équilibre de charge est la résultante du seul fait que les particules soient toutes insérées sur le même bord du domaine (en haut). Nous avons choisi comme taille de maillage en espace carré de 1000 x 1000 mailles.



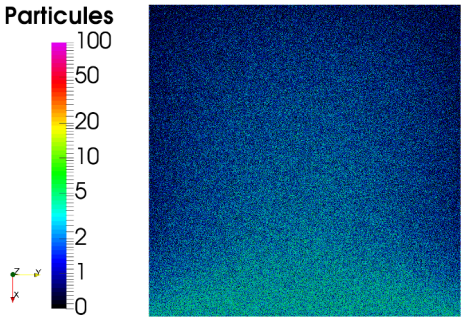
(a) Domaine physique dont la composition est homogène.



(b) Distribution des particules (i=20).



(c) Distribution des particules (i=480).



(d) Distribution des particules (i=1000).

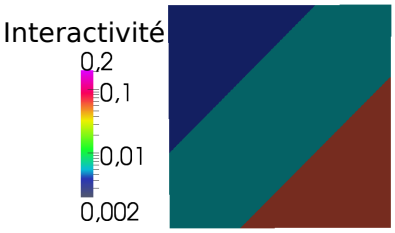
FIGURE 2.3 – Cas test homogène par insertion latérale.

Notre deuxième cas test est, en revanche, un cas hétérogène. Ce cas test est présenté en figure 2.4. L'intérêt de ce cas est que le domaine est divisé en trois régions aux propriétés d'interactions particules matière très différentes. Ici, le problème d'équilibre de charge est à la fois la résultante du fait qu'il n'y ait qu'une seule maille source, placée à la frontière entre deux régions aux propriétés d'interactions différentes, et des propriétés d'interactions du domaine. Ainsi, les particules partent du centre puis se dispersent avec la même probabilité dans toutes les directions mais cette fois leurs histoires sont très différentes. En particulier, les particules rencontrant la zone à forte interactivité ont tendance à être repoussées ou absorbées. Nous avons choisi un maillage carré de 1000 x 1000 mailles.

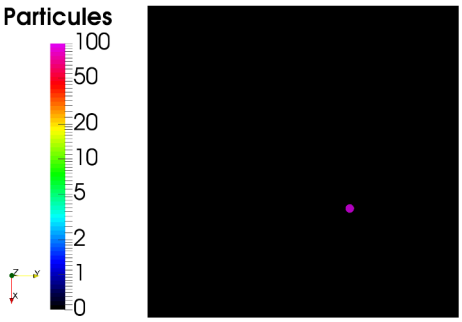
Enfin notre dernier cas test, dit le ≪dos d'âne≫, est présenté en figure 2.5. Les particules sont insérées sur le bord gauche du domaine. Ce cas est intéressant car les particules vont rencontrer un ≪obstacle≫, à savoir une zone à forte interactivité. Similairement au deuxième cas test, les particules rencontrant la zone à forte interactivité ont tendance à être repoussées ou absorbées. Nous avons choisi un maillage rectangulaire de 800 mailles en hauteur et 1200 mailles en longueur.

2.1.7 Conclusion

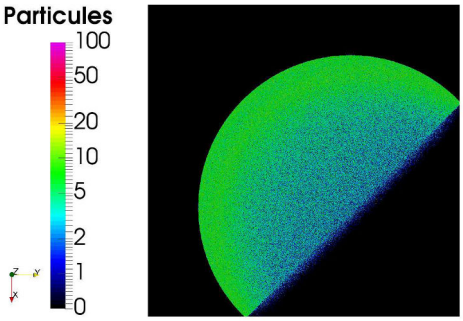
Le transport de particules Monte-Carlo permet de résoudre des problèmes physiques complexes. Le processus Monte-Carlo consiste à étudier les histoires des particules que nous injectons



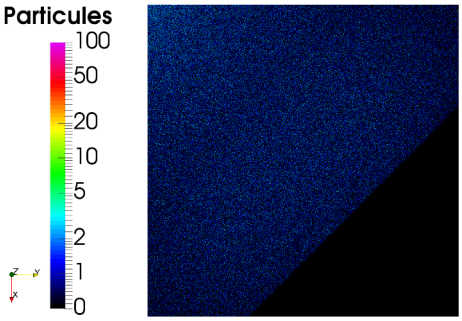
(a) Domaine physique dont la composition est hétérogène avec trois zones d'interactivité différentes.



(b) Distribution des particules (i=20).



(c) Distribution des particules (i=480).

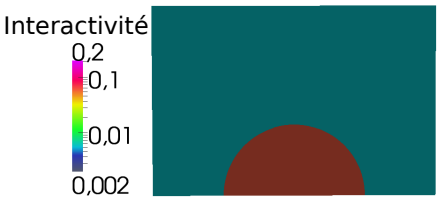


(d) Distribution des particules (i=1000).

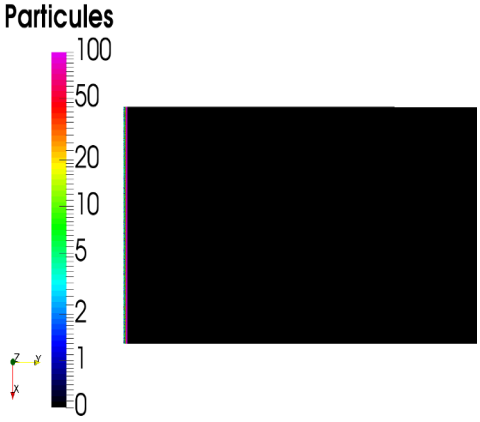
FIGURE 2.4 – Cas test hétérogène par insertion centralisée avec un domaine en trois parties.

dans un système physique. Ces particules subissent des évènements se présentant sous diverses formes. La simulation du transport de particules Monte-Carlo nécessite beaucoup d'éléments de simulation pour être précise. En effet, la finesse du maillage permet de représenter fidèlement les propriétés du domaine physique que nous souhaitons étudier. De plus, le nombre de particules définit la précision statistique de la méthode.

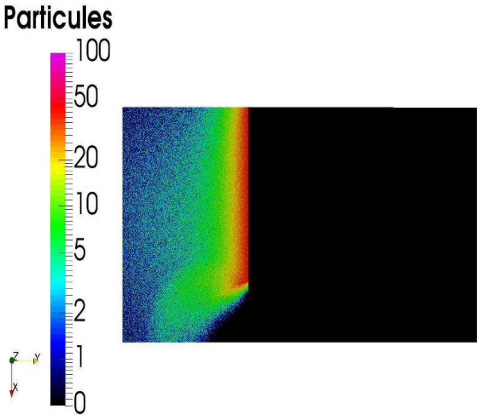
Dans le but d'obtenir une simulation précise, nous avons besoin d'un grand nombre de particules et de mailles. Pour l'application que nous avons utilisée, le stockage mémoire d'une particule nécessite environ 100 octets de mémoire et le stockage mémoire d'une maille requiert environ 200 octets. À titre d'exemple, le stockage mémoire d'un milliard de particules et de cent millions de mailles nécessiterait environ 120 Go de mémoire. Un grand nombre de particules et de mailles requiert donc des capacités de stockage mémoire conséquentes afin de pouvoir stocker en mémoire l'ensemble des données de simulation. Étant donné que le traitement d'une particule pour une itération, pour l'application utilisée, prend environ 1 μs, il faudrait un millier de secondes de calcul pour traiter une seule itération. L'augmentation du nombre de ressources de calcul doit permettre de réduire le temps de restitution de la simulation. Nous avons donc besoin de plateformes de calcul haute performance.



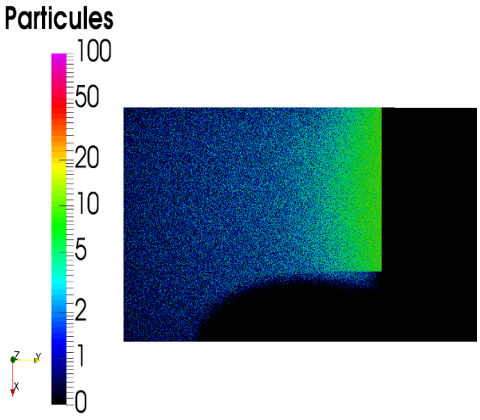
(a) Domaine physique dont la composition est hétérogène avec une zone en forme de ≪dos d'âne≫ de forte interactivité.



(b) Distribution des particules (i=20).



(c) Distribution des particules (i=420).



(d) Distribution des particules (i=900).

FIGURE 2.5 – Cas test hétérogène par insertion latérale avec un domaine possédant une zone en forme de ≪dos d'âne≫ de forte interactivité.

2.2 Plateformes de calcul haute performance

Pour répondre aux besoins de puissance de calcul et de stockage mémoire toujours plus importants, les architectures dites massivement parallèles sont devenues incontournables. Nous allons commencer par présenter ces architectures.

2.2.1 Ressources de calcul sur supercalculateurs

Pour exploiter un très grand nombre de cœurs de calcul, les architectures massivement parallèles multi-nœuds se sont développées. Chaque nœud dispose de sa propre mémoire (quelques dizaines de gigaoctets) et de ses propres cœurs de calcul (8, 16, 32, etc.). Nous nous focalisons

sur les architectures homogènes, par conséquent les nœuds de calcul ont la même puissance de calcul (mêmes processeurs) et la même capacité mémoire. Ces nœuds sont reliés entre eux par un réseau d'interconnexion. Ce réseau permet de faire communiquer les nœuds de calcul quand les informations contenues dans leur mémoires respectives doivent être échangées. La latence et la bande passante réseau constituent une problématique de performance pour de telles architectures. Le développement de réseaux de type InfiniBand [4] ont permis de répondre en partie à ces problématiques en offrant une faible latence (environ 1 microseconde) et un haut débit (plusieurs dizaines de Gbits/s). Les différentes topologies réseaux (*Fat Tree*, tore, etc.) ont également joué un rôle déterminant dans l'amélioration de l'exploitation d'architectures massivement parallèles.

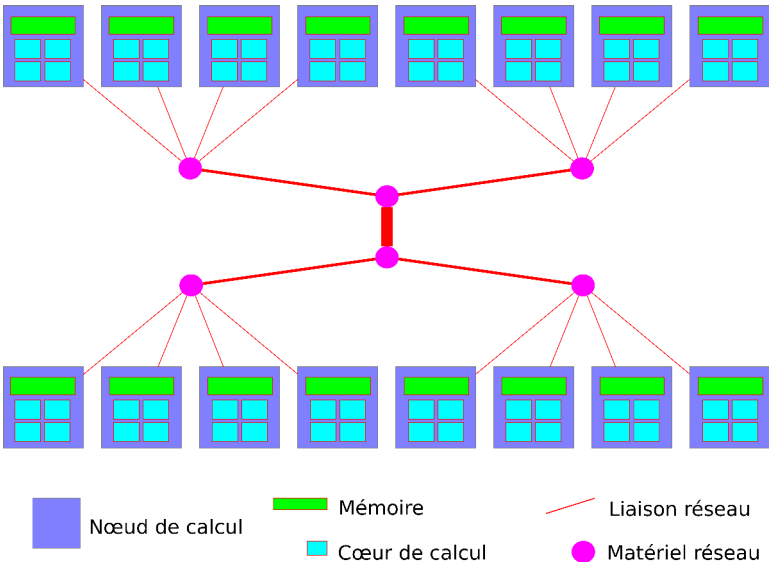


FIGURE 2.6 – Représentation d'une architecture massivement parallèle de type *Fat tree* comportant 64 cœurs de calcul.

La figure 2.6 présente un exemple d'architecture massivement parallèle de type *Fat Tree*. Nous avons dans cet exemple 4 cœurs par nœud de calcul et 16 nœuds de calcul. Le réseau d'interconnexion est également représenté. Notons que la bande passante d'une liaison réseau varie selon son emplacement dans la topologie (typiquement, la bande passante est la plus faible au niveau des nœuds de calcul). L'augmentation de la bande passante d'une liaison réseau permet de réduire les problèmes de contention.

L'exécution parallèle sur de telles architectures consiste à ce que chaque cœur de calcul traite une partie du problème. Lorsque quelques cœurs de calcul n'ont pas de travail à effectuer, ces architectures sont sous-exploitées. La parallélisation sur de telles architectures doit s'exprimer à la fois en inter-nœuds et en intra-nœud. La répartition de la charge de calcul et des données sur ces architectures se situe donc à plusieurs niveaux. Pour chacun de ces niveaux, des interfaces de programmation parallèle ont été proposées. Nous allons présenter les interfaces les plus couramment utilisées.

2.2.2 Interfaces de programmation parallèle

Dans le domaine du calcul haute performance, le parallélisme a grande échelle a fait émerger des problématiques complexes. En effet, la gestion de ressources de calcul et de mémoires toujours plus complexes et hiérarchisées nécessite la mise en place de mécanismes à des granularités

différentes. Ces mécanismes, tels que les communications en contexte à mémoire distribuée ou l'ordonnancement de *threads* en contexte à mémoire partagée, sont difficiles à implanter. Dans le but de faciliter la parallélisation des applications, des interfaces de programmation parallèle ont été créées. Ces interfaces ont pour objectif d'abstraire les couches logicielles les plus basses, c'est-à-dire les plus proches du matériel, afin de faciliter l'écriture et d'améliorer la portabilité des applications parallèles.

Nous distinguons des familles différentes d'interfaces de programmation parallèle. En effet, selon la granularité du parallélisme considéré, les interfaces de programmation parallèle ne sont pas les mêmes. Nous allons présenter dans cette sous-section les interfaces de programmation qui ont été utilisées pendant cette thèse. *MPI (Message Passing Interface)* [5] est une interface de programmation parallèle particulièrement bien adaptée pour le parallélisme inter-nœuds. En effet, cette interface fournit l'ensemble des fonctions de communication nécessaires aux transferts des données entre les nœuds. De plus, les performances des supports exécutifs sont élevées.

Le déploiement de processus *MPI* sur tous les cœurs de calcul implique, pour certaines applications, une augmentation significative du temps passé dans les communications *MPI* lorsque le nombre de cœurs utilisés augmente. Nous observons également une augmentation non négligeable de la consommation mémoire lorsque certaines données doivent être répliquées comme par exemple les mailles fantômes. En intra-nœud, la mémoire est partagée par tous les cœurs de calcul. Dans la mesure où l'ensemble des données est accessible par tous les cœurs du nœud, les mécanismes de communication ne sont pas nécessaires. *OpenMP* [6] est une interface de programmation parallèle destinée à la parallélisation en intra-nœud. Cette interface propose des techniques de parallélisation optimisées pour les cas irréguliers. Nous avons donc choisi *OpenMP* [6] pour la parallélisation en intra-nœud.

MPI (Message Passing Interface) : parallélisation en contexte à mémoire distribuée

MPI est une norme définissant une interface de bibliothèque. Les fonctionnalités définies par cette norme permettent de déployer des processus *MPI* sur des nœuds de calcul et de les faire communiquer par passage de messages. Nous retrouvons plusieurs types de communication. Par exemple, les communications peuvent être point-à-point ou collectives, bloquantes ou non-bloquantes, etc. Nous allons présenter les fonctions de communication qui nous ont été utiles dans l'élaboration de nos algorithmes et dans l'implantation, plus généralement, de notre application. Parmi les fonctions de communication collective, nous pouvons citer les fonctions *MPI_Reduce* et *MPI_Allreduce*. La fonction *MPI_Reduce* permet de réaliser une opération sur des données distribuées (addition, multiplication, recherche du minimum, recherche du maximum, etc.) dont le résultat est retourné à un processus *MPI* désigné. La fonction *MPI_Allreduce* permet de réaliser un *MPI_Reduce* en renvoyant le résultat à l'ensemble des processus *MPI*. Dans le cas du transport de particules Monte-Carlo, cette fonction est utile pour mettre à jour le nombre total de particules présentes dans le domaine. La fonction *MPI_Gather* permet de regrouper des données distribuées par un procédé de concaténation dans la mémoire d'un processus *MPI* désigné. La fonction *MPI_Allgather* permet de transmettre la concaténation des données à l'ensemble des processus *MPI*.

Le passage de message d'un processus P0 (un processus *MPI*) à un processus P1 (un autre processus) peut être réalisé en utilisant des méthodes différentes. La norme *MPI* définit un ensemble de fonctions de communication. Lorsqu'une communication est bloquante, les ressources spécifiées par l'utilisateur lors de l'appel à la fonction de communication peuvent être réutilisées immédiatement. En revanche, lorsqu'une communication est non-bloquante, les res-

sources spécifiées par l'utilisateur lors de l'appel à la fonction de communication ne peuvent pas être réutilisées immédiatement.

MPI_Send est la fonction de communication correspondant à un envoi de message bloquant. À la sortie de la fonction, le message à envoyer sera pris en compte par le support exécutif, le tampon mémoire utilisé pour la communication pourra être réutilisé. Cela ne signifie pas que le message sera nécessairement reçu par l'autre processus. Cela dépend du protocole réseau utilisé. Le transfert du message sera totalement transparent pour l'utilisateur. *MPI_Recv* est la fonction de communication correspondant à une réception de message. À la sortie d'un *MPI_Recv* le message est disponible dans le tampon mémoire de réception.

MPI_Isend et *MPI_Irecv* désignent les versions non-bloquantes des fonctions de passages de messages. Ces fonctions permettent de transmettre une requête au support exécutif *MPI* concernant le transfert d'un message. En revanche, le transfert n'est pas nécessairement traité immédiatement par le support exécutif. La fonction *MPI_Wait* permet de compléter une communication non-bloquante. La fonction *MPI_Test* permet de tester la complétion locale d'une communication. L'appel périodique à cette fonction permet de faire progresser et de compléter la communication.

La figure 2.7 présente un avantage à utiliser les fonction *MPI_Isend* et *MPI_Test* en comparaison d'une utilisation d'un *MPI_Send*. Nous prenons comme exemple le transfert de particules d'un processus P0 vers des processus P1 et P2 avant une itération de transport de particules Monte-Carlo. Le processus P1 est désynchronisé, il est en retard par rapport à P0 et P2. Ce cas peut se produire en cas de déséquilibre de charge. Les particules transmises par P0 seront traitées au cours de l'itération. Dans cet exemple, le processus P0 ne reçoit pas de particule. P0 fait appel aux fonctions de communication en commençant par celles dont le destinataire est P1. L'utilisation de *MPI_Test* permet à P0 de compléter la communication concernant P2 en attendant que P1 entre dans l'itération courante. À la fin de l'itération, nous arrivons sur une synchronisation. En effet, nous devons mettre à jour le nombre total de particules présentes dans le domaine. Une communication de type *MPI_Allreduce* est alors invoquée.

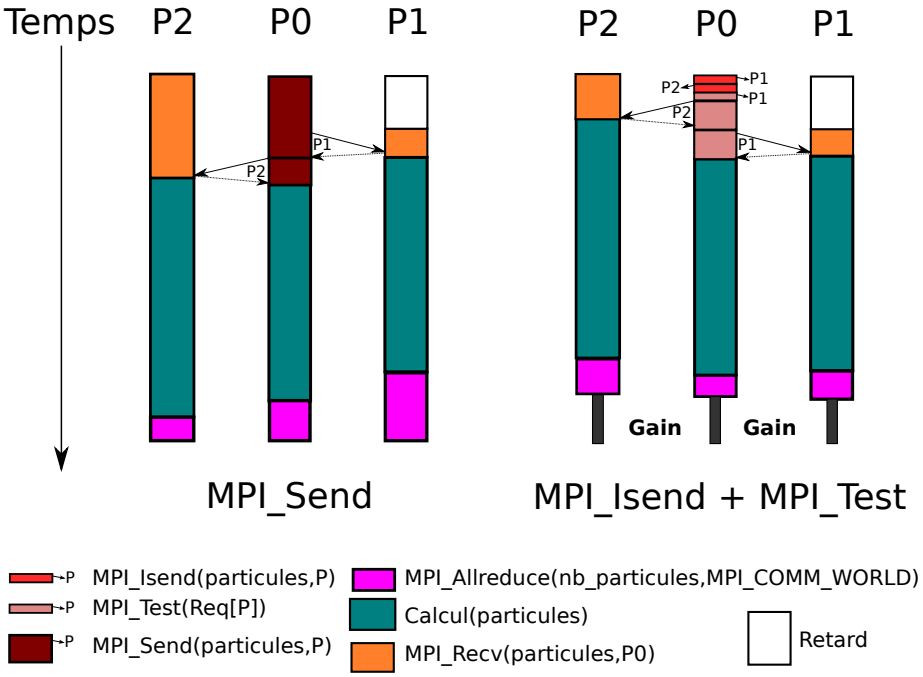


FIGURE 2.7 – Cas d'utilisation de *MPI_Isend* + *MPI_Test* par rapport à *MPI_Send*.

La norme *MPI* définit également un ensemble de fonctions de communication dites unidirectionnelles [7][8]. Ce type de communication permet d'écrire ou de lire des données distantes sans nécessiter l'intervention explicite dans la communication du processus possédant la donnée. Par exemple, les fonctions *MPI_Put* et *MPI_Get* définissent, respectivement, une écriture distante et une lecture distante. La fonction *MPI_Accumulate* permet d'incrémenter une donnée distante d'une valeur passée en paramètre. La norme *MPI*-3 définit la fonction *MPI_compare_and_swap* permettant de réaliser une opération de *compare and swap* sur une donnée distante. Nous rappelons que le *compare and swap* consiste à modifier une valeur dans le cas où une condition sur la valeur est satisfaite. Le contenu de la valeur est ensuite retourné. Par exemple, *MPI_compare_and_swap(x,y,z,...)* permet de mettre une valeur distante à x seulement si cette valeur vaut y. La fonction met à jour z avec le contenu de la valeur avant l'appel à la fonction. Dans le cas où $z = y$ à la sortie de la fonction, cela signifie que la valeur a été mise à jour. La figure 2.8 présente un exemple d'utilisation de *MPI_compare_and_swap*. Cet exemple illustre le principe du jeton distant. Dans cet exemple, les processus tentent de prendre le jeton en écrivant leur rang. Le jeton est pris seulement si, au moment de l'appel à *MPI_compare_and_swap*, la valeur du jeton est égale au rang du processus où est stocké en mémoire le jeton (zéro dans notre cas).

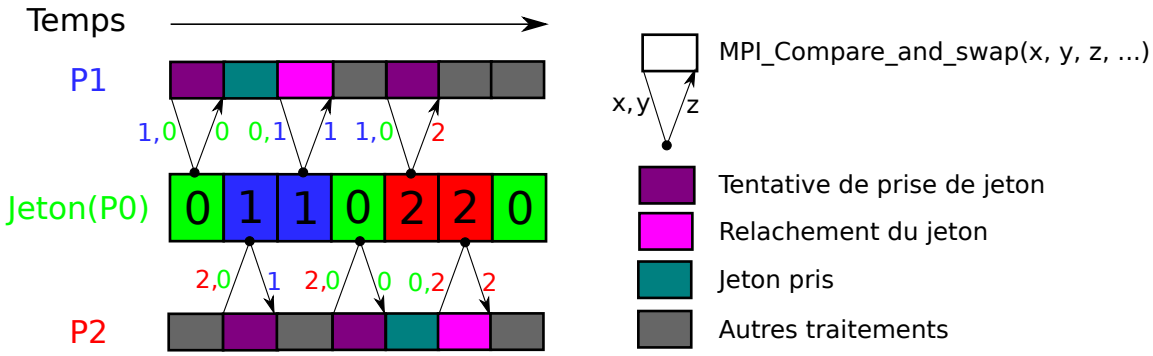


FIGURE 2.8 – Exemple du jeton avec *MPI_compare_and_swap*.

Les fonctions de communication unidirectionnelle accèdent à des données distantes. Ces accès s'opèrent à l'aide de *MPI_Window*. Les objets de type *MPI_Window* permettent de gérer les accès distants et locaux à certaines données. À chaque *MPI_Window* correspond des données dont il permet l'accès. Ces objets permettent de gérer la concurrence des accès distants.

OpenMP : parallélisation en contexte à mémoire partagée

La norme *OpenMP* permet de définir le déploiement de *threads*, fils d'execution se partageant l'espace d'adressage d'un même processus, sur les cœurs de calcul. Nous pouvons répartir des instructions sur des données à l'aide de directives de compilations. La directive *pragma omp parallel* permet de démarrer une région parallèle *OpenMP* dont le nombre de *threads* souhaité par l'utilisateur peut être donné en argument. Les *threads* ont accès à un segment de mémoire partagée et à un segment de mémoire privée. L'utilisateur peut choisir quelles données seront dans quels segments. Les accès aux données appartenant au segment de mémoire doivent être protégés. Il incombe à l'utilisateur d'assurer cette protection.

Certaines directives permettent de répartir les accès aux données, et les calculs qui y sont appliqués, lorsque ceux-ci sont réalisés à l'aide d'une boucle. La directive *pragma omp for* permet de répartir les itérations d'une boucle entre les *threads*. *OpenMP* définit plusieurs techniques d'ordonnancement appliquées à la répartition des itérations de boucles. L'ordonnancement statique

présente l'avantage de ne pas nécessiter de mécanisme particulier mais ne tient pas compte de la charge opérationnelle de chaque itération. L'ordonnancement dynamique permet d'attribuer les itérations de boucle dynamiquement, en choisissant des *threads* disponibles. Cette technique permet de répartir correctement la charge de calcul d'une boucle lorsque les itérations de boucle ont des charges opérationnelles très hétérogènes. Par exemple, considérons la parallélisation du traitement des particules. Les particules sont localisées sur des mailles. chaque particule met à jour la maille sur laquelle elle se trouve (contribution). La parallélisation du traitement des particules consiste alors à répartir les mailles entre les *threads*. Les particules localisées sur la même maille sont traitées par le même *thread*. Les particules ne sont pas nécessairement équitablement réparties entre les mailles. La figure 2.9 illustre alors l'intérêt d'un ordonnancement dynamique pour le traitement des particules du transport Monte-Carlo.

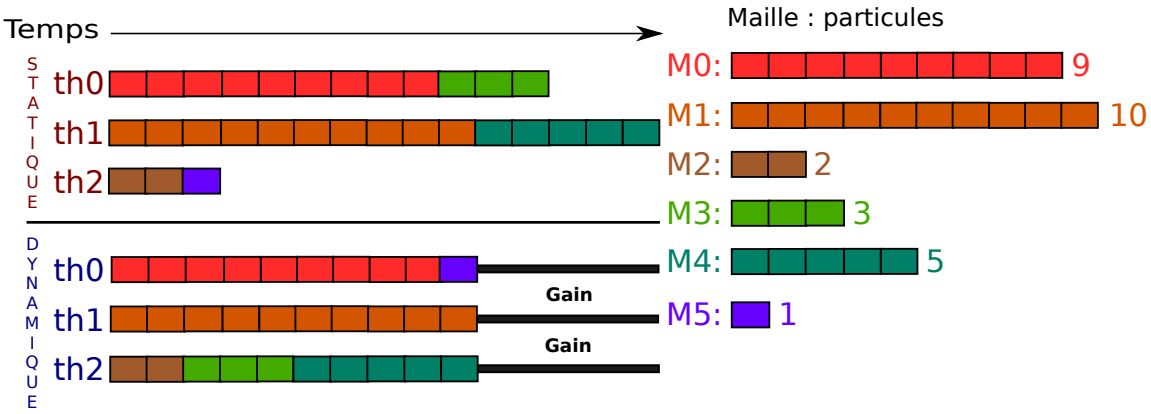


FIGURE 2.9 – Comparaison entre les ordonnancements statique et dynamique.

La répartition de la charge de calcul nécessite parfois un grain un peu plus fin. Les calculs ne peuvent pas toujours être exprimés à l'aide d'une boucle. Dans ces cas là, l'utilisation du *pragma omp for* ne suffit plus à répartir la charge de calcul. *OpenMP* a intégré depuis la norme *OpenMP-4* [9] les tâches *OpenMP* (*OMP_Tasks*). Il s'agit de regrouper des instructions entre elles formant des tâches. Ainsi, les *threads* se répartissent les tâches à exécuter.

L'utilisation d'interfaces de programmation parallèle performantes permet d'optimiser l'exploitation des architectures parallèles. Une bonne exploitation des ressources permet de réduire le temps de restitution de la simulation et de limiter la consommation mémoire au sein d'un nœud de calcul. Le passage à l'échelle d'une application désigne sa capacité à exploiter efficacement des ressources de plus en plus nombreuses. Pour estimer cette capacité, nous faisons appel à des métriques que nous allons maintenant développer.

2.2.3 Métriques d'extensibilité parallèle

La parallélisation d'une application sur une architecture parallèle est obtenue par l'utilisation d'une méthode de parallélisation. Le choix de la méthode soulève plusieurs problématiques [10].

- Cette méthode est-elle adaptée à l'architecture parallèle cible ?
- Cette méthode peut-elle s'adapter à des configurations différentes ? (taille du problème, nombre de cœurs)
- Comment comparer cette méthode aux autres méthodes pour le même problème ?

Nous devons alors choisir des métriques adaptées à caractériser les performances parallèles d'une méthode. Nous allons présenter des métriques selon deux approches différentes.

Métriques d'extensibilité forte

Pour mesurer la capacité d'une application à exploiter efficacement des architectures massivement parallèles, une approche consiste à étudier le comportement de l'application en fonction du nombre de ressources de calcul pour un problème de taille fixe. Nous pouvons étudier la variation du temps d'exécution et la variation de la consommation mémoire de cette application. Cette étude permet, par exemple, de déterminer le temps de restitution minimal pour un problème d'une taille donnée sur une machine spécifique. Nous parlons alors d'une étude en extensibilité forte. Une extensibilité forte optimale est obtenue lorsque le temps de restitution et la consommation mémoire par ressource diminue proportionnellement au nombre de ressources. Soit P^1 désignant un résultat obtenu lors d'une exécution séquentielle (temps de restitution ou consommation mémoire). L'efficacité parallèle avec N ressources en extensibilité forte se mesure alors au moyen de l'équation 2.1.

$$E^N = \frac{\frac{P^1}{N}}{P^N} \, . \tag{2.1}$$

La figure 2.10 présente une extensibilité forte optimale. Dans cet exemple, la taille du problème est constante. Nous remarquons que le temps de restitution et la consommation mémoire par ressource diminuent proportionnellement à l'augmentation du nombre de ressources (noté N).

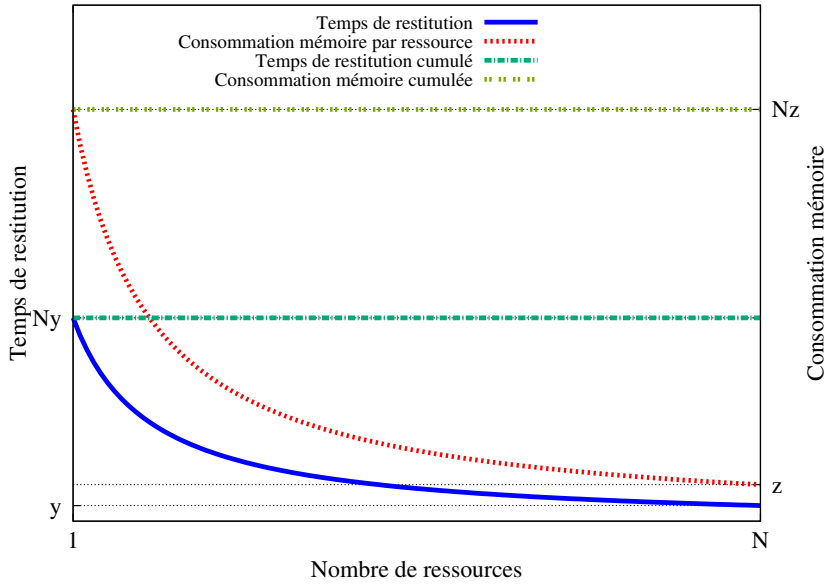


FIGURE 2.10 – Extensibilité forte optimale.

Métriques d'extensibilité faible

L'utilisation d'architectures parallèles ne permet pas seulement de réduire le temps de restitution pour un problème donné mais aussi de traiter des problèmes de plus grandes tailles. Pour mesurer la capacité d'une application à exploiter efficacement des architectures massivement parallèles, une autre approche consiste à étudier le comportement de l'application en fonction du nombre de ressources de calcul pour un problème de taille variable. La taille du problème est proportionnelle au nombre de ressources. Nous pouvons étudier la variation du temps d'exécution et la variation de la consommation mémoire de cette application en fonction du nombre de ressources.

Nous parlons alors d'une étude en extensibilité faible. Une extensibilité faible optimale est obtenue lorsque le temps de restitution et la consommation mémoire par ressource demeurent constants. Soit P^1 désignant un résultat obtenu lors d'une exécution séquentielle (temps de restitution ou consommation mémoire). L'efficacité parallèle avec N ressources en extensibilité faible se mesure alors au moyen de l'équation 2.2.

$$E^N = \frac{P^1}{P^N} \, . \tag{2.2}$$

La figure 2.11 présente une extensibilité faible optimale. Dans cet exemple, la taille du problème est proportionnel au nombre de ressources. Nous constatons que le temps de restitution et la consommation mémoire par ressource sont constants alors que le nombre de ressources et la taille du problème augmentent.

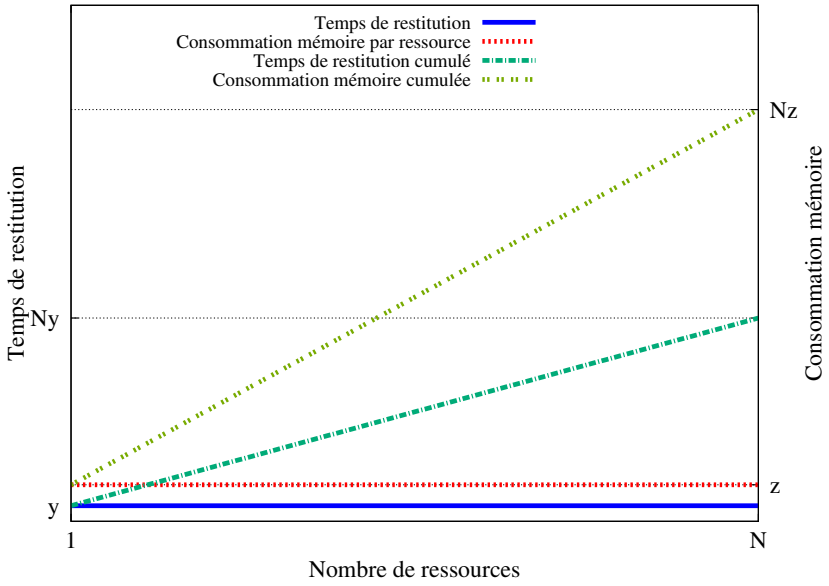


FIGURE 2.11 – Extensibilité faible optimale.

L'exploitation efficace des ressources de calcul nécessite une bonne répartition de la charge de calcul et des données entre les ressources. Une méthode de parallélisation désigne une approche permettant de distribuer la charge de calcul et les données entre les ressources. Dans le domaine du calcul haute performance, certaines de ces méthodes sont très utilisées. Ces méthodes feront l'objet d'une présentation dans la prochaine sous-section.

2.2.4 Présentation des méthodes classiques de parallélisation

Dans le domaine du calcul haute performance, l'exploitation optimale des ressources de calcul mises à dispositions constitue un objectif prioritaire. Des méthodes de parallélisation sont utilisées pour atteindre cet objectif. Nous allons décrire dans cette section trois méthodes parmi les plus connues et les plus utilisées en simulation numérique : décomposition de domaine [11][12], réplication de domaine [12] et hybride [12][13][14].

Méthode de décomposition de domaine

La méthode de décomposition de domaine [11] [12] a pour principale propriété de créer un ensemble de sous-domaines. L'approche consiste à décomposer le problème initial en sous-problèmes de plus petites tailles. Soit D le domaine et n le nombre de sous-domaines, la décomposition en sous-domaines répond à la condition de l'équation 2.3.

$$\bigcup_{i=0}^{n} d_i = D \, . \tag{2.3}$$

Chaque sous-domaine ainsi créé est assigné à une ressource de calcul donnée. Chaque ressource de calcul se voit affecter les données de simulation correspondant à son sous-domaine. Les ressources de calcul pourront alors réaliser directement les calculs appliqués à leurs données de simulation. Les sous-domaines ne sont pas nécessairement disjoints entre eux. Nous parlons alors de recouvrement. Le recouvrement est utile notamment dans le cas où un calcul sur une donnée, appartenant à un sous-domaine, nécessite des données voisines appartenant à un autre sous-domaine. La duplication de ces données évite alors une communication.

Afin d'obtenir une bonne distribution des données de simulation sur l'ensemble des ressources de calcul, les sous-domaines doivent être, le plus possible, de la même taille. Étant donné N le nombre de ressources, dans le cas d'une décomposition de domaine, nous avons $n = N$. La figure 2.12 présente un exemple de décomposition de domaine en deux sous-domaines, avec ou sans recouvrement.

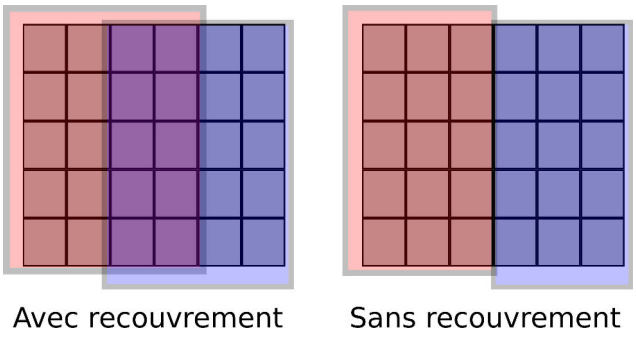


FIGURE 2.12 – Décompositions de domaine.

Appliquée au transport de particules Monte-Carlo, cette méthode consiste à répartir les mailles entre les ressources. Ensuite, chaque ressource traite les particules présentes sur ses mailles.

Méthode de réplication de domaine

La méthode de réplication de domaine [12] s'oppose à la méthode de décomposition de domaine. L'intégralité du domaine est répliqué sur l'ensemble des ressources de calcul tel que défini par l'équation 2.4. La réplication de domaine peut être appliquée lorsque le parallélisme d'une application n'est pas inhérent au domaine à proprement parler. Dans le cas du transport de particules Monte-Carlo, le parallélisme s'exprime davantage sur le traitement des particules que sur le traitement du domaine. De plus, l'usage de cette méthode est particulièrement intéressant dans la mesure où n'importe quelle ressource peut traiter n'importe quelle particule. Chaque ressource de calcul peut donc étudier l'histoire de n'importe quelle particule du début à la fin de la simulation. Aucune communication de particule n'est alors nécessaire. La méthode de réplication de domaine a

d'ailleurs été implantée dans une application Monte-Carlo nommée Mercury [15]. À l'initialisation de la simulation du transport de particules Monte-Carlo, une distribution équitable des particules entre les ressources est réalisée.

$$\forall i \in [0, N[,\ d_i = D\ . \tag{2.4}$$

Méthode hybride décomposition-réplication de domaine

La méthode hybride décomposition-réplication de domaine [12][13][14] consiste à réaliser une décomposition de domaine avec un nombre de sous-domaines strictement inférieur au nombre de ressources. Chaque sous-domaine est alors partagé par plusieurs ressources de calcul. Nous introduisons la notion de degré de réplication. Au lieu de créer N sous-domaines, nous créons $\frac{N}{R}, 1 < R < N$ sous-domaines où R est appelé degré de réplication.

Si nous considérons d_i le sous-domaine de la ième ressource de calcul, il y a $R - 1$ autres ressources de calcul possédant le même sous-domaine (voir l'équation 2.5). Dans le cas du transport de particules Monte-Carlo, les ressources, assignées au même sous-domaine, se répartissent équitablement l'ensemble des particules appartenant à ce sous-domaine. La méthode hybride permet de faire un compromis entre une décomposition stricte du domaine et une réplication intégrale du domaine.

$$\forall d_i \in D, \exists J = \{j_1, j_2, ..., j_R\}, \forall j \in J,\ d_i = d_j\ . \tag{2.5}$$

La valeur de R peut être fixée selon deux méthodes. Soit K une constante. La première consiste à exprimer R comme une constante indépendante de N. Dans ce cas $R = K$. La seconde consiste à exprimer R en fonction de N. Dans ce cas, $R = \frac{N}{K}$. Dans le cas du transport de particules Monte-Carlo, le paramétrage de cette méthode a fait l'objet de travaux de recherche [13][14]. Par ailleurs, des implantations de la méthode hybride ont été proposées dans des applications de transport de particules Monte-Carlo [15][16].

Conclusion

Les méthodes de parallélisation consistent à répartir les données et la charge de calcul entre les ressources. La décomposition de domaine consiste à répartir les mailles de manière équilibrée. La décomposition du domaine doit également minimiser les interfaces entre les sous-domaines. En effet, le volume de communication est, en général, proportionnel à la somme des tailles des interfaces entre les sous-domaines. La problématique consiste alors à décomposer équitablement le domaine tout en minimisant les interfaces. Des techniques de partitionnement permettent de résoudre cette problématique. Nous étudierons ces techniques dans la prochaine section.

2.3 Techniques de partitionnement

Répartir équitablement la charge de calcul entre les sous-domaines tout en minimisant les interfaces peut, dans certaines configurations, être une problématique complexe. Nous trouvons dans la littérature des outils de partitionnement de maillage optimisé permettant de résoudre cette problématique. Nous pouvons citer *MeTiS* [17], *Scotch* [18] et *Pampa* [19]. Ces outils s'appuient sur des heuristiques de résolution. La plupart des ces heuristiques repose sur une modélisation du problème initial sous forme de graphe [20]. Nous proposons de présenter les graphes [21] dans la sous-section 2.3.1. Nous présenterons ensuite les techniques de partitionnement de graphe en

sous-section 2.3.2. Le partitionnement multi-critères sera étudié en sous-section 2.3.3. Enfin, en sous-section 2.3.4 nous introduirons les techniques de repartitionnement.

2.3.1 Le graphe : un modèle abstrait

Les graphes sont des modèles abstraits très utilisés dans de nombreux domaines. Leur généricité permet de les adapter à des problématiques très différentes. Les graphes présentent l'avantage d'être à la fois concis et exhaustifs. En effet, un graphe est un ensemble de sommets et d'arêtes. Un lien peut être orientée, c'est-à-dire unidirectionnel, nous parlons alors d'arc. Les sommets comme les arêtes permettent de représenter des objets et leurs interactions. Par exemple, en informatique, un graphe peut représenter le flot d'instructions d'une application où chaque sommet représente une instruction et chaque arête représente une dépendance. Il est communément appelé graphe de flot de contrôle.

Les concepts de sommets et d'arêtes, orientées ou non, permettent de modéliser des données de simulation, des instructions, des processus, etc. Cette modélisation peut être affinée à l'aide des pondérations des sommets et des arêtes. En effet, la pondération des graphes permet de différencier deux sommets entre eux ou deux arêtes entre elles, en leurs attribuant des poids différents. À titre d'exemple, nous pourrions différencier des sommets du graphe de flot de contrôle en fonction du coût en temps d'une instruction.

Dans le contexte du calcul haute performance, les graphes peuvent être utilisés pour modéliser des charges de calcul et des données. La répartition de ces charges de calcul et de ces données peut être obtenue à l'aide d'algorithmes de partitionnement de graphe.

2.3.2 Algorithmes de partitionnement de graphe

Le partitionnement de graphe est un problème NP-Difficile [22]. En effet, le nombre de partitionnements possibles pour un graphe donné est exponentiel. Le partitionnement peut être soumis à des contraintes initiales. En calcul haute performance, la difficulté d'un partitionnement réside dans deux problématiques antagonistes. La première consiste à trouver un partitionnement le plus équilibré possible. La seconde consiste à minimiser le temps de restitution du partitionnement. Parmi les contraintes initiales, nous avons, la plupart du temps, l'équilibre en nombre de sommets par partie.

Plusieurs heuristiques de partitionnement peuvent être trouvées dans la littérature. Globalement, les algorithmes proposés cherchent à minimiser la coupe (la taille des interfaces) dans le respect des contraintes initiales. Nous pouvons par exemple citer la métaheuristique multi-niveau [23][24][25]. Des approches parallèles de cette heuristique ont été proposées [26] pour réduire le temps de restitution.

Les heuristiques sont nombreuses et quelques unes ont été implémentées dans des outils de partitionnement de graphe. Parmi ces outils, nous pouvons citer *MeTiS* [17] et *Scotch* [18]. Ces deux outils sont dotés d'une version parallèle, respectivement *ParMeTiS* [27] et *PT-Scotch* [28], l'objectif étant de réduire le temps de partitionnement.

La figure 2.13 donne un exemple de partitionnement équilibré en nombre de sommets obtenu avec *MeTiS* [17]. Le partitionnement est une problématique complexe. Lorsque les contraintes initiales sont multiples, les algorithmes de partitionnements doivent considérer plusieurs critères. Nous parlons de partitionnement multi-critères.

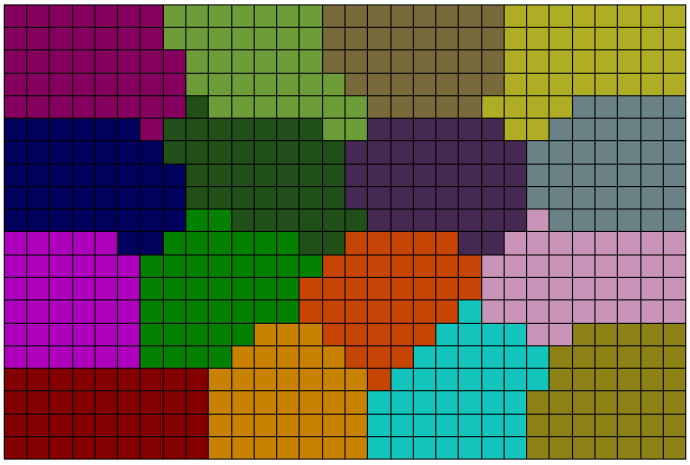


FIGURE 2.13 – Exemple d'un partitionnement en 16 parties, équilibré en nombre de sommets.

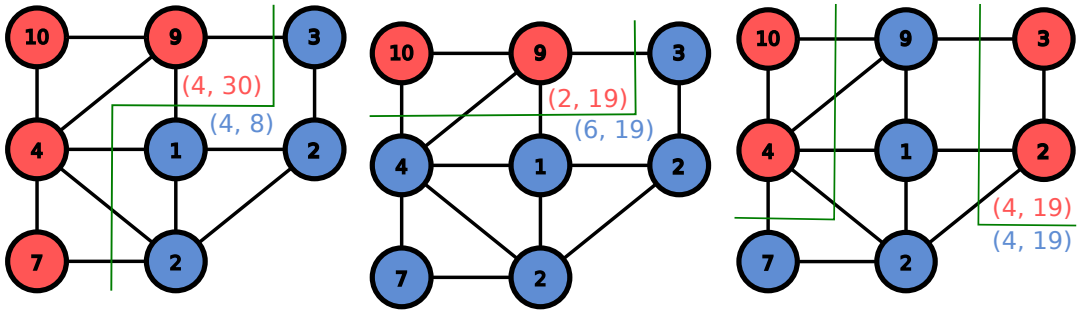
2.3.3 Partitionnement multi-critères

Une partition est dite valide lorsque celle-ci respecte la totalité des critères. Une partition de très bonne qualité sur pratiquement tous les critères mais ne respectant pas la contrainte fixée pour un seul et unique critère ne sera pas jugée valide. En revanche, une partition de qualité passable sur tous les critères mais respectant toutes les contraintes sera jugée valide. Les contraintes fixées dépendent du problème à résoudre. Un non-respect de l'une de ces contraintes peut avoir des conséquences dramatiques. Par exemple, si l'une de ces contraintes concerne la consommation mémoire d'une application, une violation de la contrainte pourrait entraîner l'interruption de l'application.

La figure 2.14 présente des exemples de partitionnements d'un graphe pondéré aux propriétés très différentes. Le poids de chaque sommet est inscrit au centre du sommet. Dans notre cas, les arêtes ont toutes le même poids. Nous souhaitons obtenir une partition équilibrée à la fois en nombre de sommets et en poids. La partition de la sous-figure 2.14a est équilibrée en nombre de sommets (4 par partie). Nous observons une coupe relativement peu élevée avec 5 arêtes coupées. En revanche, l'équilibre en poids n'est pas du tout respecté, avec 30 d'un côté et seulement 8 de l'autre. La sous-figure 2.14b présente un partitionnement équilibré en poids (19 par partie). Nous observons une coupe encore meilleure avec seulement 4 arêtes coupées. Cependant, l'équilibre en nombre de sommets n'est absolument pas respecté puisque nous avons 2 sommets d'un côté et 6 de l'autre. Enfin, l'équilibre en nombre de sommets et en poids est obtenu avec le partitionnement de la sous-figure 2.14c. En revanche, la coupe est plus élevée que les partitionnements précédents avec 8 arêtes coupées, soit 2 fois plus que le partitionnement précédent. De plus, l'un des sous-graphes n'est pas connexe. Cet exemple illustre la difficulté de partitionner selon plusieurs critères tout en minimisant la coupe.

Le partitionnement multi-critères nécessite d'autres heuristiques que le partitionnement mono-critère [29][30][31][32]. Par ailleurs, le partitionnement multi-critères n'est pas supporté par tous les outils de partitionnement. En effet, contrairement à *MeTiS* [17], *Scotch* [18] ne gère pas le partitionnement multi-critères.

Lorsque la répartition de la charge de calcul évolue au cours du temps, la validité de la partition peut être remise en cause et le calcul d'une nouvelle partition peut alors être envisagé. Nous



(a) Exemple d'un partitionnement équilibré en nombre de sommets.

(b) Exemple d'un partitionnement équilibré en poids.

(c) Exemple d'un partitionnement équilibré en nombre de sommets et en poids.

FIGURE 2.14 – Exemples de partitionnements.

proposons donc maintenant d'introduire les algorithmes de repartitionnement.

2.3.4 Algorithmes de repartitionnement

Le calcul d'une nouvelle partition pose une problématique de coût de migration dans le cas où le partitionnement influe sur la répartition des données de simulation. En effet, lorsqu'un sommet change de partie, les données de simulation correspondant à ce sommet doivent être transférées. Cette problématique de coût de migration de donnée a été étudiée dans la littérature [33][34][35][36][37]. Cette contrainte supplémentaire complique sérieusement le partitionnement d'un graphe lorsque ce dernier est très déséquilibré. Nous devons dans certains cas faire un compromis entre un partitionnement de moins bonne qualité et un coût de migration plus important.

Les outils de partitionnement de graphe *ParMeTiS* [27] et *Scotch* [18] proposent une méthode de repartitionnement en prenant comme paramètre la partition précédente.

2.4 Méthodologie d'expérimentations

Dans ce document plusieurs méthodes de parallélisation seront présentées. Nous avons déjà présentés les méthodes classiques de parallélisation en sous-section 2.2.4. Chaque méthode présentée fera l'objet d'une étude expérimentale (chapitres 3, 4 et 6). Nous avons défini l'extensibilité d'une application (sous-section 2.2.3). Nous avons vu qu'elle pouvait se présenter sous deux formes nommées extensibilité forte et extensibilité faible. Nous évaluerons donc les méthodes avec les deux approches.

Cette étude sera réalisée sur les trois cas test présentés en sous-section 2.1.6 et sera décomposée en trois parties. Nous commencerons par observer la répartition de la charge de calcul et la répartition des données de simulation. Les performances en extensibilité forte seront ensuite observées puis analysées. Enfin, nous observerons et analyserons les performances en extensibilité faible.

2.4.1 Plateforme expérimentale et paradigmes de programmation parallèle

Nous souhaitons tester des méthodes sur des cas test suffisamment complexes pour être pertinents. En l'occurrence, nous sélectionnons des cas test de plusieurs centaines de millions de parti-

cules et quelques millions de mailles. L'évaluation des méthodes de parallélisation pour le transport nécessite alors des ressources de calcul et de mémoire conséquentes.

Nous avons choisi un supercalculateur de très grande envergure pour effectuer nos expérimentations. En effet, nous avons choisi le supercalculateur Curie [1]. Curie offre une performance crête de 2 Pflops. Curie est constitué de 360 nœuds de calcul larges, 16 nœuds hybrides et 5040 nœuds standards. Nous avons sélectionné les nœuds standards composés de processeurs octo-cœurs Intel® Sandy Bridge EP (E5-2680) cadencés à 2.7 GHz et disposant de 64 Go de mémoire. Les nœuds de calcul sont interconnectés par un réseau InfiniBand QDR selon une topologie dite Fat Tree (voir figure 2.6, sous-section 2.2.1).

Toutes les implantations des méthodes que nous étudierons utilisent un paradigme de programmation parallèle *MPI + OpenMP*. Ce paradigme de programmation est celui utilisé dans le code de simulation duquel est extrait notre prototype [12]. Pour toutes les méthodes, nous avons déployé un processus *MPI* par nœud de calcul. Dans notre cas, une ressource de calcul désigne donc un nœud de calcul. Nous avons ensuite utilisé l'ensemble des cœurs de calcul de chaque nœud en déployant des *threads OpenMP* sur chacun des cœurs. Le calcul est alors partagé par tous les *threads* en utilisant un ordonnancement dynamique.

Nous avons utilisé, pour toutes les méthodes nécessitant des communications de mailles et de particules, une technique de communication avec recouvrement des communications par du calcul, telle que définie par l'algorithme 3. Ainsi, pendant que le *thread* maître termine les opérations de communication (ligne 6), les autres *threads* commencent la phase de calcul. Ils traitent, durant cette première étape, les mailles et les particules du processus *MPI* qui ne sont pas concernées par une communication (ligne 13). Le *thread* maître les rejoint lorsque toutes les communications sont achevées. À partir du moment où la première étape de calcul est terminée et que le *thread* maître a terminé les communications, tous les *threads* terminent la phase de calcul en traitant les mailles et les particules reçues en début d'itération (ligne 14). Ensuite, le *thread* maître initialise les communications non-bloquantes qui seront terminées au cours de l'itération suivante, et ainsi de suite. L'implantation *MPI* utilisée est Bullxmpi en version 1.2.8.4 et l'implantation *OpenMP* utilisée est Intel *OpenMP* en version 5.0.20140401. Le compilateur utilisé est le compilateur Intel (ICC) en version 14.0.3.20140422.

2.4.2 Étude de régulation de charge et de donnée

Les performances en extensibilité d'une méthode sont en partie la conséquence de la répartition de la charge de calcul et des données induite par la méthode. Par exemple, une mauvaise répartition des particules impliquera un ralentissement sur le temps d'exécution et une surconsommation mémoire importante au niveau des ressources les plus chargées. Nous commencerons donc notre étude par analyser le comportement de chaque méthode concernant la répartition de la charge de calcul et des données. Nous étudierons le déséquilibre maximal, en nombre de particules, observé à chaque itération. Le déséquilibre maximal est obtenu en calculant le rapport entre le nombre de particules de la ressource la plus chargée en particules et le nombre moyen de particules par ressource. Soit P^{max} le nombre de particules de la ressource la plus chargée et P^{moyen} le nombre de particules moyen. L'équation 2.6 donne la formule utilisée pour le calcul du déséquilibre de charge.

$$\Delta_{charges} = \frac{P^{max}}{P^{moyen}} \, . \tag{2.6}$$

La mesure de déséquilibre de charge permettra de déterminer si une méthode parvient à répartir

1. `http://www-hpc.cea.fr/fr/complexe/tgcc-curie.htm`

Algorithme 3 : Algorithme de recouvrement des communications par le calcul en contexte *MPI + OpenMP*.

```
   /* L'ensemble des itérations de la simulation                          */
1  tant que !fin_application faire
       /* Début de l'itération                                           */
2      OMP_debut_region_parallele()
3      si id_thread = 0 alors
           /* le thread maître termine les communications en cours       */
4          tant que nombre_requetes_non_finies > 0 faire
5              pour chaque Requete_non_finie faire
6                  MPI_Test( Requete, Maillage, Particules )
7                  si Status = finie alors
8                      nombre_requetes_non_finies ← nombre_requetes_non_finies − 1
9                  fin
10             fin
11         fin
12     fin
       /* Phase de Transport première partie                             */
13     Monte_carlo_transport_OMP( Maillage, Particules )
       /* Phase de Transport seconde partie - les données issues des MPI_Irecv
        */
14     Monte_carlo_transport_OMP_reste( Maillage, Particules )
       /* Fin de l'itération                                             */
15     OMP_fin_region_parallele()
       /* Envoi de mailles et particules                                 */
16     MPI_Isends( Requetes, Maillage, Particules )
       /* Réception de mailles et particules                             */
17     MPI_Irecvs( Requetes, Maillage, Particules )
18 fin
```

convenablement la charge de calcul. Une surcharge est pénalisante en termes de temps de restitution mais, à priori, elle ne compromet pas l'exécution complète de la simulation. En revanche, une surconsommation mémoire excessive peut entraîner l'interruption immédiate de la simulation. Nous étudierons donc la surconsommation mémoire impliquée par chaque méthode en termes de particules et de mailles. La surconsommation mémoire est obtenue en calculant le rapport entre le nombre d'éléments de la ressource ayant le plus de donnée stockées en mémoire et le nombre d'éléments par ressource dans le cas optimal. Soit Mem^{max} le nombre d'éléments de simulation de la ressource ayant la plus grande consommation mémoire et $Mem^{optimal}$ le nombre d'éléments de simulation par ressource dans le cas d'une répartition optimale. L'équation 2.7 donne la formule utilisée pour le calcul de la surconsommation mémoire.

$$\Delta_{mem} = \frac{Mem^{max}}{Mem^{optimal}} \, . \tag{2.7}$$

Nous réaliserons l'étude de régulation de charge et de donnée pour 32 et 256 nœuds de calcul pour l'ensemble des cas test (voir sous-section 2.1.6).

2.4.3 Étude d'extensibilité forte

L'étude en extensibilité forte consiste à observer les performances d'une méthode de parallélisation lorsque la charge totale de calcul reste constante alors que le nombre de ressources de calcul utilisées augmente. Nous commencerons par une mesure en temps en faisant varier le nombre de nœuds de calcul de 1 à 256. Nous utiliserons donc jusqu'à 4096 cœurs de calcul.

Le nombre de particules doit être suffisamment élevé pour que le résultat de la simulation soit pertinent. Toutefois, pour réaliser une étude en extensibilité forte, nous devons choisir un nombre de particules permettant le stockage en mémoire des particules sur un seul nœud de calcul pour permettre une exécution séquentielle. Nous devons également choisir un nombre de particules permettant un temps de restitution sur un seul nœud de calcul de seulement quelques heures pour maintenir la durée d'expérimentation à un niveau raisonnable. En tenant compte de ces contraintes, nous avons fixé le nombre de particules à 128 millions.

Afin d'analyser plus finement les résultats d'extensibilité observés nous proposerons un profilage de l'application. L'application sera décomposée en sections de programme définies comme suit :

- Traitement des particules + communication ;
- Traitement des mailles ;
- Partitionnement(s) ;
- Algorithme de régulation (hors partitionnement).

La première section correspond au temps passé dans l'algorithme 3. Nous avons choisi de regrouper les communications et le traitement des particules pour prendre en compte le recouvrement. La deuxième section correspond à la mise à jour des listes de particules sur les mailles ainsi que l'opération de réduction sur les contributions aux mailles. Les deux premières sections correspondent aux éléments de simulation et sont donc très impactés par la méthode de parallélisation utilisée. En effet, une mauvaise parallélisation donnera un traitement des particules et/ou un traitement des mailles plus long. La troisième section permet d'évaluer le coût en temps du partitionnement lorsque la méthode de parallélisation recourt à une technique de partitionnement. Le coût de partitionnement exprimé regroupe l'ensemble de tous les mécanismes nécessaire à la réalisation du partitionnement, communications comprises. Enfin, la quatrième et dernière section permet de déterminer le coût d'une méthode quand celle-ci nécessite des traitements spécifiques.

Chaque processus *MPI* mesure le temps passé dans chaque section de programme. Lorsqu'un point de synchronisation, impliquant la synchronisation de l'ensemble des processus *MPI*, est atteint, nous prenons pour référence les temps obtenus par le processus *MPI* arrivant en dernier au point de synchronisation. Nous réaliserons ce profilage avec 256 nœuds de calcul. Nous déterminerons alors les causes des éventuels problèmes de performance constatés.

2.4.4 Étude d'extensibilité faible

Cette étude consiste à observer les performances d'une méthode de parallélisation lorsque le nombre de particules par ressource de calcul est fixé. Nous ferons varier le nombre de ressources de calcul de 1 à 256, soit jusqu'à 4096 cœurs de calcul. Nous avons fixé le nombre de particules par ressource à 4 millions. Nous ferons donc varier le nombre de particules de 4 millions à 1024 millions.

La précision statistique de l'application dépend essentiellement du nombre de particules. L'augmentation du nombre de mailles augmente considérablement le temps de restitution de l'application. Dans notre cas, cela impact également la granularité d'une itération. Étant donné que le temps de restitution de l'application n'augmente pas proportionnellement au nombre de mailles et que l'augmentation du nombre de particules seul permet une amélioration de la précision statistique, nous avons choisi un nombre de mailles constant.

Similairement à l'étude en extensibilité forte, nous commencerons par une mesure en temps en faisant varier le nombre de nœuds de calcul puis nous procéderons à un profilage de l'application suivant la même méthode que le profilage en extensibilité forte.

2.5 Conclusion

Le transport de particules Monte-Carlo nécessite des ressources de calcul importantes pour répondre au besoin de précision et de rapidité de restitution. Pour exploiter des ressources de calcul complexes, des méthodes de parallélisation sont employées dans le domaine de la simulation numérique. Parmi ces méthodes, nous avons présenté la décomposition de domaine, la réplication de domaine et la méthode hybride (sous-section 2.2.4). Certaines de ces méthodes nécessitent de répartir des données entre les ressources.

L'extensibilité des applications de transport de particules Monte-Carlo dépend, en grande partie, de la capacité de la méthode de parallélisation utilisée à répartir équitablement les données de simulation et la charge de calcul. Pour évaluer l'extensibilité des méthodes de parallélisation que nous avons présentées dans ce chapitre, nous proposerons, dans un premier temps, une analyse théorique de ces méthodes. Cette analyse aura pour objectif de définir les éventuelles limitations de ces méthodes. Dans un second temps, nous évaluerons expérimentalement ces méthodes dans les configurations développées dans ce chapitre (en section 2.4).

Chapitre 3

Analyse des méthodes classiques de parallélisation appliquées au transport Monte-Carlo

Dans le domaine du calcul haute performance, et en particulier dans le domaine de la simulation numérique, il existe plusieurs méthodes de parallélisation. Ces méthodes ont pour objectifs de distribuer la charge de calcul et les données sur un très grand nombre de cœurs. Ces distributions sont soumises à un ensemble de contraintes que nous avons détaillées au chapitre 2. Ces distributions doivent être les plus équilibrées possible afin de garantir que :

– l'ensemble des données puisse être placé en mémoire ;
– le calcul sur ces données puisse être le plus efficace possible.

Les objectifs ont été présentés dans l'introduction de cette thèse. Pour rappel, nous souhaitons limiter l'empreinte mémoire afin de garantir que les données tiennent en mémoire et nous souhaitons minimiser le déséquilibre de charge en termes de particules, notamment. Enfin, les coûts des mécanismes de régulation doivent être réduits afin de pouvoir être compensés par une meilleure exécution parallèle.

Nous allons expliquer ici en quoi des méthodes de parallélisation dites ≪classiques≫ ne peuvent être utilisées directement pour atteindre les objectifs fixés. Nous avons sélectionné trois méthodes. Il s'agit de la méthode de décomposition de domaine, de la méthode de réplication de domaine et enfin de la méthode hybride de décomposition-réplication de domaine. Nous allons étudier ces méthodes de manière théorique et sur un ensemble de cas test. Nous commencerons par présenter les métriques de l'analyse théorique en section 3.1. La section 3.2 présentera l'analyse théorique des méthodes classiques. Enfin, l'étude expérimentale sera détaillée en section 3.3.

3.1 Métriques d'analyse théorique

Notre étude a pour objectif d'exprimer les forces et faiblesses des approches utilisées par rapport aux objectifs que nous nous sommes fixés. L'étude théorique consiste à vérifier l'atteinte des objectifs par ces méthodes. Nous évaluerons dans le même temps le gain et le coût de la méthode. L'étude va fournir pour chaque méthode des métriques indiquant la qualité de la répartition de la charge de calcul et des volumes de donnée.

3.1.1 Définitions

Soit P le nombre de particules total du système, M le nombre de mailles associées au domaine et N le nombre de ressources de calcul. Dans notre cas, une ressource de calcul désigne un nœud de calcul. Soit ψ_m la charge de calcul en temps associée à une maille et ψ_p la charge de calcul en temps associée à une particule. Les charges de calcul relatives aux données de simulation de la $i$ème ressource de calcul peuvent s'écrire $\Psi_i = M_i\psi_m + P_i\psi_p$ où M_i est le nombre de mailles et P_i le nombre de particules de la $i$ème ressource. Soit t_m la taille mémoire associée à une maille, t_p la taille mémoire associée à une particule et $T_i = M_i t_m + P_i t_p$ la consommation mémoire de la $i$ème ressource.

Soit $\overline{M}^N$ et $\overline{P}^N$ respectivement le nombre de mailles maximal et le nombre de particules maximal par ressource dans une configuration donnée. $\overline{M}^N$ est le plus petit majorant de l'ensemble des $M_i, \forall i \in [0, N[$. $\overline{P}^N$ est le plus petit majorant de l'ensemble des $P_i, \forall i \in [0, N[$. Nous avons alors les conditions de l'équation 3.1. Pour un cas donné, nous obtenons alors $\overline{\Psi}^N$, la charge de calcul maximale définie par l'équation 3.2. De la même manière, nous obtenons $\overline{T}^N$, la consommation mémoire maximale définie par l'équation 3.3.

$$\begin{cases} \forall i \in [0, N[\ , & M_i \leq \overline{M}^N\ ; \\ \forall i \in [0, N[\ , & P_i \leq \overline{P}^N\ . \end{cases} \tag{3.1}$$

$$\overline{\Psi}^N = \overline{M}^N \psi_m + \overline{P}^N \psi_p\ . \tag{3.2}$$

$$\overline{T}^N = \overline{M}^N t_m + \overline{P}^N t_p\ . \tag{3.3}$$

Nous proposons de caractériser l'extensibilité de chaque méthode. Pour cela, nous allons étudier la répartition de la charge de calcul et des données lorsque nous augmentons le nombre de ressources à l'infini. Pour un cas donné, nous définissons la charge de calcul maximale avec une infinité de ressources $\overline{\Psi}^\infty$ par l'équation 3.4. De même, nous définissons la consommation mémoire maximale avec une infinité de ressources $\overline{T}^\infty$ par l'équation 3.5.

$$\overline{\Psi}^\infty = \lim_{N \to +\infty} \overline{\Psi}^N\ . \tag{3.4}$$

$$\overline{T}^\infty = \lim_{N \to +\infty} \overline{T}^N\ . \tag{3.5}$$

Soit Γ la capacité mémoire d'une ressource de calcul donnée ($\Gamma > 0$). Les données de simulation tiennent en mémoire avec N ressources de calcul lorsque l'équation 3.6 est vérifiée. Dans le cas où la fonction définissant la répartition des données en fonction du nombre de ressources est monotone, l'équation 3.7 est définie.

$$\overline{T}^N \leq \Gamma\ . \tag{3.6}$$

$$\overline{T}^\infty \leq \Gamma \implies \exists N, \overline{T}^N \leq \Gamma\ . \tag{3.7}$$

Soit $\hat{\overline{M}}^N$ et $\hat{\overline{P}}^N$, respectivement, le nombre maximal de mailles et le nombre maximal de particules, dans la pire configuration. De même $\check{\overline{M}}^N$ et $\check{\overline{P}}^N$ représentent, respectivement, le nombre maximal de mailles et le nombre maximal de particules, dans la meilleure configuration. Nous

définissons alors $\overset{\hat{}}{\overline{\Psi}}^N = \overset{\hat{}}{\overline{M}}^N \psi_m + \overset{\hat{}}{\overline{P}}^N \psi_p$ et $\overset{\hat{}}{\overline{T}}^N = \overset{\hat{}}{\overline{M}}^N t_m + \overset{\hat{}}{\overline{P}}^N t_p$, respectivement, la charge de calcul maximale et la consommation mémoire maximale, dans la pire configuration. Nous définissons $\overset{\check{}}{\overline{\Psi}}^N = \overset{\check{}}{\overline{M}}^N \psi_m + \overset{\check{}}{\overline{P}}^N \psi_p$ et $\overset{\check{}}{\overline{T}}^N = \overset{\check{}}{\overline{M}}^N t_m + \overset{\check{}}{\overline{P}}^N t_p$, respectivement, la charge de calcul maximale et la consommation mémoire maximale, dans la meilleure configuration. Quand cela sera pertinent, nous distinguerons la meilleure et la pire configuration dans l'analyse d'une méthode.

Les mécanismes mise en œuvre pour optimiser la répartition de la charge de calcul et pour limiter la consommation mémoire, tout en n'altérant pas la précision de la simulation, peuvent se présenter sous plusieurs formes. Soit f_c la fonction croissante définie sur $\mathbb{N}$, la fonction de coût des échanges de donnée. Étant donné x le volume de donnée à échanger et sachant que le coût en communication est fonction du volume de donnée, f_c est résumée par l'équation 3.8. Cette fonction ne s'annule pas en zéro. En effet, le coût minimal est noté L. Nous considérons alors la latence réseau comme étant le coût minimal d'une communication. Le coût total, sur l'ensemble de la simulation, de ces mécanismes est noté C^N. Enfin, nous définissons C^∞ le coût des mécanismes de régulation de charge et de donnée avec une infinité de ressources (équation 3.9).

$$\begin{cases} \lim\limits_{x \to +\infty} & f_c(x) = +\infty \ ; \\ \lim\limits_{x \to 0} & f_c(x) = L \ ; \\ \forall x \in \mathbb{N} & f_c'(x) \geq 0 \ . \end{cases} \tag{3.8}$$

$$C^\infty = \lim_{N \to +\infty} C^N \ . \tag{3.9}$$

Une méthode de parallélisation aura pour objectif de minimiser la consommation mémoire maximale, la charge de calcul maximale et le coût des mécanismes de régulation de charge et de donnée. Dans le but d'évaluer les méthodes de parallélisation, nous proposons de les comparer à un référentiel commun représentant une méthode optimale.

3.1.2 Méthode optimale

Supposons l'existence d'une méthode de parallélisation optimale telle que la charge et les données soient toujours parfaitement réparties. Soit $\underset{min}{\overline{\Psi}}^N$ la charge de calcul et $\underset{min}{\overline{T}}^N$ la consommation mémoire de la ressource la plus chargée, en utilisant cette méthode. $\underset{min}{\overline{\Psi}}^N$ est définie par l'équation 3.10 et $\underset{min}{\overline{T}}^N$ est définie par l'équation 3.11.

$$\underset{min}{\overline{\Psi}}^N = \frac{M\psi_m + P\psi_p}{N} \ . \tag{3.10}$$

$$\underset{min}{\overline{T}}^N = \frac{Mt_m + Pt_p}{N} \ . \tag{3.11}$$

Supposons que l'on dispose d'un nombre de ressources de calcul infini. La charge de calcul maximale et la consommation mémoire maximale, avec une méthode optimale, sont définies respectivement par les équations 3.12 et 3.13. Notons que, dans le cas idéal, augmenter le nombre de ressources permet de réduire la charge maximale de calcul et la consommation mémoire de manière proportionnelle. Avec une méthode optimale, l'équation 3.7 est satisfaite. En effet, $\underset{min}{T}^\infty = 0$ alors que $\Gamma > 0$.

$$\overline{\Psi}^{\infty}_{min} = \lim_{N\to+\infty} \overline{\Psi}^{N}_{min} = \lim_{N\to+\infty} \frac{M\psi_m + P\psi_p}{N} = 0 \,. \tag{3.12}$$

$$\overline{T}^{\infty}_{min} = \lim_{N\to+\infty} \overline{T}^{N}_{min} = \lim_{N\to+\infty} \frac{Mt_m + Pt_p}{N} = 0 \,. \tag{3.13}$$

Une méthode optimale est une méthode ayant une extensibilité parfaite. Cela nécessite que le coût de régulation de charge et de donnée soit négligeable par rapport au temps de calcul. L'objectif étant d'exprimer un référentiel, nous définissons $\underset{min}{C^N} = 0$. Nous obtenons donc un coût nul avec une infinité de ressources ($\underset{min}{C^{\infty}} = 0$).

3.2 Analyse théorique

Les méthodes classiques de parallélisation ont l'avantage d'être rapides à mettre en place. D'ailleurs, ces méthodes s'appliquent très bien pour un large panel d'applications. Toutefois, nous allons voir dans cette section qu'elles ne permettent pas d'atteindre les objectifs visés pour les applications de transport de particules Monte-Carlo.

3.2.1 Méthode de décomposition de domaine

La méthode de décomposition de domaine est une méthode de parallélisation très utilisée en simulation numérique [11] [12]. La décomposition de domaine consiste à répartir équitablement les mailles entre les ressources. Elle présente l'avantage de garantir une répartition parfaite des données de simulation relative au domaine de simulation. Nous allons voir dans cette sous-section en quoi cette méthode ne garantit pas la réalisation des objectifs pour toutes les configurations.

L'utilisation d'une méthode de décomposition de domaine pour le transport Monte-Carlo nécessite des communications lorsque des particules changent de sous-domaine. Nous définissons alors $\underset{DD}{C^N}$, le coût des mécanismes de régulation de charge et de donnée de la méthode de décomposition de domaine. Nous supposons que le volume de communication maximal entre les ressources est proportionnel au nombre de particules de chacune de ces ressources. Le coût maximal est donc proportionnel à $\overline{P}^N$. Soit $q, 0 < q \leq 1$ le ratio de particules concernées par une communication d'une ressource à une autre. $\underset{DD}{C^N}$ est alors défini par l'équation 3.14. Notons que nous avons choisi de ne pas considérer le coût de la décomposition de domaine proprement dite.

$$\underset{DD}{C^N} = f_c(\overline{P}^N t_p q) \,. \tag{3.14}$$

La répartition des particules n'est pas nécessairement uniforme sur le domaine. Par conséquent, une répartition équitable des particules entre les sous-domaines n'est pas garantie. Concrètement, le comportement de cette méthode dépend largement de la configuration. Nous nous plaçons donc dans la meilleure configuration et dans la pire configuration.

Meilleure configuration

Dans la meilleure configuration, les particules sont équitablement distribuées entre tous les sous-domaines. Nous avons alors $\underset{DD}{\overline{\check{M}}^N} = \frac{M}{N}$ et $\underset{DD}{\overline{\check{P}}^N} = \frac{P}{N}$. Nous obtenons donc $\underset{DD}{\overline{\check{\Psi}}^N}$ défini par l'équation 3.15 et $\underset{DD}{\overline{\check{T}}^N}$ défini par l'équation 3.16.

De plus, les particules ne migrent pas de sous-domaine, il n'y a donc pas de communication de particules. Nous définissons donc $\check{C}_{DD}^{N}$ au moyen de l'équation 3.17.

$$\overline{\check{\Psi}}_{DD}^{N} = \frac{M}{N}\psi_m + \frac{P}{N}\psi_p \,. \tag{3.15}$$

$$\overline{\check{T}}_{DD}^{N} = \frac{M}{N}t_m + \frac{P}{N}t_p \,. \tag{3.16}$$

$$\check{C}_{DD}^{N} = \lim_{N\to+\infty} f_c(0) = L \,. \tag{3.17}$$

Avec une infinité de ressources, nous obtenons alors $\overline{\check{\Psi}}_{DD}^{\infty}$ défini par l'équation 3.18 et $\overline{\check{T}}_{DD}^{\infty}$ défini par l'équation 3.19. Dans la meilleure configuration, la répartition de la charge de calcul et des données de la méthode de décomposition de domaine est parfaite. L'équation 3.7 est alors vérifiée. Avec une infinité de ressources le coût des communications est minimisé (équation 3.20).

$$\overline{\check{\Psi}}_{DD}^{\infty} = \lim_{N\to+\infty} \overline{\check{\Psi}}_{DD}^{N} = 0 \,. \tag{3.18}$$

$$\overline{\check{T}}_{DD}^{\infty} = \lim_{N\to+\infty} \overline{\check{T}}_{DD}^{N} = 0 \,. \tag{3.19}$$

$$\check{C}_{DD}^{\infty} = \lim_{N\to+\infty} L = L \,. \tag{3.20}$$

Pire configuration

Lorsque toutes les particules sont toujours dans un seul sous-domaine et qu'elle changent de sous-domaine à chaque itération, nous sommes dans la pire configuration. Dans celle-ci, nous avons $\hat{\overline{M}}_{DD}^{N} = \frac{M}{N}$ et $\hat{\overline{P}}_{DD}^{N} = P$. Nous obtenons alors $\hat{\overline{\Psi}}_{DD}^{N}$ défini par l'équation 3.21 et $\hat{\overline{T}}_{DD}^{N}$ défini par l'équation 3.22.

$$\hat{\overline{\Psi}}_{DD}^{N} = \frac{M}{N}\psi_m + P\psi_p \,. \tag{3.21}$$

$$\hat{\overline{T}}_{DD}^{N} = \frac{M}{N}t_m + Pt_p \,. \tag{3.22}$$

Avec une infinité de ressources, nous obtenons alors $\hat{\overline{\Psi}}_{DD}^{\infty}$ défini par l'équation 3.23 et $\hat{\overline{T}}_{DD}^{\infty}$ défini par l'équation 3.24. Dans la pire configuration, la répartition des particules de la méthode de décomposition de domaine est très mauvaise. En effet, les calculs relatifs aux particules ainsi que le stockage mémoire de ces dernières n'impactnt qu'une seule ressource. Étant donné que la consommation mémoire diminue lorsque N augmente, la consommation mémoire minimale est obtenue avec une infinité de ressources. Avec une infinité de ressources, les données de simulation peuvent être stockées en mémoire si $Pt_p \leq \Gamma$. La méthode ne garantit donc pas, dans la pire configuration, que les données de simulation tiennent en mémoire lorsque $P > \dfrac{\Gamma}{t_p}$ et/ou lorsque $t_p > \dfrac{\Gamma}{P}$. Cette mauvaise répartition des particules constitue donc une importante limitation de la

méthode de décomposition de domaine. De plus, le coût en communication avec une infinité de ressources, $\hat{\underset{DD}{C}}^{\infty}$, est élevé (équation 3.25). En effet, il est proportionnel au nombre de particules.

$$\hat{\underset{DD}{\overline{\Psi}}}^{\infty} = \lim_{N \to +\infty} \hat{\underset{DD}{\overline{\Psi}}}^{N} = P\psi_p \; . \tag{3.23}$$

$$\hat{\underset{DD}{\overline{T}}}^{\infty} = \lim_{N \to +\infty} \hat{\underset{DD}{\overline{T}}}^{N} = Pt_p \; . \tag{3.24}$$

$$\hat{\underset{DD}{C}}^{\infty} = \lim_{N \to +\infty} f_c(Pt_p) = f_c(Pt_p) \; . \tag{3.25}$$

La méthode DD permet de répartir équitablement les mailles entre les ressources. En revanche, les déséquilibres potentiels en nombre de particules par ressource ne permettent pas à cette méthode d'obtenir une bonne extensibilité. Nous devons donc opter pour une méthode où les particules sont mieux réparties.

3.2.2 Méthode de réplication de domaine

La méthode de réplication de domaine est une méthode de parallélisation très utilisée pour les applications de type transport Monte-Carlo [12]. Nous allons voir dans cette section pourquoi cette méthode est-elle si souvent utilisée mais aussi pourquoi elle ne constitue pas une solution viable à la problématique d'équilibrage de charge et de donnée.

Cette méthode garantit un parfait équilibre de charge à l'initialisation. Néanmoins, des déséquilibres en termes de particules peuvent apparaître lorsque des particules sont absorbées et/ou émises en cours de simulation. Les populations de particules associées à chaque ressource de calcul sont totalement indépendantes. Si une ressource de calcul n'a plus de particules aucun mécanisme de régulation de charge n'est prévu. En théorie, dans la pire configuration nous pouvons nous retrouver dans la même situation que la décomposition de domaine. Le cas d'étude qui nous intéresse ici nécessite un nombre de particules significatif. Les particules sont uniformément distribuées entres les ressources et la répartition des graines est aléatoire. Nous supposons alors qu'il y a très peu de déséquilibre entre les ressources.

Soit $\underset{RD}{\varepsilon}, 0 < \underset{RD}{\varepsilon} < 1$ le facteur d'incertitude sur la répartition des particules. Les mailles ne sont pas du tout distribuées. Nous avons alors $\underset{RD}{\overline{M}}^{N} = M$ et $\underset{RD}{\overline{P}}^{N} = (1 + \underset{RD}{\varepsilon})\frac{P}{N}$. Nous obtenons alors $\underset{RD}{\overline{\Psi}}^{N}$ défini par l'équation 3.26 et $\underset{RD}{\overline{T}}^{N}$ défini par l'équation 3.27.

$$\underset{RD}{\overline{\Psi}}^{N} = M\psi_m + (1 + \underset{RD}{\varepsilon})\frac{P}{N}\psi_p \; . \tag{3.26}$$

$$\underset{RD}{\overline{T}}^{N} = Mt_m + (1 + \underset{RD}{\varepsilon})\frac{P}{N}t_p \; . \tag{3.27}$$

Cette méthode ne met aucun mécanisme de communication de particules en jeu. En revanche, certaines données de simulation, calculées sur les mailles, doivent être échangées, à la fin de la simulation, entre toutes les ressources de calcul. En effet, les données répliquées doivent être sommées pour obtenir un résultat cohérent. Il s'agit d'un mécanisme de communication collective appelé réduction. Nous notons t'_m la taille mémoire associée à la donnée à échanger de chaque maille. Nous définissons $f_r(x, y)$, définie sur $\mathbb{N}^2$, la fonction de coût d'une réduction. Cette fonction est croissante sur $\mathbb{N}^2$ (voir l'équation 3.28). Étant donné que la réduction est dimensionnée en fonction de la taille du maillage et du nombre de ressources, $\underset{RD}{C}$ est alors obtenu par l'équation 3.29.

$$\begin{cases} \lim\limits_{x\to+\infty} & f_r(x,y) = +\infty \ ; \\ \lim\limits_{y\to+\infty} & f_r(x,y) = +\infty \ ; \\ \lim\limits_{x\to 0} & f_r(x,y) = 0 \ ; \\ \lim\limits_{y\to 0} & f_r(x,y) = L \ . \end{cases} \tag{3.28}$$

$$\underset{RD}{C}^N = f_r(N, Mt'_m) \ . \tag{3.29}$$

Avec une infinité de ressources, nous obtenons alors $\overline{\Psi}^{\infty}_{RD}$ et $\overline{T}^{\infty}_{RD}$ définis, respectivement, par les équations 3.30 et 3.31. La non-décomposition du domaine amène un coût de traitement du maillage non réductible et une surconsommation mémoire importante. En effet, le nombre de mailles par ressource est constant. La surcharge liée aux mailles est un handicap vis à vis de l'extensibilité de l'application. La surconsommation mémoire peut avoir des conséquences bien plus dramatiques dans la mesure où elle contraint très largement le degré de raffinement du maillage. En effet, pour toutes les valeurs de Γ, il existe un couple (M, t_m) pour lequel la capacité mémoire d'un nœud de calcul n'est pas suffisante. Le fait que le nombre de mailles par ressource soit non réductible constitue donc une sérieuse limitation de la méthode de réplication de domaine.

$$\overline{\Psi}^{\infty}_{RD} = \lim_{N\to+\infty} \overline{\Psi}^{N}_{RD} = M\psi_m \ . \tag{3.30}$$

$$\overline{T}^{\infty}_{RD} = \lim_{N\to+\infty} Mt_m + (1 + \underset{RD}{\varepsilon})\frac{P}{N}t_p = Mt_m \ . \tag{3.31}$$

L'équation 3.32 donne le coût en communication avec une infinité de ressources. Le coût en communication augmente proportionnellement au nombre de ressources. La méthode de réplication de domaine est donc également limitée par le coût très important de la réduction.

$$\underset{RD}{C}' = \lim_{N\to+\infty} f_r(N, Mt'_m) = +\infty \ . \tag{3.32}$$

La méthode RD permet de résoudre le problème de répartition des particules. En revanche, la répartition des mailles n'est pas assurée. Nous devons donc opter pour une méthode où le domaine est au moins en partie décomposé.

3.2.3 Méthode hybride

Nous avons vu que qu'aucune des méthodes de décomposition de domaine et de réplication de domaine n'atteignaient les objectifs fixés. Néanmoins, la méthode de réplication de domaine permet de minimiser les déséquilibres de charge de calcul. La méthode de décomposition de domaine, quant à elle, parvient aisément à restreindre la consommation mémoire relative aux mailles. Le principe est donc de trouver un juste milieu pour allier le point fort des deux méthodes. L'approche hybride [12][13][14] permet de réaliser un compromis entre les deux méthodes précédentes. Nous avons vu au chapitre 2 que la méthode hybride peut être considérée de deux façons différentes. Il existe deux méthodes de choix de R. Soit K une constante, nous avons le choix entre $R = \frac{N}{K}$ ou $R = K$. Nous réalisons une étude, sur cette méthode de parallélisation, pour chaque méthode de choix de R. Nous nous placerons dans la meilleure configuration et dans la pire configuration.

Bien que le nombre de sous-domaines soit strictement inférieur au nombre de ressources, la méthode hybride réalise une décomposition de domaine. Par conséquent, cette méthode implique

des communications de particules entre les sous-domaines de la même façon que la méthode de décomposition de domaine. Étant donné que plusieurs ressources se partagent un même sous-domaine, certaines données de simulation, calculées sur les mailles, doivent être échangées, à la fin de la simulation, entre les ressources de calcul au sein d'un même sous-domaine. Nous obtenons alors, au moyen de l'équation 3.33, $\underset{H_R}{C}$, le coût des mécanismes de régulation de charge et de donnée de la méthode hybride.

$$\underset{H_R}{C}^N = f_c(\overline{P}^N t_p q) + f_r(R, \overline{M}^N t'_m) \ . \tag{3.33}$$

Meilleure configuration

Supposons que nous sommes dans la meilleure configuration. Dans cette configuration, les particules sont parfaitement distribuées entre les sous-domaines et les particules ne changent pas de sous-domaine. Pour les mêmes raisons que la méthode de réplication de domaine, les ressources de calcul se partageant un sous-domaine ne sont pas nécessairement parfaitement équilibrées entre elles. Il est, en effet, tout à fait possible qu'une ressource ait plus de particules qu'une autre. Le facteur d'incertitude sur la répartition des particules de la méthode hybride est noté $\underset{H_R}{\varepsilon}$ tel que $0 < \underset{H_R}{\varepsilon} < 1$. Nous obtenons donc $\underset{H_R}{\overline{\breve{M}}}^N = \dfrac{MR}{N}$ et $\underset{H_R}{\overline{\breve{P}}}^N = (1 + \underset{H_R}{\varepsilon})\frac{P}{N}$. Nous obtenons alors, dans la meilleure configuration, $\underset{H_R}{\overline{\breve{\Psi}}}^N$ la charge de calcul maximale (équation 3.34) et $\underset{H_R}{\overline{\breve{T}}}^N$ la consommation mémoire maximale (équation 3.35), selon la méthode de choix de la valeur de R.

$$\begin{cases} Si\ R = K & \underset{H_R}{\overline{\breve{\Psi}}}^N = \frac{MK}{N}\psi_m + (1 + \underset{H_R}{\varepsilon})\frac{P}{N}\psi_p \ ; \\ Si\ R = \dfrac{N}{K} & \underset{H_R}{\overline{\breve{\Psi}}}^N = \frac{M}{K}\psi_m + (1 + \underset{H_R}{\varepsilon})\frac{P}{N}\psi_p \ . \end{cases} \tag{3.34}$$

$$\begin{cases} Si\ R = K & \underset{H_R}{\overline{\breve{T}}}^N = \frac{MK}{N}t_m + (1 + \underset{H_R}{\varepsilon})\frac{P}{N}t_p \ ; \\ Si\ R = \dfrac{N}{K} & \underset{H_R}{\overline{\breve{T}}}^N = \frac{M}{K}t_m + (1 + \underset{H_R}{\varepsilon})\frac{P}{N}t_p \ . \end{cases} \tag{3.35}$$

Avec une infinité de ressources, l'équation 3.36 donne la charge de calcul maximale et l'équation 3.37 donne la consommation mémoire maximale d'une ressource de calcul. Nous observons qu'en utilisant la seconde méthode de choix de R ($R = \frac{N}{K}$), nous ne parvenons pas à réduire de manière proportionnelle le nombre de mailles. Par conséquent, cette méthode est impactée par les mêmes limitations, concernant la répartition de la charge de calcul et des données, que la méthode RD. En revanche, en utilisant la première méthode de choix de R ($R = K$), nous parvenons à une répartition de la charge de calcul et des données proche de l'optimal. En effet, malgré une surcharge et une surconsommation mémoire dues au degré de réplication, la charge de calcul maximale et la consommation mémoire maximale décroissent proportionnellement à l'augmentation du nombre de ressources.

$$\begin{cases} Si\ R = K & \underset{H_R}{\overline{\breve{\Psi}}}^\infty = \lim\limits_{N \to +\infty} \frac{MK}{N}\psi_m + (1 + \underset{H_R}{\varepsilon})\frac{P}{N}\psi_p = 0 \ ; \\ Si\ R = \dfrac{N}{K} & \underset{H_R}{\overline{\breve{\Psi}}}^\infty = \lim\limits_{N \to +\infty} \frac{M}{K}\psi_m + (1 + \underset{H_R}{\varepsilon})\frac{P}{N}\psi_p = \frac{M}{K}\psi_m \ . \end{cases} \tag{3.36}$$

$$\begin{cases} Si\ R = K & \check{\overline{T}}^{\infty}_{H_R} = \lim\limits_{N\to+\infty} \frac{MK}{N} t_m + (1 + \underset{H_R}{\varepsilon}) \frac{P}{N} t_p = 0\ ; \\ Si\ R = \dfrac{N}{K} & \check{\overline{T}}^{\infty}_{H_R} = \lim\limits_{N\to+\infty} \frac{M}{K} t_m + (1 + \underset{H_R}{\varepsilon}) \frac{P}{N} t_p = \frac{M}{K} t_m\ . \end{cases} \tag{3.37}$$

Enfin, l'équation 3.38 donne le coût maximal en fonction de la méthode de choix de R dans la meilleure configuration. En utilisant la première méthode de choix de R, nous parvenons à obtenir un coût décroissant proportionnellement au nombre de ressources. Le coût maximal est bien plus pénalisant en utilisant la seconde méthode de choix de R. En effet, la seconde méthode est très impactée par la réduction. Ce coût est une sérieuse limitation concernant l'extensibilité de cette méthode.

$$\begin{cases} Si\ R = K & \check{C}^{\infty}_{H_R} = \lim\limits_{N\to+\infty} f_c(0) + f_r(K, \frac{MK}{N} t'_m) = L\ ; \\ Si\ R = \dfrac{N}{K} & \check{C}^{\infty}_{H_R} = \lim\limits_{N\to+\infty} f_c(0) + f_r(\frac{N}{K}, \frac{M}{K} t'_m) = +\infty\ . \end{cases} \tag{3.38}$$

Pire configuration

Supposons que nous sommes maintenant dans la pire configuration. Dans cette configuration, Les particules sont toutes dans le même sous-domaine et les particules changent de sous-domaine à chaque itération. Dans la pire configuration, nous obtenons $\hat{\overline{\Psi}}^{N}_{H_R}$ la charge de calcul maximale (équation 3.39) et $\hat{\overline{T}}^{N}_{H_R}$ la consommation mémoire maximale (équation 3.40), selon la méthode de choix de la valeur de R.

$$\begin{cases} Si\ R = K & \hat{\overline{\Psi}}^{N}_{H_R} = \frac{MK}{N} \psi_m + (1 + \underset{H_R}{\varepsilon}) \frac{P}{K} \psi_p\ ; \\ Si\ R = \dfrac{N}{K} & \hat{\overline{\Psi}}^{N}_{H_R} = \frac{M}{K} \psi_m + (1 + \underset{H_R}{\varepsilon}) \frac{PK}{N} \psi_p\ . \end{cases} \tag{3.39}$$

$$\begin{cases} Si\ R = K & \hat{\overline{T}}^{N}_{H_R} = \frac{MK}{N} t_m + (1 + \underset{H_R}{\varepsilon}) \frac{P}{K} t_p\ ; \\ Si\ R = \dfrac{N}{K} & \hat{\overline{T}}^{N}_{H_R} = \frac{M}{K} t_m + (1 + \underset{H_R}{\varepsilon}) \frac{PK}{N} t_p\ . \end{cases} \tag{3.40}$$

Avec une infinité de ressources, l'équation 3.41 donne la charge de calcul maximale et l'équation 3.42 donne la consommation mémoire maximale d'une ressource de calcul. Nous constatons que la deuxième méthode obtient des résultats constants. En effet, nous obtenons la même répartition que dans la meilleure configuration. En revanche, en utilisant la première méthode de choix de R, nous obtenons une répartition de la charge de calcul et des données loin d'être satisfaisante. En effet, la charge de calcul maximale et la consommation mémoire maximale ne décroissent pas proportionnellement à l'augmentation du nombre de ressources. Dans cette configuration, il existe un couple (P,t_p) pour lequel la capacité mémoire d'un nœud de calcul n'est pas suffisante pour stocker toutes les données de simulation. Le degré de réplication permet néanmoins d'atténuer la surcharge et la surconsommation mémoire dues à la très mauvaise répartition des particules.

$$\begin{cases} Si\ R = K & \overset{\hat{}}{\overline{\Psi}}^{\infty}_{H_R} = \lim\limits_{N\to+\infty} \frac{MK}{N}\psi_m + (1 + \underset{H_R}{\varepsilon})\frac{P}{K}\psi_p = (1 + \underset{H_R}{\varepsilon})\frac{P}{K}\psi_p\ ; \\ Si\ R = \dfrac{N}{K} & \overset{\hat{}}{\overline{\Psi}}^{\infty}_{H_R} = \lim\limits_{N\to+\infty} \frac{M}{K}\psi_m + (1 + \underset{H_R}{\varepsilon})\frac{PK}{N}\psi_p = \frac{M}{K}\psi_m\ . \end{cases} \tag{3.41}$$

$$\begin{cases} Si\ R = K & \overset{\hat{}}{\overline{T}}^{\infty}_{H_R} = \lim\limits_{N\to+\infty} \frac{MK}{N}t_m + (1 + \underset{H_R}{\varepsilon})\frac{P}{K}t_p = (1 + \underset{H_R}{\varepsilon})\frac{P}{K}t_p\ ; \\ Si\ R = \dfrac{N}{K} & \overset{\hat{}}{\overline{T}}^{\infty}_{H_R} = \lim\limits_{N\to+\infty} \frac{M}{K}t_m + (1 + \underset{H_R}{\varepsilon})\frac{PK}{N}t_p = \frac{M}{K}t_m\ . \end{cases} \tag{3.42}$$

Enfin, l'équation 3.43 donne le coût maximal en fonction de la méthode de choix de R dans la pire configuration. En utilisant la première méthode de choix de R, nous ne parvenons pas à obtenir un coût décroissant proportionnellement au nombre de ressources. Le coût maximal, en utilisant la seconde méthode de choix de R, demeure quant à lui extrêmement pénalisant.

$$\begin{cases} Si\ R = K & \hat{C}^{\infty}_{H_R} = \lim\limits_{N\to+\infty} f_c((1 + \underset{H_R}{\varepsilon})\frac{P}{K}t_p q) + f_r(K, \frac{MK}{N}t'_m) \\ & \hat{C}^{\infty}_{H_R} = f_c((1 + \underset{H_R}{\varepsilon})\frac{P}{K}t_p) + L\ ; \\ Si\ R = \dfrac{N}{K} & \hat{C}^{\infty}_{H_R} = \lim\limits_{N\to+\infty} f_c((1 + \underset{H_R}{\varepsilon})\frac{PK}{N}t_p q) + f_r(\frac{N}{K}, \frac{M}{K}t'_m) = +\infty\ . \end{cases} \tag{3.43}$$

La méthode H_R permet de réaliser un compromis entre les méthodes RD et DD. Selon la méthode de choix de R utilisée, cette méthode se rapproche soit de la méthode DD soit de la méthode RD. Elle ne parvient malheureusement pas à corriger simultanément les limitations des deux méthodes. Pour conclure, alors que la méthode hybride avec R dépendant de N ne permet d'atteindre les objectifs que ce soit dans la meilleure ou la pire configuration, nous observons que la méthode hybride avec R constant atteint les objectifs dans la meilleure configuration.

3.2.4 Conclusion

Les méthodes RD, DD et H_R ne permettent pas de répondre efficacement aux besoins. Ces méthodes sont en effet limitées soit par une mauvaise distribution des charges de calcul, soit par une mauvaise distribution des données, soit par une mauvaise distribution à la fois des charges et des données. En effet, si la méthode RD garantit une répartition des particules satisfaisante, elle ne permet pas en revanche de distribuer les mailles. La méthode DD, quant à elle, ne garantit pas une répartition suffisamment équitable des particules. Enfin, la méthode H_R ne permet pas non plus, à priori, de satisfaire les objectifs pour toutes les configurations.

L'étude théorique réalisée a permis de déterminer les points faibles et les points forts des méthodes classiques de parallélisation. Nous proposons maintenant d'affiner notre analyse à l'aide d'études expérimentales.

3.3 Études expérimentales

Nous avons observés de nombreuses limitations pour chacune de méthodes étudiées dans ce chapitre dans l'analyse théorique. Nous souhaitons valider expérimentalement les conclusions déduites de cette analyse. Nous nous appuierons sur les cas test présentés au chapitre 2, sous-section 2.1.6. La méthodologie d'expérimentation utilisée est celle que nous avons présentée en section 2.4. Nous avons évalué la méthode H_R avec $R = 2$, $R = 4$ et $R = \frac{N}{2}$. Notons que les écarts entre les sous-domaines en nombre de mailles est au maximum de 5 %.

3.3.1 Étude de régulation de charge et de donnée

L'analyse théorique nous a permis de conclure que les méthodes DD et H_R ne garantissaient pas une bonne répartition des particules. Elle nous a également permis de déduire que la méthode RD ne provoquait que très peu de déséquilibre de particules. Nous proposons donc de confirmer ces constatations par une étude expérimentale de performance en termes de régulation de charge. Nous allons présenter les résultats en déséquilibre de charge (voir l'équation 2.6) et en surconsommation mémoire (voir l'équation 2.7). Nous avons inséré pour chacun des cas test 128 millions de particules. Nous avons réalisé cette étude comparative pour 32 et 256 ressources de calcul.

Étude de régulation de charge

La figure 3.1 présente les résultats obtenus sur le cas test homogène pour 32 et 256 ressources. Nous remarquons que le déséquilibre de charge augmente quand le degré de réplication diminue. La méthode de décomposition de domaine obtient les pires résultats ($R = 1$) alors que la méthode de réplication de domaine obtient un équilibre quasiment parfait à chaque itération ($R = N$). Ce résultat est tout à fait en adéquation avec l'analyse théorique. Pour le cas de la méthode $H_{\frac{N}{2}}$, nous observons que le déséquilibre est de 2 durant les premières itérations avant de retomber à 1 puis de remonter les itérations suivantes. Étant donné que nous avons seulement 2 sous-domaines, l'explication est simple. En effet, durant les premières itérations, seul l'un des deux sous-domaines possède des particules (déséquilibre d'un facteur 2), puis les particules vont progressivement être transférées du premier sous-domaine vers le deuxième (le déséquilibre baisse) jusqu'à ce que ce dernier devienne surchargé (le déséquilibre remonte). Enfin, nous observons que les déséquilibres observés pour les méthodes DD et H_R avec R constant augmentent quand N augmente. En revanche, les déséquilibres des méthodes RD et H_R avec R dépendant de N sont pratiquement identique entre $N = 32$ et $N = 256$. Cela confirme que les répartitions des particules des méthodes RD et H_R avec $R = \frac{N}{K}$ ne dépendent pas de N.

La figure 3.2 présente les résultats obtenus sur le cas test hétérogène par insertion centralisée pour 32 et 256 ressources. Nous observons que la méthode RD obtient encore une fois un équilibre quasiment parfait à chaque itération. Nous observons que les déséquilibres se sont accentués pour les méthodes DD et H_R avec R constant par rapport au premier cas test, allant même jusqu'à un facteur N pour la méthode DD. L'unicité de la maille source est la cause principale. En effet, les déséquilibres les plus conséquents sont observés durant les premières itérations. À ce moment là, seules quelques mailles possèdent des particules. Les particules sont alors dans un seul sous-domaine. En revanche, les déséquilibres des méthodes RD et H_R avec R dépendant de N demeurent pratiquement inchangés par rapport au cas test précédent.

La figure 3.3 présente les résultats obtenus sur le cas test hétérogène par insertion latérale pour 32 et 256 ressources. Nous observons que les méthodes RD et H_R avec $R = \frac{N}{K}$ conservent toujours la même tendance. Ce cas test est intéressant car il correspond, en quelque sorte, à un mélange des deux premiers cas test. Les résultats des méthodes DD et H_R avec R constant le confirment. En effet, le déséquilibre initial est assez comparable à celui observé avec le cas test homogène mais cette fois le déséquilibre ne suis plus une trajectoire décroissante. En effet, les déséquilibres ont tendance à osciller de bas en haut pour $N = 32$. Ce phénomène s'explique essentiellement pas les interactions entre les particules avec la zone du domaine à forte interactivité. En effet, ces interactions produiront des absorptions de particules conjuguées à des changements de direction et de vitesse des particules impactant la répartition des particules entre les sous-domaines. En revanche, avec $N = 256$, il est intéressant de constater que les déséquilibres des méthodes DD et H_R avec R constant restent très élevés tout au long de la simulation. En effet, si le déséquilibre

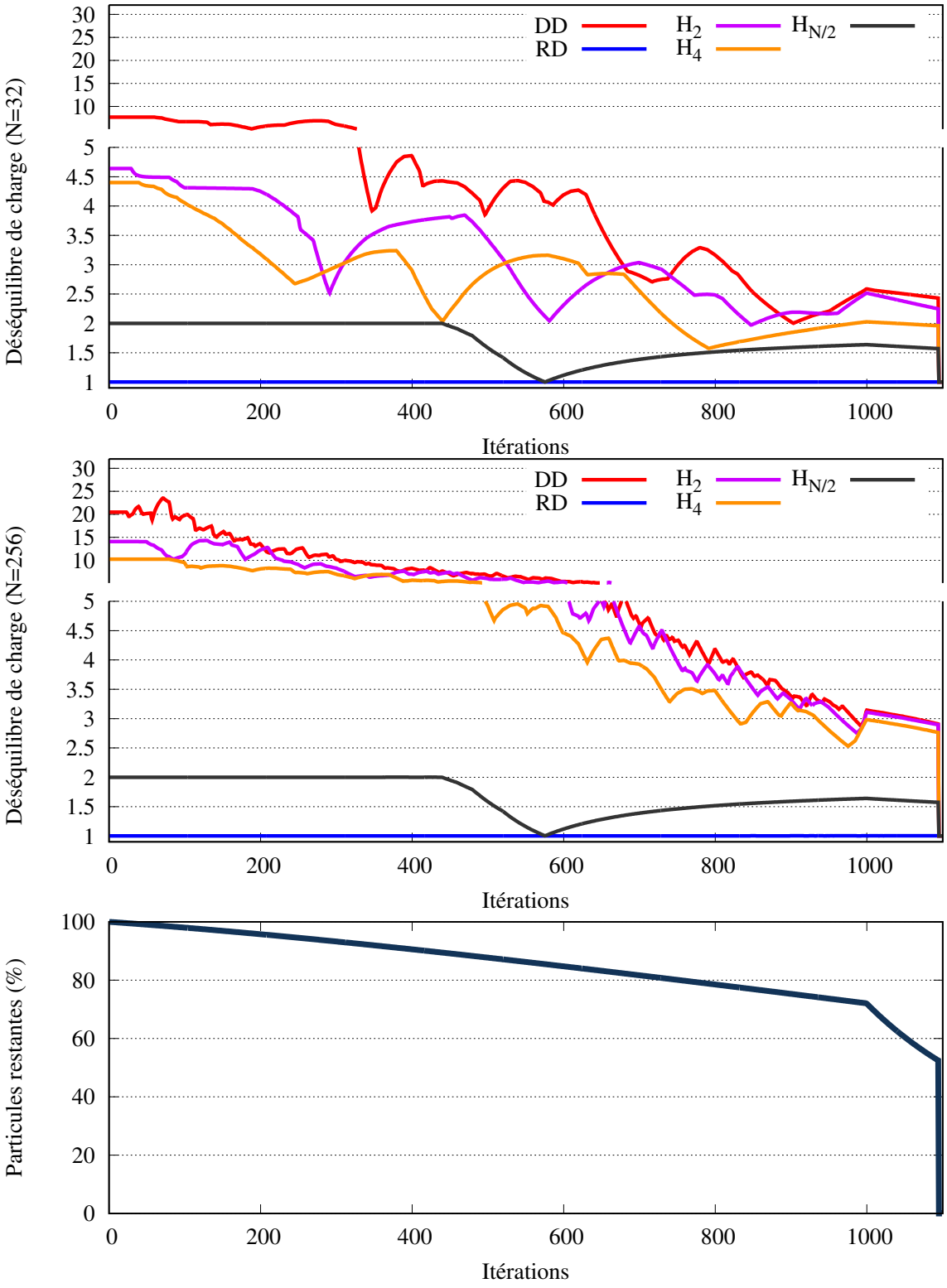


FIGURE 3.1 – Étude du déséquilibre de charge pour le cas test homogène (voir figure 2.3).

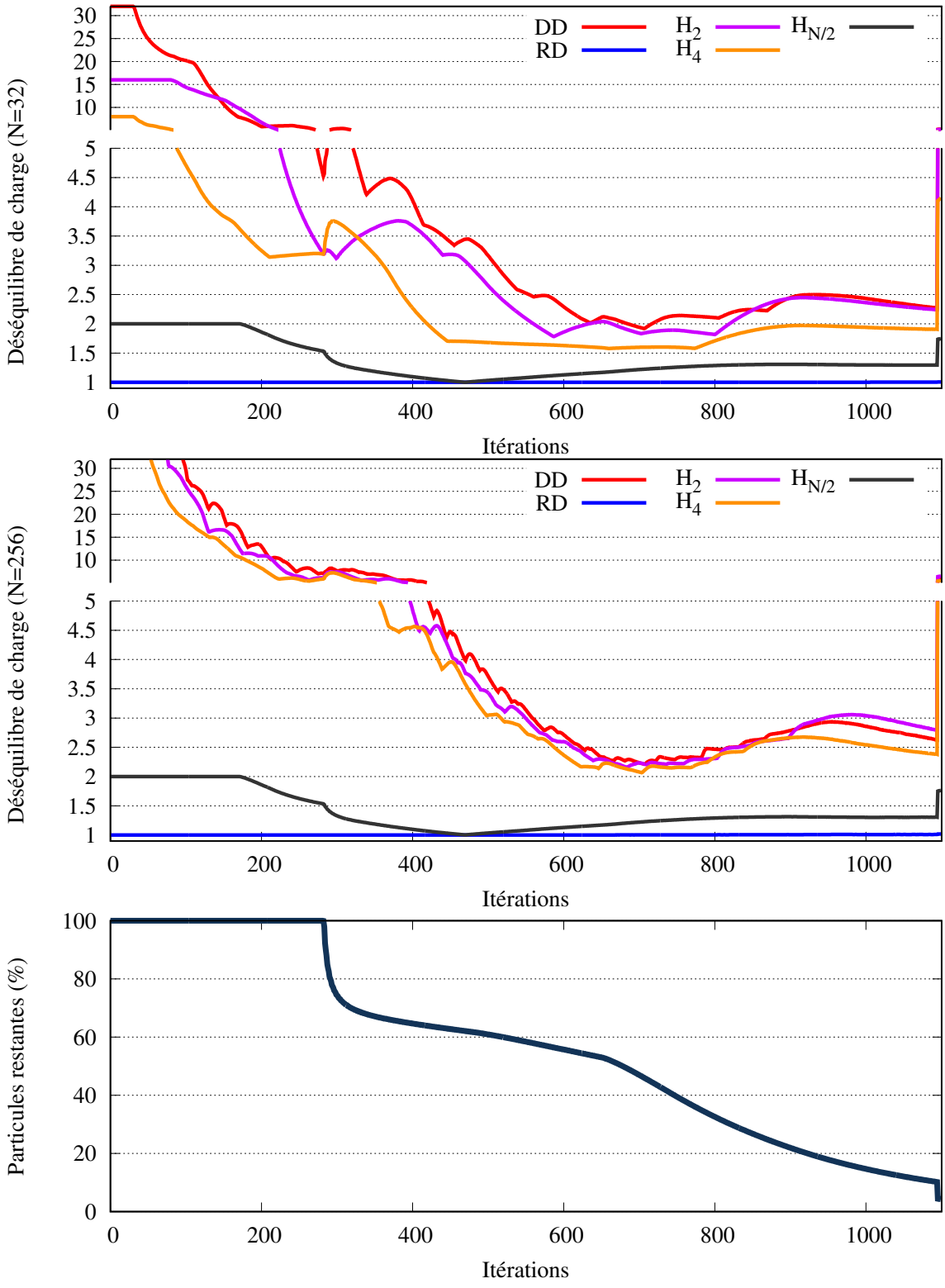


FIGURE 3.2 – Étude du déséquilibre de charge pour le cas test hétérogène par insertion centralisée (voir figure 2.4).

N	Méthodes	DD	H_2	H_4	$H_{\frac{N}{2}}$	RD
32	Particules	**7.68**	**4.64**	**4.4**	2.0	1.0
	Mailles	1.05	2.1	**4.2**	**16.7**	**26.0**
256	Particules	**23.2**	**14.1**	**10.2**	2.0	1.0
	Mailles	1.05	2.1	**4.2**	**86.9**	**74.5**

TABLE 3.1 – Surconsommation mémoire pour le cas homogène.

initial est largement moins important que sur le cas test hétérogène par insertion centralisée, la zone à forte interactivité produit davantage de déséquilibre.

Nous avons observés que la méthode RD n'est pratiquement pas déséquilibrée tel que l'analyse théorique l'a présenté. De même, la méthode H_R avec $R = \frac{N}{K}$ se comporte de la même façon, au facteur K près, que la méthode RD. Nous avons également pu observer les performances assez désastreuses de la méthode DD en termes de répartition de particules. De même, nous avons vu que la méthode H_R avec R constant induit des déséquilibres de charge également relativement importants. Le degré de réplication permet d'atténuer le déséquilibre.

L'analyse théorique avait pour autre objectif de valider la capacité d'une méthode à limiter la consommation mémoire. L'analyse théorique nous avait permis de conclure que les méthodes DD et H_R ne permettait pas de garantir cette limitation de consommation mémoire à cause du manque de régulation des particules. L'analyse théorique nous avait alors permis de déduire également que l'absence de distribution des mailles en utilisant la méthode RD constituait une limitation de la méthode RD en termes de consommation mémoire. Nous proposons donc maintenant de nous intéresser aux performances des méthodes en termes de régulation de donnée.

Étude de régulation de donnée

La table 3.1 présente les résultats obtenus en régulation de donnée sur le cas test homogène pour 32 et 256 ressources. Nous observons que l'augmentation du degré de réplication permet de réduire la consommation mémoire liée au particules mais implique également d'augmenter la consommation mémoire relative aux mailles. Une particularité est observable avec la méthode RD : en théorie, la surconsommation mémoire de cette méthode avec N ressources devrait être égal à N. Or, nous observons une surconsommation mémoire inférieure à N. Nous précisons que nous ne comptabilisons, dans le calcul du nombre de mailles stockées en mémoire, que les mailles possédant des particules. Nous avons un maillage de 1000×1000 mailles, soit 1 million de mailles. Or, avec $N = 256$ et 128 millions de particules, nous avons donc 500000 particules par ressource. Autrement dit, nous avons moins de particules par ressource qu'il n'y a de mailles en tout. Par conséquent, nous ne pouvons pas avoir des particules sur toutes les mailles. Le même phénomène est observable pour la méthode H_R avec $R = \frac{N}{K}$. Étonnamment, nous obtenons une surconsommation mémoire en nombre de mailles plus importante que la méthode RD avec 256 ressources. Cela peut être expliqué par le fait qu'au niveau de chaque ressource, le nombre de particules par maille étant plus élevé, les particules occupent davantage l'ensemble des mailles. En effet, la dispersion des particules sur les mailles est moins prononcée. Nous observons tout de même une forte surconsommation mémoire relative aux mailles pour les méthodes RD et H_R avec $R = \frac{N}{K}$.

La table 3.2 présente les résultats obtenus en régulation de donnée sur le cas test hétérogène avec insertion centralisée pour 32 et 256 ressources. Nous constatons que la surconsommation mémoire relative aux particules pour les méthodes DD et H_R avec R constant est maximale. En effet, elle est égale à N. Cela est dû au fait qu'il n'y ait qu'une maille source. La surconsommation

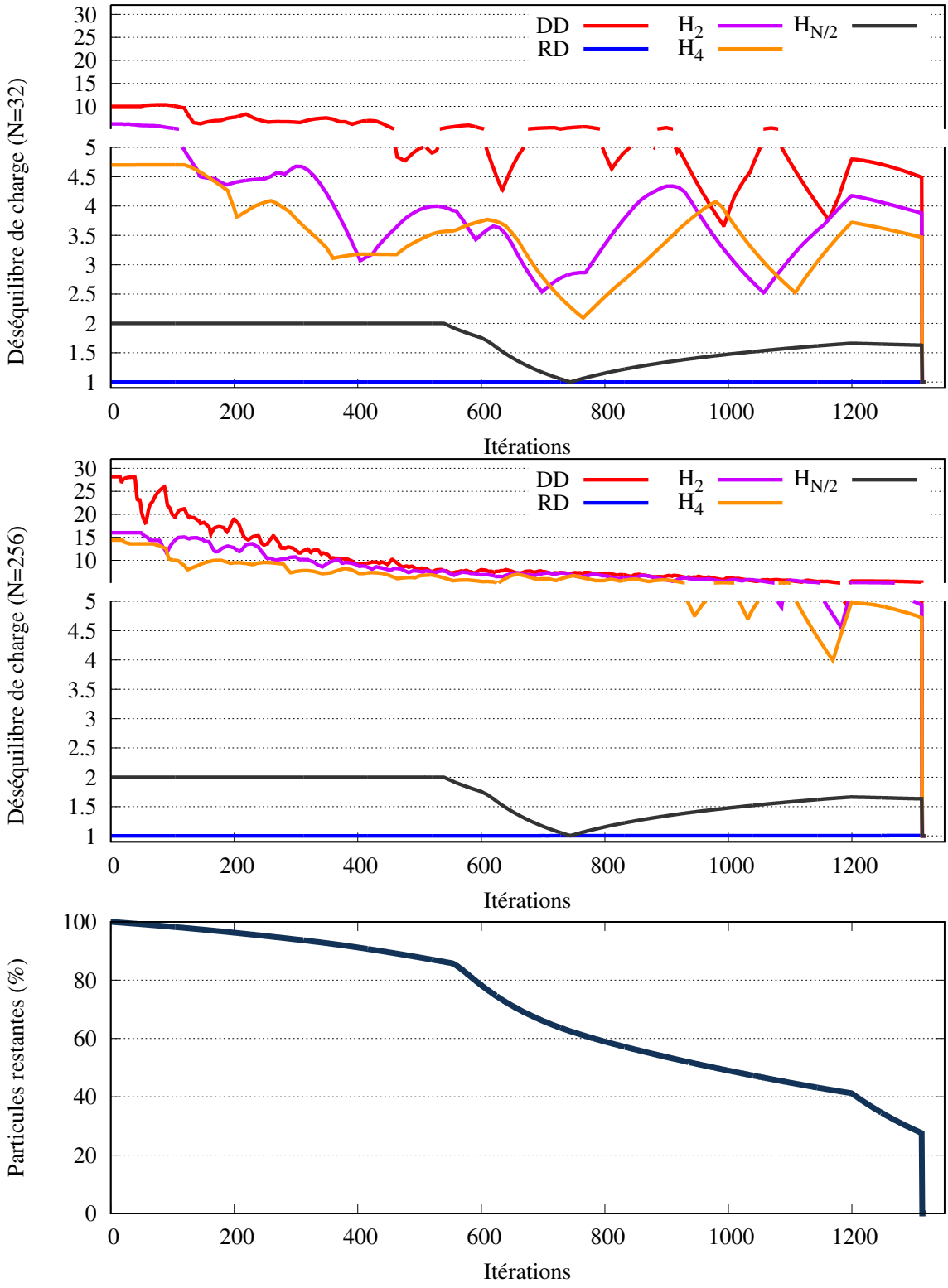


FIGURE 3.3 – Étude du déséquilibre de charge pour le cas test hétérogène par insertion latérale (voir figure 2.5).

N	Méthodes	DD	H_2	H_4	$H_{\frac{N}{2}}$	RD
32	Particules	**32.0**	**16.0**	**8.0**	2.0	1.0
	Mailles	1.05	2.1	**4.2**	**13.7**	**21.2**
256	Particules	**256**	**128**	**64.0**	2.0	1.0
	Mailles	1.05	2.1	**4.2**	**52.0**	**55.1**

TABLE 3.2 – Surconsommation mémoire pour le cas hétérogène par insertion centralisée.

N	Méthodes	DD	H_2	H_4	$H_{\frac{N}{2}}$	RD
32	Particules	**10.2**	**6.2**	**4.7**	2.0	1.0
	Mailles	1.05	2.1	**4.2**	**14.7**	**17.5**
256	Particules	**28.2**	**16.0**	**14.4**	2.0	1.0
	Mailles	1.04	2.08	**4.19**	**75.5**	**59.2**

TABLE 3.3 – Surconsommation mémoire pour le cas hétérogène par insertion latérale.

mémoire relative aux mailles est, quant à elle, inchangée. Les méthodes RD et H_R avec $R = \frac{N}{K}$ obtiennent des résultats assez comparables par rapport au cas test précédent.

La table 3.3 présente les résultats obtenus en régulation de donnée sur le cas test hétérogène avec insertion latérale pour 32 et 256 ressources. Nous observons que les résultats obtenus sont très proches de ceux obtenus avec le cas test homogène. Nous constatons une surconsommation mémoire relative aux particules pour les méthodes DD et H_R avec R constant légèrement plus élevé par rapport au cas test homogène.

Pour conclure, le constat de cette étude expérimentale est que ces méthodes ne parviennent pas à satisfaire les objectifs de régulation de charge et de donnée. La méthode H_R permet une réduction significative de la consommation mémoire concernant les mailles par rapport à la méthode de réplication de domaine ainsi qu'une baisse de la consommation mémoire concernant les particules en comparaison avec la méthode de décomposition de domaine. Nous proposons d'analyser les conséquences par une étude en extensibilité forte.

3.3.2 Étude d'extensibilité forte

Nous proposons de réaliser une étude comparative en temps d'exécution des méthodes DD, RD et H_R en extensibilité forte. Nous allons étudier l'évolution du temps d'exécution de 1 à 256 ressources de calcul. Nous fixons le nombre total de particules insérées à 128 millions. Nous calculons l'efficacité parallèle pour chacune des méthodes en prenant la même référence, à savoir le temps d'exécution avec $N = 1$, en utilisant donc 16 cœurs de calcul.

Efficacité parallèle

La figure 3.4 présente les résultats obtenus en efficacité parallèle sur le cas test homogène jusqu'à 256 ressources. Nous observons que l'efficacité est fonction du degré de réplication. En effet, la meilleure efficacité est obtenue par la méthode RD alors que la pire est obtenue par la méthode DD. L'impact du degré de réplication, sur la qualité de la répartition de la charge de calcul, explique certainement cette classification. Nous constatons par ailleurs que, bien que parfaitement équilibrée en nombre de particules par ressource, la méthode RD obtient une efficacité parallèle avec 256 ressources assez faible (environ 0,3). L'efficacité de la méthode H_R, avec R constant, plonge dès que le nombre de ressources devient plus grand que le degré de réplication. En effet,

avec $N \leq R$, la méthode H_R se comporte de la même façon que la méthode RD. Puis, à partir de $N > R$, le comportement de la méthode H_R, avec R constant, tend vers le comportement de la méthode DD. Nous observons le comportement opposé avec la méthode H_R avec $R = \frac{N}{K}$. En effet, la méthode H_R, avec $R = \frac{N}{K}$, se comporte identiquement à la méthode DD tant que $N \leq K$. Puis, étant donné que le degré de réplication augmente avec le nombre de ressources, le comportement de la méthode H_R, avec $R = \frac{N}{K}$, tend vers celui de la méthode RD.

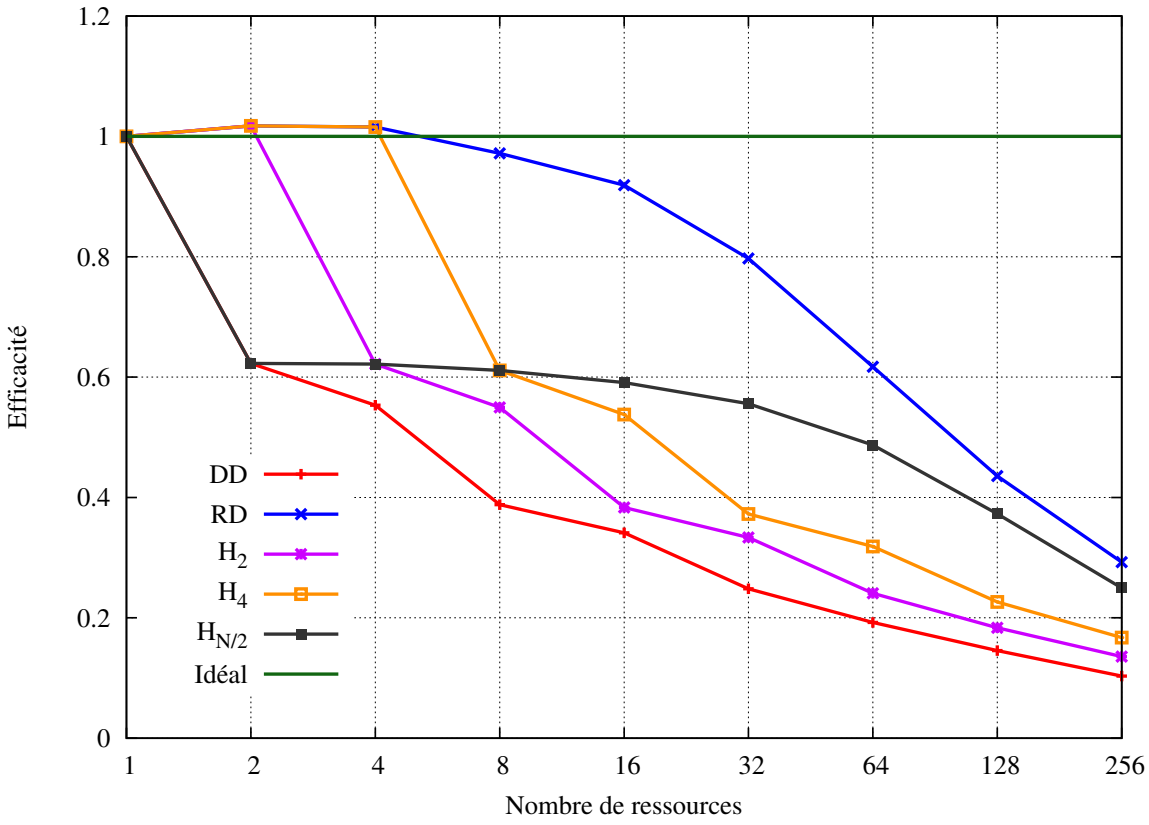


FIGURE 3.4 – Extensibilité forte pour le cas test homogène (meilleur en haut).

La figure 3.5 présente les résultats obtenus en efficacité parallèle sur le cas test hétérogène avec insertion centralisé jusqu'à 256 ressources. Nous observons les mêmes tendances que pour le cas précédent. Toutefois, les écarts d'efficacité se sont creusés entre les méthodes RD, H_R avec $R = \frac{N}{K}$ et les méthodes DD, H_R avec R constant. En effet, les efficacités des méthodes DD, H_R avec R constant ont chuté de près de 50 % en moyenne contre moins de 20 % pour les méthodes RD, H_R avec $R = \frac{N}{K}$.

La figure 3.6 présente les résultats obtenus en efficacité parallèle sur le cas test hétérogène avec insertion latérale jusqu'à 256 ressources. Nous observons, ici aussi, les mêmes tendances que pour les deux premiers cas. Comme dans le cas précédent, les écarts d'efficacité sont plus importants entre les méthodes RD, H_R avec $R = \frac{N}{K}$ et les méthodes DD, H_R avec R constant que pour le cas homogène.

Nous avons constaté qu'augmenter le degré de réplication permettait de se rapprocher des performances de la méthode RD. Cependant, bien qu'équilibrée, la méthode RD ne permet pas d'obtenir des résultats en extensibilité forte pleinement satisfaisants. Nous proposons donc de réaliser un profilage de l'application afin d'identifier les origines des pertes de performance observées.

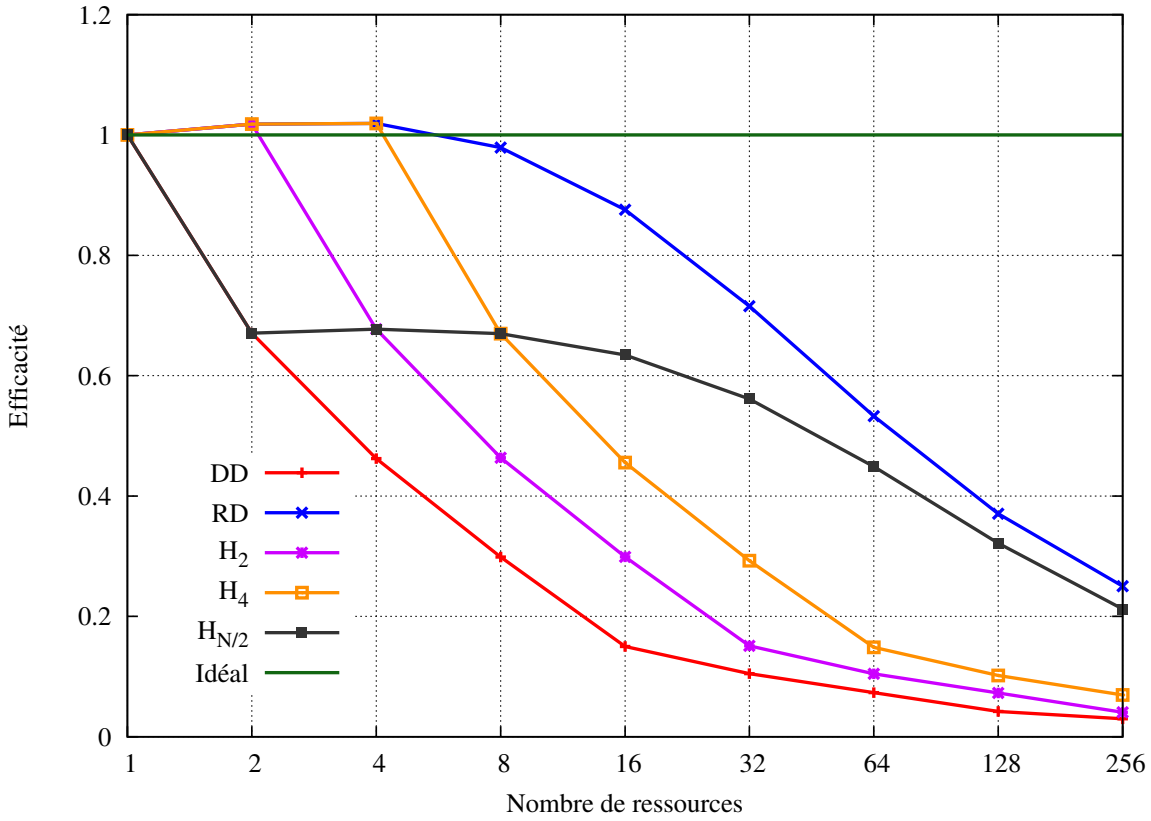


FIGURE 3.5 – Extensibilité forte pour le cas test hétérogène par insertion centralisée (meilleur en haut).

Profilage

La table 3.4 présente le profilage de l'application pour le cas test homogène en extensibilité forte. Nous constatons que le temps de traitement des particules diminue lorsque le degré de réplication augmente. Cela est cohérent avec les mesures de déséquilibre en nombre de particules par ressource réalisées en section 3.3.1. Inversement, le temps de traitement des mailles augmente lorsque le degré de réplication augmente. Nous constatons que la méthode RD et la méthode H_R avec $R = \frac{N}{K}$ obtiennent des temps de traitement des mailles largement plus élevés que les autres méthodes. Le temps de traitement des mailles est donc bel et bien la cause principale de la perte d'efficacité parallèle de la méthode RD et, dans une moindre mesure, de la méthode H_R avec $R = \frac{N}{K}$.

Méthodes	DD	H_2	H_4	$H_{\frac{N}{2}}$	RD
Particules + Communications	**298**	**223**	**176**	70.9	38.4
Mailles	9.06	9.32	9.78	**50.0**	**71.7**
Partitionnement(s)	1.57	0.83	0.51	0.24	0.0
Régulation (hors partitionnement)	0.0	0.0	0.0	0.0	0.0
Temps total	309	233	186	121	110

TABLE 3.4 – Profilage (secondes) en extensibilité forte pour le cas homogène.

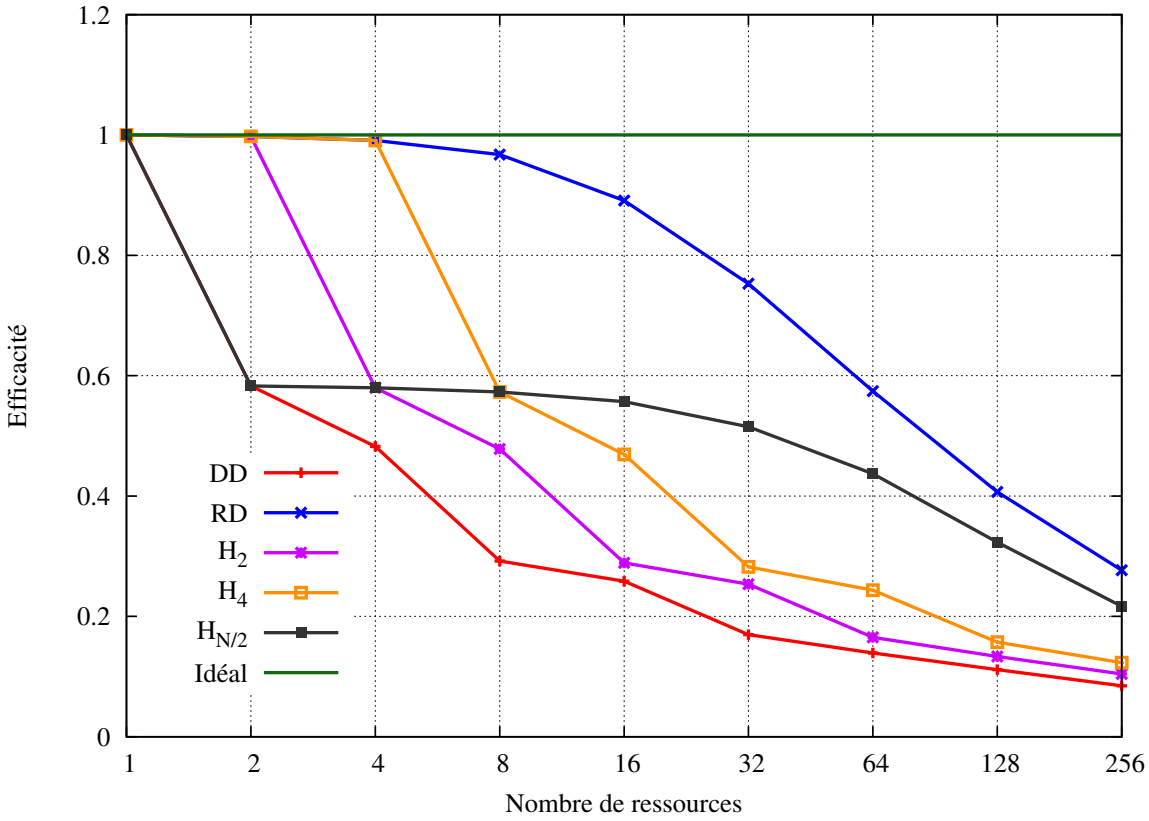


FIGURE 3.6 – Extensibilité forte pour le cas test hétérogène par insertion latérale (meilleur en haut).

La table 3.5 présente le profilage de l'application pour le cas test hétérogène par insertion centralisée en extensibilité forte. Nous observons les mêmes phénomènes que pour le cas homogène. Les déséquilibres de charge, plus élevés que pour le cas homogène, impliquent des temps de traitement de particules et de communication conséquents.

La table 3.6 présente le profilage de l'application pour le cas test hétérogène par insertion latérale en extensibilité forte. Nous observons les mêmes phénomènes que pour les cas précédents. Le fait que les résultats soient très semblables d'un cas test à l'autre illustre l'aspect statique de ces méthodes.

Le profilage de l'application nous permet de conclure que l'absence de décomposition de domaine constitue une réelle limitation de la méthode RD en extensibilité forte. Nous avons également observé que les déséquilibres en termes de particules dégradaient les performances des méthodes de décomposition de domaine et hybride.

Conclusion

En terme d'extensibilité forte, le méthode RD obtient les meilleures performances. Toutefois, compte-tenu de la différence conséquente que nous avons observée en termes de répartition des particules, nous pouvons conclure que l'absence de décomposition de domaine constitue une réelle limitation de la méthode RD en extensibilité forte. La méthode DD a montré des performances très insuffisantes, souffrant d'un déséquilibres de charge trop important. Les résultats de la méthode H_R

Méthodes	DD	H_2	H_4	$H_{\frac{N}{2}}$	RD
Particules + Communications	**744**	**549**	**313**	61.3	30.2
Mailles	10.7	11.0	11.3	**37.6**	**60.6**
Partitionnement(s)	1.57	0.83	0.51	0.24	0.0
Régulation (hors partitionnement)	0.0	0.0	0.0	0.0	0.0
Temps total	757	561	325	99.1	90.7

TABLE 3.5 – Profilage (secondes) en extensibilité forte pour le cas hétérogène par insertion centralisée.

Méthodes	DD	H_2	H_4	$H_{\frac{N}{2}}$	RD
Particules + Communications	**330**	**264**	**213**	100	37.7
Mailles	10.5	10.5	10.8	**26.5**	**67.2**
Partitionnement(s)	1.56	0.81	0.52	0.2	0.0
Régulation (hors partitionnement)	0.0	0.0	0.0	0.0	0.0
Temps total	342	276	224	127	105

TABLE 3.6 – Profilage (secondes) en extensibilité forte pour le cas hétérogène par insertion latérale.

n'ont fait que confirmer les conclusions de l'analyse théorique, en l'occurrence que cette méthode permet de faire un compromis mais ne permet pas de résoudre les problèmes des méthodes RD et DD.

3.3.3 Étude d'extensibilité faible

Nous proposons de réaliser une étude comparative en temps d'exécution des méthodes DD, RD et H_R en extensibilité faible. Nous allons étudier l'évolution du temps d'exécution de 1 à 256 ressources de calcul. Nous fixons le nombre moyen de particules insérées par ressource à 4 millions, autrement dit jusqu'à 1,024 milliard de particules avec 256 ressources.

Efficacité parallèle

La figure 3.7 présente les résultats obtenus en efficacité parallèle sur le cas test homogène jusqu'à 256 ressources en extensibilité faible. Nous constatons que l'efficacité parallèle baisse lorsque le degré de réplication diminue. Nous observons que les efficacités parallèles de la méthode RD et de la méthode $H_R, R = \frac{N}{K}$, à partir de $N = K$, sont relativement constantes. Ce résultat est plutôt logique dans la mesure où le nombre de particules et le nombre de mailles par ressource est constant. Nous pouvons admettre que, pour $N = 256$, le coût de la réduction sur les mailles est très peu impactant.

La figure 3.8 présente les résultats obtenus en efficacité parallèle sur le cas test hétérogène par insertion centralisée jusqu'à 256 ressources en extensibilité faible. Nous observons les mêmes tendances concernant les efficacités parallèles que pour le cas test homogène. Cependant, nous n'avons pas de mesure, pour les méthodes DD et H_R, au delà de respectivement $N = 64$ et $N = 64R$. Cette absence de mesures est due au fait que ces méthodes nécessitent une capacité mémoire excessive entraînant l'interruption immédiate de la simulation.

La figure 3.9 présente les résultats obtenus en efficacité parallèle sur le cas test hétérogène par insertion latérale jusqu'à 256 ressources en extensibilité faible. Nous observons, ici aussi, les mêmes tendances que pour le cas test homogène.

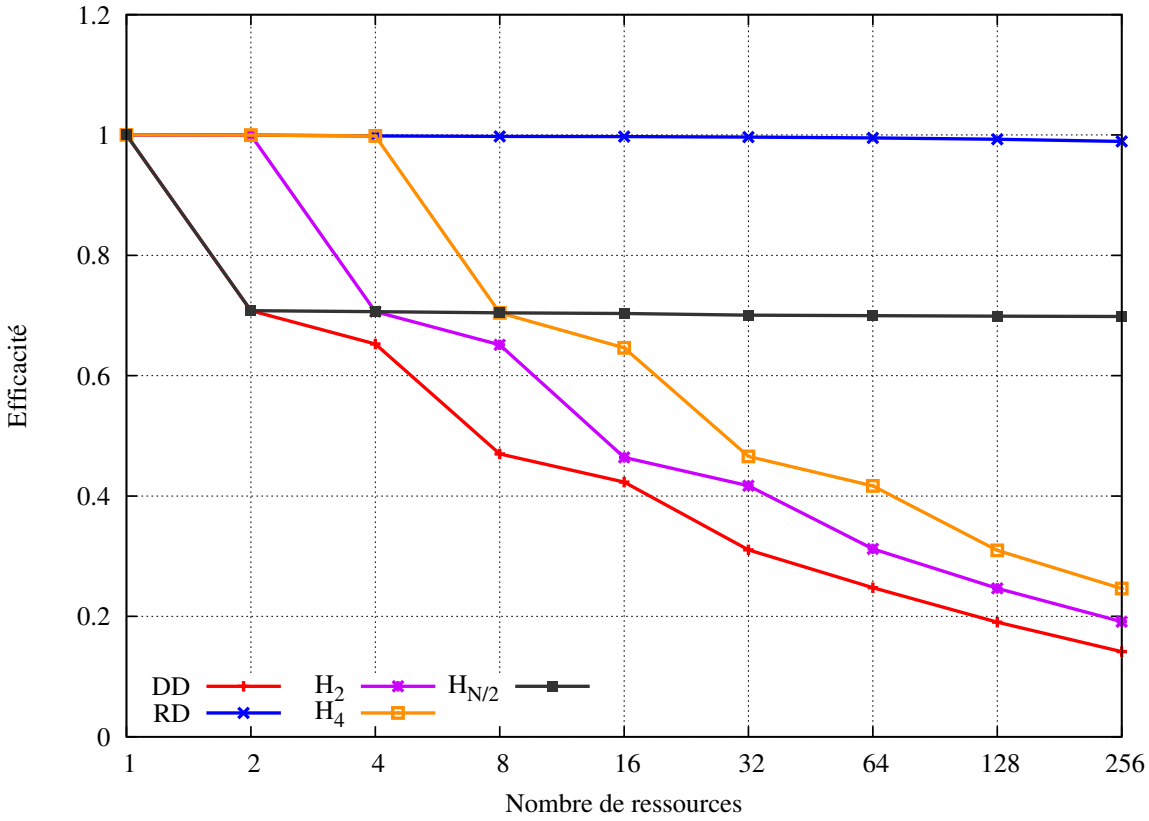


FIGURE 3.7 – Extensibilité faible pour le cas test homogène (meilleur en haut).

Nous constatons donc, également en extensibilité faible, que le degré de réplication permet d'améliorer les performances en temps d'exécution. En effet, la méthode RD obtient toujours les meilleures performances alors que la méthode DD obtient toujours les pires. Les écarts de performance en extensibilité faible sont toutefois plus prononcés qu'en extensibilité forte. Nous proposons donc de profiler l'application afin d'en déterminer l'origine.

Profilage

La table 3.7 présente le profilage de l'application pour le cas test homogène en extensibilité faible. Nous observons que les temps de traitement des particules et des communications sont très élevés pour les méthodes les plus déséquilibrées. En effet, les écarts de temps entre une méthode équilibrée et une méthode déséquilibrée sont bien plus conséquents que pour l'étude en extensibilité forte. Nous constatons également qu'en proportion le temps passé dans le traitement des mailles est moins important pour la méthode RD et H_R avec $R = \frac{N}{K}$.

La table 3.8 présente le profilage de l'application pour le cas test hétérogène par insertion centralisée en extensibilité faible. En conséquence du dépassement mémoire, certaines mesures sont absentes. Nous pouvons remarquer que le temps de communication et de traitement des particules pour la méthode H_4 est presque 14 fois plus élevé que la méthode RD. Cette mesure illustre bien l'impact du déséquilibre du nombre de particules par ressource.

La table 3.9 présente le profilage de l'application pour le cas test hétérogène par insertion latérale en extensibilité faible. Les résultats obtenus avec ce cas test viennent appuyer les consta-

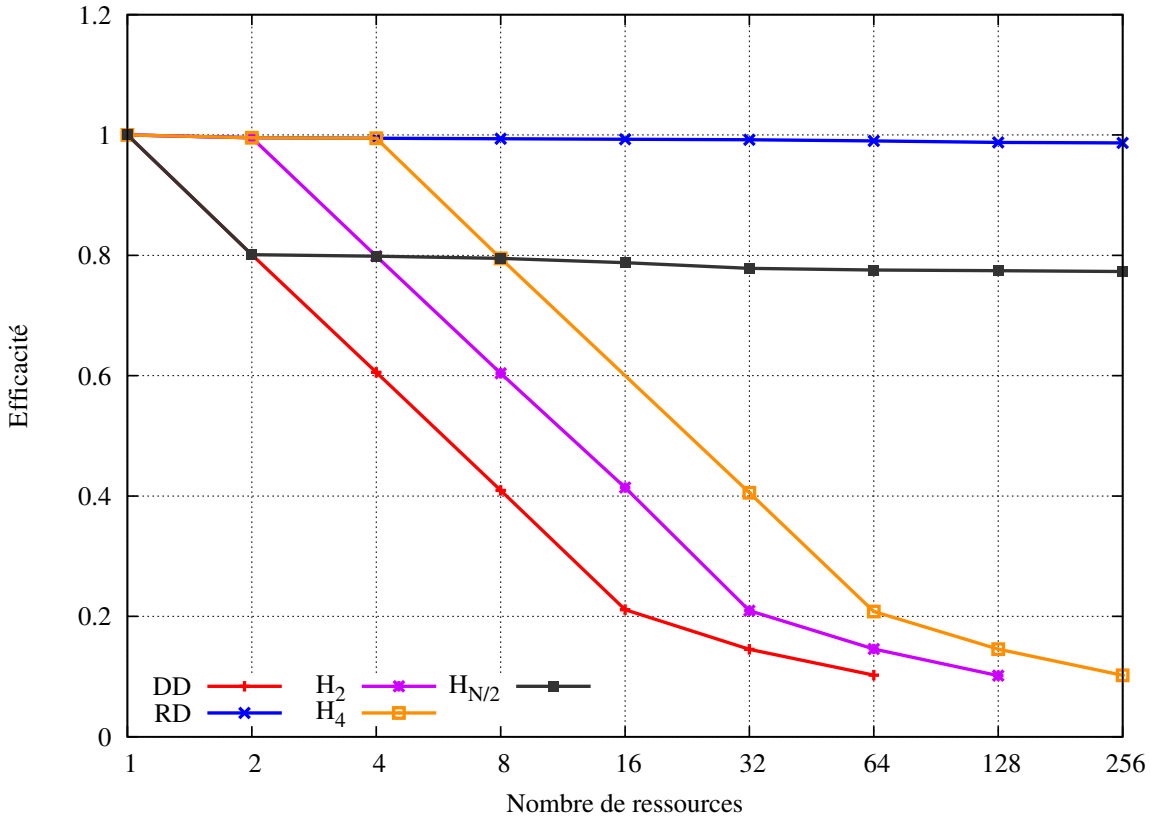


FIGURE 3.8 – Extensibilité faible pour le cas test hétérogène par insertion centralisée (meilleur en haut).

tions réalisées jusque là. En effet, nous constatons un surcoût très conséquent, pour les méthodes DD et H_R, par rapport au traitement des particules et aux communications.

Le profilage de l'application nous permet de conclure que les déséquilibres de charge constituent une forte limitation des méthodes DD et H_R en extensibilité faible.

Conclusion

En terme d'extensibilité faible, la méthode RD obtient clairement les meilleures performances. Cette fois, la différence conséquente que nous avons observée en termes de répartition des particules, s'est faite beaucoup plus ressentir. C'est peu surprenant dans le mesure où le temps de calcul lié au traitement des particules demeure constant en extensibilité faible. Les méthodes DD et H_R

Méthodes	DD	H_2	H_4	$H_{\frac{N}{2}}$	RD
Particules + Communications	**2311**	**1704**	**1326**	424	263
Mailles	9.11	9.41	9.86	**42.7**	**67.7**
Partitionnement(s)	1.57	0.83	0.51	0.24	0.0
Régulation (hors partitionnement)	0.0	0.0	0.0	0.0	0.0
Temps total	2322	1715	1336	467	331

TABLE 3.7 – Profilage (secondes) en extensibilité faible pour le cas homogène.

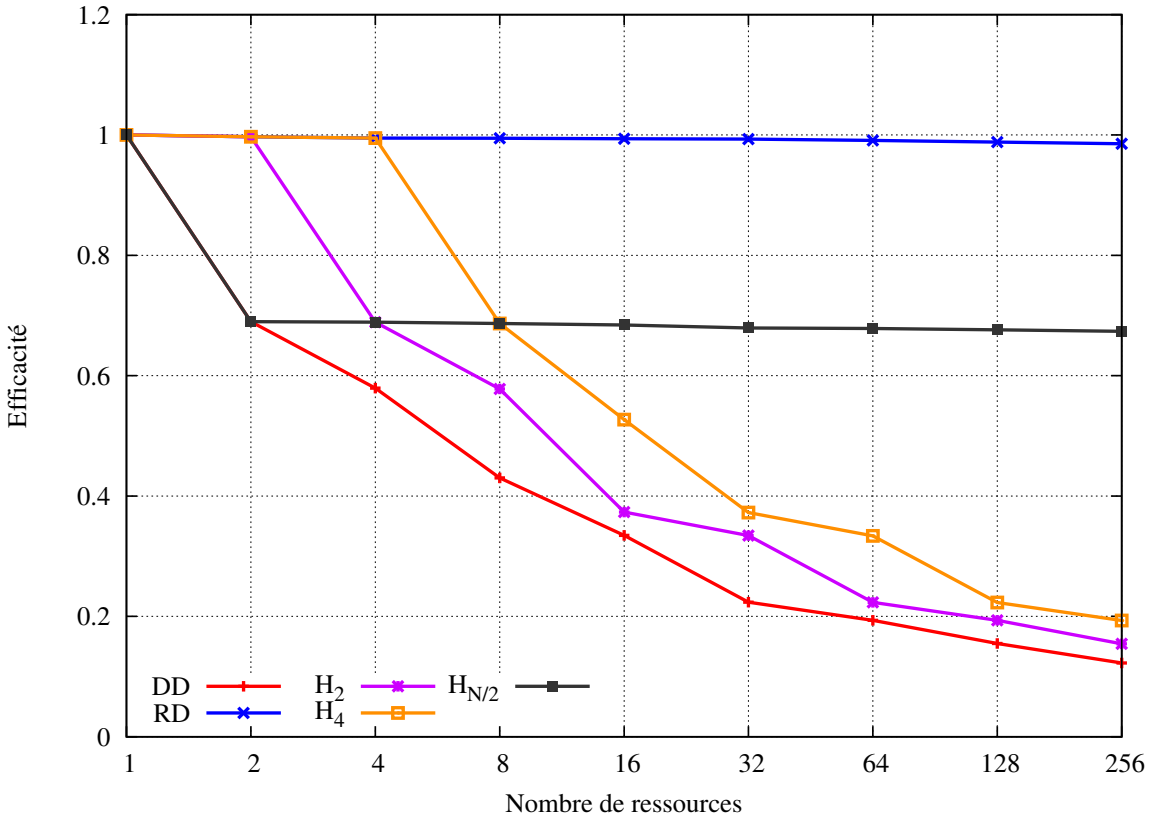


FIGURE 3.9 – Extensibilité faible pour le cas test hétérogène par insertion latérale (meilleur en haut).

ont montré de sérieuses limitations, obtenant des performances insuffisantes en efficacité parallèle. Ces méthodes ont même échoué à satisfaire la condition de non dépassement mémoire pour le cas test hétérogène par insertion centralisée.

3.4 Conclusion

Les méthodes de décomposition de domaine, de réplication de domaine et hybride sont des méthodes très utilisées en simulation numérique. Elles permettent d'exprimer du parallélisme de donnée et de calcul de manière simple. Appliquées au transport Monte-Carlo, ces méthodes ont montré de sérieuses limitations. En effet, alors que la méthode de réplication de domaine présente un surcoût calculatoire non négligeable et surtout une très conséquente surconsommation mémoire liée aux mailles, la méthode de décomposition de domaine offre une surconsommation mémoire et un surcoût calculatoire rédhibitoires. La méthode hybride, présentée comme le compromis entre les méthodes de décomposition de domaine et de réplication de domaine, ne permet pas d'atteindre les objectifs que nous avons fixés.

Les problèmes d'équilibrage de charge et de limitation de donnée sont orthogonaux, ce qui signifie que proposer une solution pour l'un des deux problèmes peut amener à dégrader la solution pour l'autre. Concrètement, le meilleur compromis de la méthode hybride possible ne permettra pas d'atteindre les objectifs que nous souhaitons atteindre. Cette méthode hybride offre néanmoins de nouvelles perspectives. En effet, le degré de réplication offre la possibilité de répartir la charge

Méthodes	DD	H_2	H_4	$H_{\frac{N}{2}}$	RD
Particules + Communications	-	-	**2530**	282	184
Mailles	-	-	11.7	**41.6**	**77.0**
Partitionnement(s)	-	-	0.51	0.25	0.0
Régulation (hors partitionnement)	-	-	0.0	0.0	0.0
Temps total	-	-	2542	324	261

TABLE 3.8 – Profilage (secondes) en extensibilité faible pour le cas hétérogène par insertion centralisée.

Méthodes	DD	H_2	H_4	$H_{\frac{N}{2}}$	RD
Particules + Communications	**2547**	**2019**	**1606**	432	239
Mailles	10.6	10.8	10.6	**22.9**	**76.1**
Partitionnement(s)	1.55	0.81	0.53	0.2	0.0
Régulation (hors partitionnement)	0.0	0.0	0.0	0.0	0.0
Temps total	2559	2030	1617	456	315

TABLE 3.9 – Profilage (secondes) en extensibilité faible pour le cas hétérogène par insertion latérale.

de calcul liées à une maille donnée sur plusieurs ressources de calcul. Cette possibilité est particulièrement intéressante quand un petit ensemble de mailles contient la majorité des particules. Pour que les mailles les plus chargées soient mieux réparties, nous devons tenir compte de la répartition des particules sur les mailles lors de la décomposition du domaine.

Chapitre 4

Régulation de charge par partitionnement de graphe

Le bilan du chapitre 3 est que les méthodes classiques de parallélisation n'ont pas atteint les objectifs de répartition de charge et de donnée. En effet, aucune de ces méthodes n'est parvenue à garantir à la fois une bonne distribution des charges et des données. Le problème de ces méthodes réside dans le fait qu'elle ne répartisse qu'une partie des données : soit les mailles, soit les particules. Nous avons vu au chapitre 3, section 3.2 que dans des configurations favorables, les méthodes de décomposition de domaine et hybride pouvaient atteindre les objectifs de régulation de charge et de donnée. Nous devons donc décomposer le domaine. La question est alors de savoir comment assurer la qualité de la répartition de la charge de calcul entre les sous-domaines dans toutes les configurations.

L'approche serait donc de prendre en compte la charge de calcul au moment de réaliser la décomposition de domaine. Nous avons alors deux critères de décomposition : le nombre de mailles et le nombre de particules. La décomposition du domaine selon ces deux critères est particulièrement complexe. Nous avons introduit au chapitre 2, les techniques de partitionnement (section 2.3) et en particulier les principaux outils utilisés. Nous avons vu que ces derniers se basaient sur des modèles abstraits, les graphes. Le graphe est un modèle abstrait particulièrement intéressant dans la mesure où sa structure simple permet d'exprimer des éléments complexes. C'est précisément cet aspect qui nous intéresse. En effet, nous pouvons simplifier l'expression des données à décomposer un utilisant une modélisation sous forme de graphe. Cette modélisation sera présentée en section 4.1.

La décomposition de domaine équilibrée en nombre de particules et en mailles peut être obtenue par un partitionnement du graphe. Nous proposerons alors en section 4.2 une méthode de régulation de charge par partitionnement. Une étude expérimentale sera présentée en section 4.3. Cette étude permettra de comparer cette approche par rapport aux méthodes classiques. Enfin, nous conclurons sur cette approche en apportant une analyse critique en section 4.4.

4.1 Transport de particules Monte-Carlo : modélisation sous forme de graphe

Nous définissons par $G = (S, A)$ le graphe muni d'un ensemble de sommets noté S et un ensemble d'arêtes noté A. Nous souhaitons exprimer les mailles et les particules à l'aide des sommets et des arêtes de ce graphe. L'objectif de cette modélisation est de représenter fidèlement

les mailles et les particules. Nous commencerons par présenter la modélisation des mailles en sous-section 4.1.1. Ensuite, nous intégrerons la représentation des particules dans ce modèle en sous-section 4.1.2. Nous terminerons par une technique d'optimisation de ce modèle en sous-section 4.1.3

4.1.1 Modélisation des mailles

Une maille est une donnée sur laquelle des traitements seront réalisés. Chaque maille possède des mailles voisines. Nous proposons alors de considérer un sommet du graphe comme la représentation d'une maille et de définir une arête comment étant l'expression du voisinage de deux mailles. Nous proposons alors de définir une arête $a = (s, s') \in A, s \in S, s' \in S$ lorsque s et s' représentent des mailles voisines entre elles. La relation de voisinage des mailles est bidirectionnelle. Par conséquent, nous considérons les arêtes comme étant non-orientées. La figure 4.1 présente la modélisation du maillage sous forme de graphe.

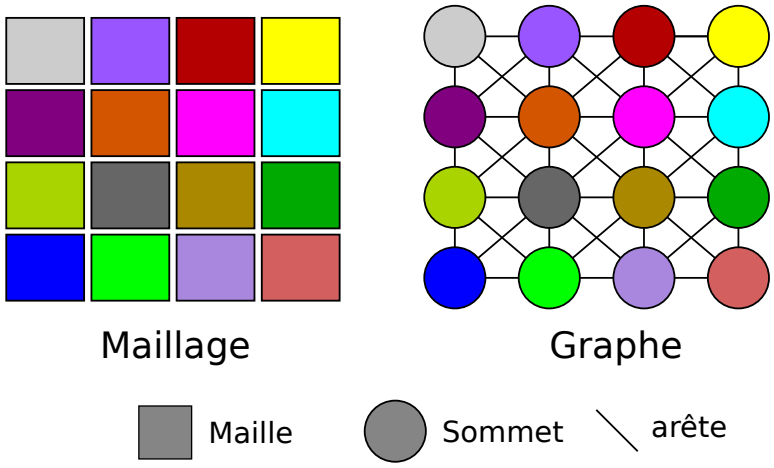


FIGURE 4.1 – Représentation du maillage sous forme d'un graphe.

4.1.2 Modélisation des particules

Les particules sont réparties entre les mailles. Il existe donc une relation d'appartenance entre une particule et une maille. Nous devons représenter cette dépendance de telle sorte qu'un sommet soit caractérisé par le nombre de particules qu'il représente. C'est pourquoi, nous avons choisi de définir la pondération d'un sommet en fonction du nombre de particules qu'il représente. ρ_m désigne le nombre de particules d'une maille notée m. Étant donné que la notation $m \in s$ signifie que la maille m est représentée par le sommet s, le poids w de chaque sommet est alors calculé selon l'équation 4.1.

$$\forall s \in S \;\; w(s) = \rho_m, m \in s \; . \tag{4.1}$$

Le nombre de particules pouvant transiter entre deux mailles dépend du nombre de particules de ces mailles. Les arêtes du graphes ont pour but d'exprimer une transition entre deux sommets. Ces transitions peuvent également être pondérées. Dans notre cas, nous proposons d'utiliser cette pondération pour exprimer le volume maximum de particules pouvant transiter entre deux mailles. Nous avons pondéré chaque sommet en fonction du nombre de particules de la maille

qu'il représente. En considérant $s' \in S$ et $s'' \in S$ deux sommets adjacents, le poids de chaque arête est alors calculé selon l'équation 4.2. Le poids d'une arête est donc la somme des poids des deux sommets.

$$\forall a = (s', s'') \in A \;\; w(a) = w(s') + w(s'') \; . \tag{4.2}$$

La figure 4.2 donne un exemple de pondération du graphe. Nous avons à gauche une représentation du maillage avec au centre de chaque maille le nombre de particules associées à la maille. Nous avons à droite le graphe correspondant dont le poids de chaque sommet est noté au centre du sommet. Le poids de chaque arête est indiqué à côté de l'arête.

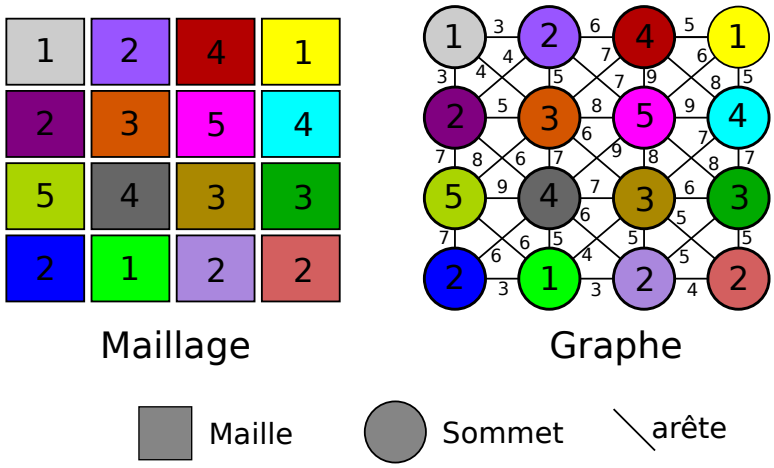


FIGURE 4.2 – Exemple de pondération du graphe.

Le modèle de graphe permet une représentation des charges de calcul et des données de simulation. Notons que la complexité de création du graphe est $\mathcal{O}(n)$ où n est le nombre de sommets. Cette complexité est optimale dans la mesure où nous devons pondérer chaque sommet et chaque arête. Dans notre cas, le nombre de sommets correspond au nombre de mailles. En considérant nos cas test (voir sous-section 2.1.6), la création du graphe concerne donc un nombre de sommets de l'ordre du million. Nous aurions alors tout intérêt à réduire la taille du graphe. Une méthode de contraction de graphe consiste à exprimer le graphe sous forme d'un autre graphe possédant moins de sommets et d'arêtes. Une réduction du temps de création du graphe peut alors être obtenue par une méthode de contraction. De plus, un graphe de plus petite taille nécessite moins de mémoire.

4.1.3 Contraction de graphe

La contraction du graphe permet de réduire le nombre de sommets. Cette réduction permet de réduire le temps de création du graphe. De manière générale, les algorithmes de graphes, ayant une complexité en $\mathcal{O}(n)$ ou plus, ont un temps de restitution qui diminue lorsque n diminue. La figure 4.3 présente un exemple d'algorithme de partitionnement dont le temps de restitution est fonction du nombre de sommets. L'outil de partitionnement utilisé est *MeTiS*[17] et le processeur utilisé est un Intel(R) Xeon(R) CPU E5530 cadencé à 2.40GHz.

Dans le but de contracter le graphe, nous proposons donc de définir un sommet du graphe comme étant la représentation d'un groupe de mailles voisines entre elles. Nous définissons alors r le nombre de mailles par sommet. La taille du graphe est alors obtenu au moyen de l'équation 4.3.

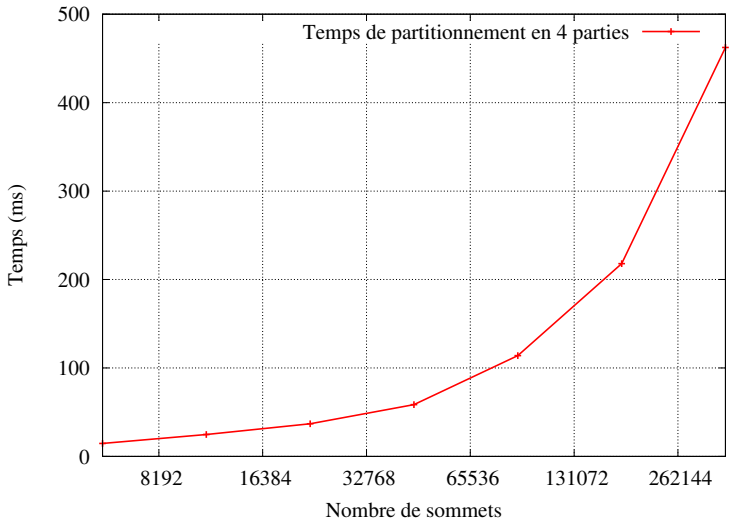


FIGURE 4.3 – Temps de partitionnement d'un graphe en 4 parties selon le nombre de sommets.

$$\#S = \left\lceil \frac{M}{r} \right\rceil . \tag{4.3}$$

Dans le graphe contracté, nous proposons de définir une arête $a = (s, s') \in A, s \in S, s' \in S$ lorsque $\exists m, m', m \in s, m' \in s'$ telles que m et m' sont des mailles voisines entre elles.

La contraction du graphe doit prendre en compte les pondérations des sommets et des arêtes. Nous définissons la pondération des sommets du graphe contracté selon l'équation 4.4. Le principe est le même que le graphe précédent, il s'agit simplement de sommer le nombre de particules de toutes les mailles représentées par chacun des sommets. Nous proposons de conserver la méthode de pondération des arêtes que nous avons définie au moyen de l'équation 4.2. Un exemple de graphe contracté est représenté en figure 4.4.

$$\forall s \in S \;\; w(s) = \sum \rho_m, \; \forall m \in s \; . \tag{4.4}$$

Nous obtenons alors un graphe aux sommets et aux arêtes pondérés permettant de modéliser les mailles et les particules. Intuitivement, décomposer ce graphe en sous-graphes permet de décomposer les mailles et les particules. Le partitionnement d'un tel graphe permet alors de répartir équitablement les mailles et les particules. Nous proposons donc de nous appuyer sur la technique de partitionnement de graphe pour réaliser de la régulation de charge et de donnée.

4.2 Régulation de charge et de donnée par partitionnement de graphe

Nous avons défini en section 2.3, les techniques de partitionnement. Le partitionnement d'un graphe permet de décomposer le graphe en parties ayant le même nombre de sommets. La coupe, déterminée par les poids des arêtes coupées, est minimisée.

Nous avons défini le poids des arêtes comme étant le nombre maximal de particules pouvant transiter. Cette pondération est adaptée au critère de minimisation de la coupe, dans la mesure où plus la coupe sera faible, moins de particules transiteront entre les sous-graphes et donc moins il y aura de communication. En revanche, le partitionnement, ne prenant en compte que le

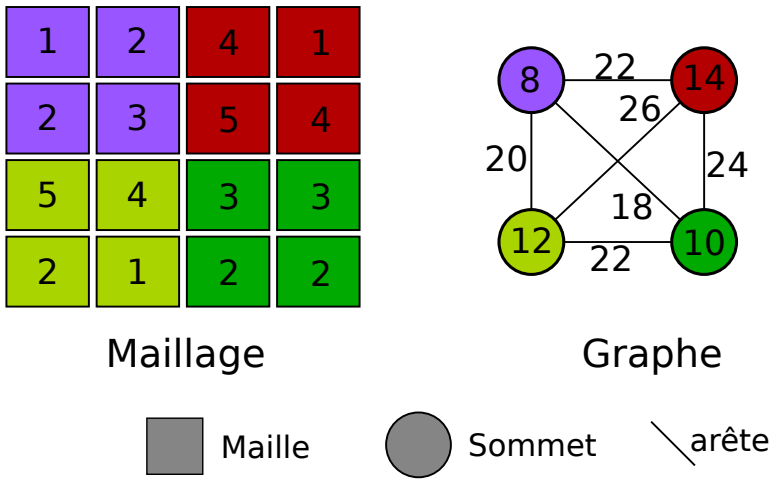


FIGURE 4.4 – Exemple de graphe contracté.

nombre de sommets par sous-graphe, n'est pas adapté à notre problématique. En effet, nous devons prendre en compte la répartition de la charge de calcul. Nous avons défini une pondération des sommets permettant de quantifier la charge de calcul représentée par chaque sommet. Nous devons alors considérer deux critères : le nombre de sommets par sous-graphe et la répartition des poids entre les sous-graphes. Nous avons présenté le principe du partitionnement multi-critères en sous-section 2.3.3. Le partitionnement multi-critères permet de résoudre cette problématique. *MeTiS* [17] permettant le partitionnement multi-critères, nous avons donc utilisé cet outil.

Le partitionnement du graphe nous permet, pour une itération donnée, d'être équilibré en particules et en mailles. Cependant, du fait de la migration des particules, des déséquilibres de charge et de donnée peuvent apparaître quelques itérations plus tard. Pour palier ce problème, nous proposons une méthode permettant d'invoquer le partitionnement dans le cas où des déséquilibres commenceraient à apparaître.

4.2.1 Algorithme général

L'algorithme 4 présente les étapes principales de la simulation lorsque l'approche par partitionnement est utilisée. Nous commençons par la phase de calcul du transport Monte-Carlo (ligne 2). Ensuite, nous devons échanger les particules transitant d'un sous-domaine à l'autre (ligne 3). L'évaluation des déséquilibres de charge et données est réalisée à ligne 4. La fonction $Evaluer\_Desequilibre(\ Maillage,\ Particules\ )$ renvoie vrai en cas de déséquilibre. Les critères de déséquilibre seront présentés en sous-section 4.2.2. En cas de déséquilibre, le partitionnement du graphe est invoqué (ligne 6). Les étapes du partitionnement du graphe seront détaillées en sous-section 4.2.3.

4.2.2 Critère de partitionnement

Soit N le nombre de ressources, M le nombre de mailles et P le nombre de particules à une itération donnée. Nous définissons $\tau_p, \tau_p \in [0, 1]$, le ratio de déséquilibre en nombre de particules à partir duquel nous estimons qu'il est avantageux de partitionner. Nous obtenons alors P_{max}, le nombre maximal toléré de particules pour une ressource donnée, au moyen de l'équation 4.5. Étant donné P_k, le nombre de particules de la ressource notée k, la condition de partitionnement sur la

Algorithme 4 : Algorithme général de la régulation par partitionnement.

```
   /* Tant que la condition de fin de simulation n'est pas atteinte          */
1  tant que !fin_application faire
       /* Phase de Transport                                                 */
2      Monte_carlo_transport( Maillage, Particules )
       /* Communications particules changeant de sous-domaine                */
3      Communications_Particules( Maillage, Particules )
       /* Déséquilibre de charge ou de donnée ? (voir sous-section 4.2.2)    */
4      desequilibre ← Evaluer_Desequilibre( Maillage, Particules )
       /* Partitionnement du graphe si déséquilibre de charge                */
5      si desequilibre = Vrai alors
           /* Invocation d'un partitionnement (voir sous-section 4.2.3)      */
6          Partitionnement_graphe( Graphe , Maillage, Particules )
7      fin
8  fin
```

répartition des particules est alors définie par l'équation 4.6. Le partitionnement est donc invoqué si au moins une ressource a plus de particules que la limite P_{max}.

$$P_{max} = (1 + \tau_p)\frac{P}{N} \, . \tag{4.5}$$

$$\exists k \in \{0, 1, ..., N-1\}, P_k > P_{max} \, . \tag{4.6}$$

Nous devons également limiter le nombre de mailles par ressource. Dans le but d'améliorer la répartition de la charge de calcul, nous proposons d'utiliser le principe du degré de réplication introduit en sous-section 2.2.4. Nous définissons $\tau_m, \tau_m \in [0, 1]$, le ratio de déséquilibre en nombre de mailles à partir duquel nous estimons qu'il est avantageux de partitionner. Nous obtenons alors M_{max}, le nombre maximal toléré de mailles pour une ressource donnée, au moyen de l'équation 4.7. Étant donné M_k, le nombre de mailles de la ressource notée k, la condition de partitionnement sur la répartition des mailles est alors définie par l'équation 4.8. Le partitionnement est donc invoqué si au moins une ressource a plus de mailles que la limite M_{max}.

$$M_{max} = (1 + \tau_m)(R\frac{M}{N}) \, . \tag{4.7}$$

$$\exists k \in \{0, 1, ..., N-1\}, M_k > M_{max} \, . \tag{4.8}$$

Pour tester ces conditions, nous devons avoir connaissance du nombre de particules et de mailles de chaque ressource. L'évaluation des conditions de partitionnement implique donc une communication collective.

Nous venons de présenter les conditions de l'invocation d'un partitionnement. Nous proposons de discuter, dans la sous-section suivante, du partitionnement du graphe.

4.2.3 Partitionnement du graphe

Lorsque au moins une des deux conditions présentées en sous-section 4.2.2 est satisfaite, un partitionnement est invoqué. Nous allons présenter la méthode utilisée.

Création du graphe et partitionnement

Nous avons proposé une modélisation sous forme de graphe (voir section 4.1). La première étape consiste donc à construire ce graphe. Nous avons fait le choix de ne pas utiliser les fonctions de repartitionnement de graphe, permettant de minimiser les migrations de donnée, de l'outil *ParMeTiS* [27]. Notre graphe est centralisé, il représente l'ensemble des mailles et l'ensemble des particules de toutes les ressources confondues. Pour construire ce graphe, chaque ressource met à jour son sous-graphe en pondérant les sommets en fonction de la répartition de ses propres particules. Ensuite, une opération de réduction est réalisée consistant à sommer les poids des sommets entre les ressources. Ainsi, le poids d'un sommet correspond au nombre de particules sur les mailles qu'il représente de toutes les ressources confondues. Enfin, les arêtes du graphe sont construites. Le partitionnement peut alors être invoqué.

Le partitionnement est réalisé selon deux critères. Le premier concerne le nombre de sommets par sous-graphe. Le second concerne le nombre de particules par sous-graphe, ce qui, dans notre cas, correspond à la somme des poids des sommets par sous-graphe. Nous proposons de définir $\tau'_m, \tau'_m \leq \tau_m$ la tolérance accordée au partitionnement par rapport au nombre de sommets par sous-graphe. Nous définissons également $\tau'_p, \tau'_p \leq \tau_p$ la tolérance accordée au partitionnement par rapport au nombre de particules par sous-graphe. Nous créons $\frac{N}{R}$ sous-graphes.

Nous obtenons une nouvelle partition. L'ensemble des ressources effectue le partitionnement en simultané. Les partitions obtenues sont strictement identiques sur l'ensemble des ressources. Ce procédé permet d'éviter qu'une ressource communique à toutes les autres le partitionnement obtenu. La dernière étape consiste à répartir les ressources entre les sous-graphes puis à échanger les mailles et les particules pour tenir compte de cette nouvelle partition. L'algorithme 5 présente le résumé de l'algorithme de partitionnement que nous venons de décrire. Nous avons fait le choix de ne pas utiliser les fonctions de repartitionnement afin de donner la priorité à la qualité des partitions. Nous devons, tout de même, minimiser autant que possible le nombre de mailles et de particules à échanger. Nous proposons donc une méthode permettant d'optimiser la répartition des ressources entre les sous-graphes (ligne 4 de l'algorithme 5).

Algorithme 5 : Algorithme du partitionnement.

```
   /* Mise à jour du sous-graphe en fonctions des données locales à la
    ressource                                                          */
1  Mise_a_jour_sous_graphe( Graphe, Maillage, Particules )
   /* Mise à jour du graphe avec les fonctions de pondérations        */
2  Mise_a_jour_graphe( Graphe )
   /* Appel à l'outil de partitionnement                              */
3  Partitionnement( Graphe, τ'_m, τ'_p)
   /* Appel à l'outil de partitionnement                              */
4  Assignation_sous_graphe( Partition, Partition_precedente)
   /* Envoi et réceptions des mailles et des particules               */
5  Echanges_mailles_particules( Graphe, Maillage, Particules )
```

Minimisation des migrations de donnée

Les migrations de donnée impliquent des communications. Nous devons minimiser le nombre de mailles et de particules devant être échangées. C'est pourquoi nous proposons d'attribuer les sous-graphes en fonction du partitionnement précédent. Pour cela, nous proposons de minimiser la distance entre les barycentres des sous-graphes issus du nouveau partitionnement et les bary-

centres des sous-graphes issus de l'ancien partitionnement. Rappelons que le calcul d'un barycentre consiste en une moyenne pondéréee de plusieurs éléments. Dans notre cas, nous considérons un élément comme étant les coordonnées d'un sommet $s \in S$. Nous considérons tous les éléments avec le même poids. Étant donné $n = \frac{N}{R}$, Nous obtenons alors $\overline{\underline{x}}_{i,n}$ le barycentre de S_i l'ensemble des sommets du sous-graphe i. Nous définissons $\overline{\underline{x}}'_{j,n}$ le barycentre de S_j l'ensemble des sommets du sous-graphe j lors du partitionnement précédent.

L'algorithme 6 présente la méthode utilisée pour répartir les sous-graphes entre les ressources. Nous souhaitons estimer, pour chaque sous-graphe i issu du partitionnement que nous venons d'effectuer, quel sous-graphe j issu du précédent partitionnement est le plus proche. Nous affecterons alors l'ensemble des ressources, précédemment affectées au sous-graphe j, au sous-graphe i. Pour cela, nous parcourons l'ensemble des sous-graphes issus du nouveau partitionnement (ligne 1). Nous choisissons alors le sous-graphe k, issu du précédent partitionnement, telle que la distance entre $\overline{\underline{x}}'_{k,n}$ et $\overline{\underline{x}}_{i,n}$ soit la plus faible possible. Nous affectons enfin l'ensemble des ressources, précédemment affectées au sous-graphe k, au sous-graphe i (ligne 9). Notons que l'algorithme d'affectation a une complexité en $\mathcal{O}(n^2)$.

Algorithme 6 : Assignation des ressources aux sous-graphes.

```
   /* Pour chaque sous-graphe issue du récent partitionnement          */
1  pour chaque i ∈ [0, n[ faire
2      D_{i,min} ← D_max
       /* Recherche du sous-graphe, issu du précédent partitionnement, le plus
        proche                                                          */
3      pour chaque j ∈ [0, n[ faire
4          si D(x̲_{i,n}, x̲'_{j,n}) ≤ D_{i,min} alors
5              k ← j
6              D_{i,min} ← D(x̲_{i,n}, x̲'_{j,n})
7          fin
8      fin
9      Assigner_sous_graphe(i, k)
10 fin
```

4.2.4 Optimisations

Le partitionnement du graphe a un coût. Premièrement, le coût du partitionnement proprement dit dépend du nombre de sommets (voir figure 4.3). La contraction du graphe que nous avons proposé permet de réduire le nombre de sommets et donc le coût de partitionnement. Le coût de partitionnement est également fonction du nombre de sous-graphes. L'utilisation du degré de réplication permet de réduire le nombre de sous-graphe d'un facteur R.

Le partitionnement du graphe entraîne la redistribution des mailles et des particules impliquant des communications. De plus, la création du graphe étant centralisée. Celle-ci implique alors la synchronisation de toutes les ressources et l'échange des informations concernant la répartition des particules. Nous devons également minimiser ce coût. Le partitionnement du graphe doit être suffisamment fréquent pour garantir une bonne répartition de la charge de calcul et des données. Toutefois, sachant que le partitionnement a un coût, partitionner de manière excessive entraînera un surcoût trop conséquent. De plus, l'évaluation des conditions de partitionnement nécessite de connaître le nombre de particules et de mailles de chaque ressource, impliquant une communication collective. Nous devons éviter de tester ces conditions trop régulièrement. C'est pourquoi, nous proposons d'évaluer que toutes les X itérations si un partitionnement s'impose.

Espacer les partitionnements d'au moins X itérations permet de limiter le nombre de partitionnements. Nous pouvons également retarder autant que possible le premier partitionnement. Nous avons fixé comme contrainte de ne jamais dépasser le nombre de mailles limite noté M_{max} (voir sous-section 4.2.2). À l'initialisation, il est tout à fait possible que le nombre de mailles chargées en particules, noté M_{src}, soit très inférieur à M_{max}. Par conséquent, si M_{src} est suffisamment petit, nous proposons de dupliquer les mailles sources sur l'ensemble des ressources de calcul. Dans le cas contraire, le partitionnement est invoqué. En généralisant ce concept, nous proposons d'invoquer le premier partitionnement du graphe seulement lorsque le nombre de mailles possédant des particules devient trop grand. La condition de partitionnement sur les mailles est redéfinie pour le cas du premier partitionnement, en fixant $\tau_m = 0$.

L'algorithme 7 présente l'algorithme optimisé de régulation par partitionnement de graphe. À l'initialisation, nous commençons par regarder si le nombre de mailles sources n'est pas trop grand pour que nous puissions les dupliquer sur toutes les ressources de calcul (ligne 2). Dans le cas où le nombre de mailles sources est inférieur à la limite fixée, toutes les ressources initialisent le maillage et les particules en utilisant le principe de la réplication de domaine (ligne 7). Les mailles sources sont initialisées par toutes les ressources et les particules sont réparties équitablement entre les ressources. En revanche, si le nombre de mailles sources est supérieur à la limite, nous devons partitionner (ligne 3).

Nous retrouvons en ligne 10 la condition de la boucle principale du transport Monte-Carlo. Chaque itération de boucle correspond à une itération du transport Monte-Carlo. Nous commençons par la phase de calcul du transport Monte-Carlo en ligne 11. Ensuite, si le partitionnement a déjà été effectué, nous devons échanger les particules transitant d'un sous-domaine à l'autre (ligne 13). Les conditions de partitionnement sont évaluées seulement toutes les X itérations (ligne 15). Afin de s'assurer que cette évaluation a bien lieue toutes les X itérations, nous utilisons un compteur d'itérations (lignes 1 et 22). Dans l'hypothèse où une condition de partitionnement au moins est atteinte, nous devons partitionner (ligne 19).

Pour illustrer cet algorithme, nous allons présenter l'évolution de la répartition des mailles et des particules sur un exemple. Nous présenterons alors le nombre de particules par maille pour l'ensemble des ressources.

4.2.5 Exemple

Soit $N = 8$, $X = 20$, $\tau'_m = 0.25$, $\tau'_p = 0.05$ et $R = 1$. Dans ce cas, $n = N$. Nous proposons de suivre l'évolution de la répartition des particules des 8 ressources pendant la durée d'une simulation. Le cas test choisi est le cas test hétérogène avec insertion centralisée particulièrement intéressant pour ses caractéristiques de déséquilibre. Dans cet exemple l'insertion des particules est réalisée exactement au milieu du domaine. La légende des couleurs associées aux nombre de particules par maille est donnée en figure 4.5. Notons que les mailles n'ayant pas de particules seront représentées en noires. Nous représenterons l'ensemble des mailles pour toutes les ressources.

La figure 4.6 présente la répartition des particules avant le premier partitionnement. Nous constatons que les ressources possèdent les mêmes mailles et se partagent équitablement les particules.

La figure 4.7 présente la répartition des particules après le premier partitionnement. Nous constatons que les ressources possèdent chacune leurs propres mailles. Le partage équitable des particules est respecté.

La figure 4.8 présente la répartition des particules après un second partitionnement ayant lieu quelques itérations après le premier. Nous constatons que la nouvelle partition est plutôt proche

Algorithme 7 : Algorithme optimisé de la méthode par partitionnement.

```
1  iter_courante ← 0
   /* Nombre de mailles sources supérieur à la limite ?                    */
2  si M_src > M_max alors
       /* Invocation d'un partitionnement                                  */
3      Partitionnement_graphe( Graphe , Maillage, Particules )
4      partitionnement_fait = Vrai
5  fin
6  sinon
7      Initialisation_avec_replication( Maillage, Particules )
8      partitionnement_fait = Faux
9  fin
   /* Tant que la condition de fin de simulation n'est pas atteinte        */
10 tant que !fin_application faire
       /* Phase de Transport                                               */
11     Monte_carlo_transport( Maillage, Particules )
12     si partitionnement_fait = Vrai alors
           /* Si le premier partitionnement a été réalisé                  */
           /* Communications particules changeant de sous-domaine          */
13         Communications_Particules( Maillage, Particules )
14     fin
       /* Toutes les X itérations, analyse de la répartition des particules
          et des mailles                                                   */
15     si iter_courante ≡ 0 (mod X) alors
           /* Déséquilibre de charge ou de donnée ?                        */
16         desequilibre ← Evaluer_Desequilibre( Maillage, Particules )
17     fin
       /* Partitionnement si au moins une des deux conditions est satisfaite
          */
18     si desequilibre = Vrai alors
           /* Invocation d'un partitionnement                              */
19         Partitionnement_graphe( Graphe , Maillage, Particules )
20         partitionnement_fait = Vrai
21     fin
22     iter_courante ← iter_courante + 1
23 fin
```

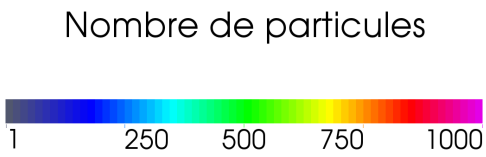


FIGURE 4.5 – Légende du nombre de particules par maille.

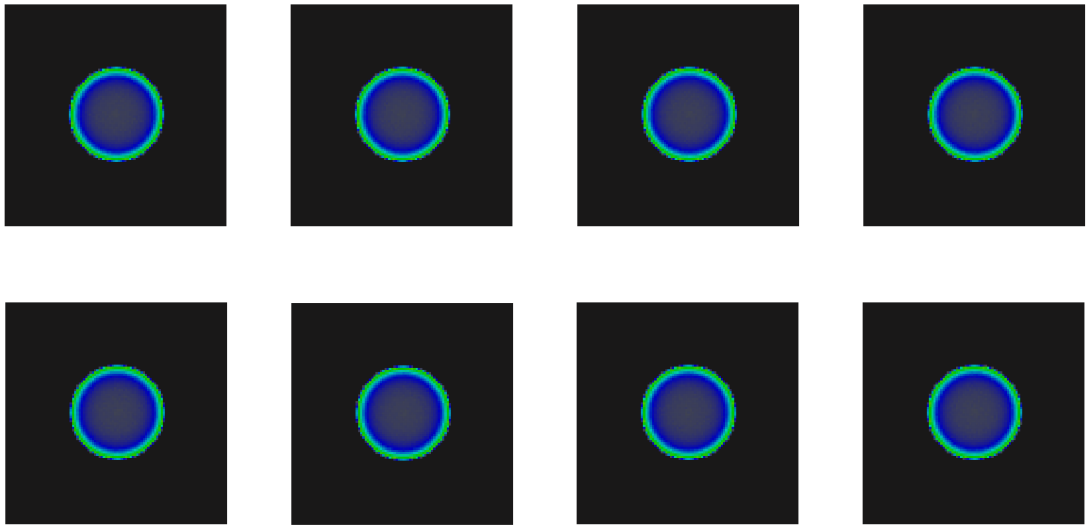


FIGURE 4.6 – Répartition des particules entre les ressources avant l'invocation du premier partitionnement.

de la précédente et que l'affectation des ressources sur les nouvelles parties a correctement pris en compte le partitionnement précédent.

La figure 4.9 présente la répartition des particules en fin de simulation. Les particules se sont dispersées dans tout le domaine. Toutes les mailles ayant des particules, nous pouvons donc déterminer la partition générée par l'outil de partitionnement. Nous observons que les parties n'ont pas toutes la même taille. Cela est dû à la tolérance de 25 % que nous avons donné à l'outil de partitionnement par rapport au nombre de sommets par partie. Nous constatons que certaines parties ne forment pas un seul et même ensemble de mailles, ce qui s'explique par la contrainte sur la répartition des particules. En effet, au risque de dégrader la coupe, l'outil de partitionnement essaie de respecter les tolérances, appliquées au partitionnement, passées en paramètres.

L'algorithme que nous venons de présenter et d'illustrer avec cet exemple, est largement dépendant de plusieurs paramètres. En particulier, il dépend du seuil de déséquilibre en nombre de particules, noté τ_p, et du nombre d'itérations minimal entre deux partitionnements, noté X. Le paramétrage de cette méthode n'est pas chose aisée. Nous proposons donc de discuter, dans la

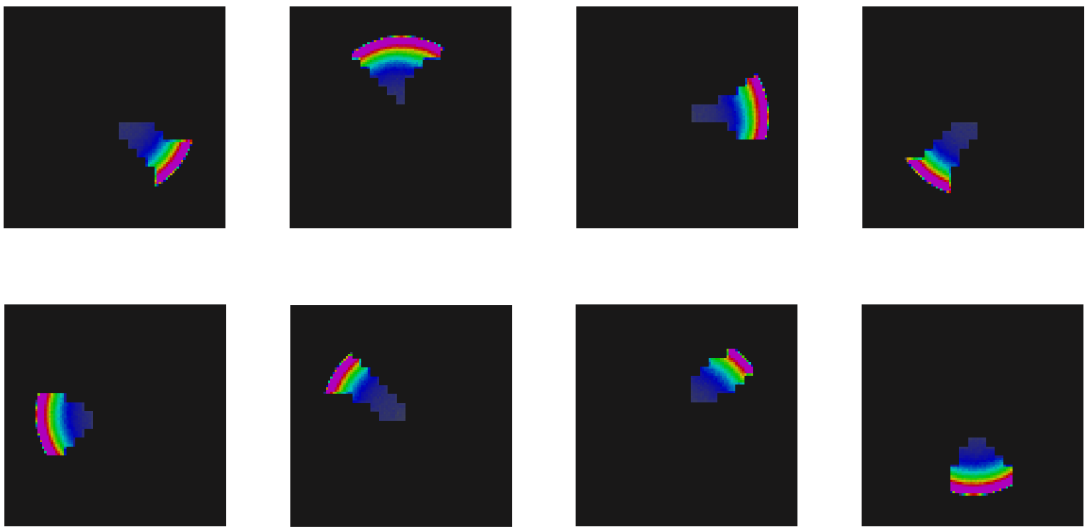


FIGURE 4.7 – Répartition des particules entre les ressources après l'invocation du premier partitionnement.

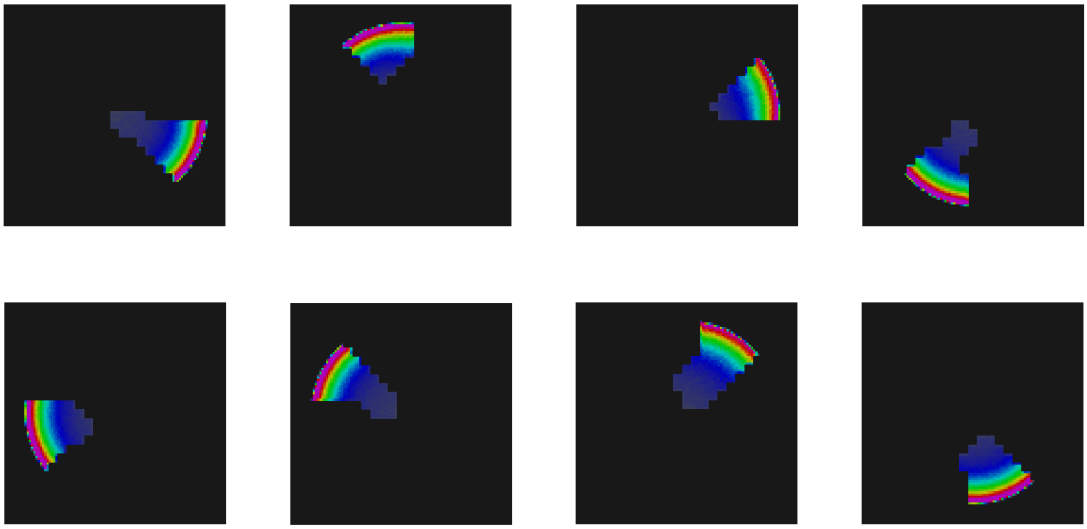


FIGURE 4.8 – Répartition des particules entre les ressources après l'invocation du deuxième partitionnement.

sous-section suivante, l'impact des paramètres.

4.2.6 Paramétrage

La contraction du graphe que nous avons proposé permet de réduire le nombre de sommets et donc le coût de partitionnement. Cependant, nous devons définir un graphe ayant suffisamment de sommets à partitionner. Nous définissons $\#S_{min}$, le nombre minimal de sommets par sous-graphe,

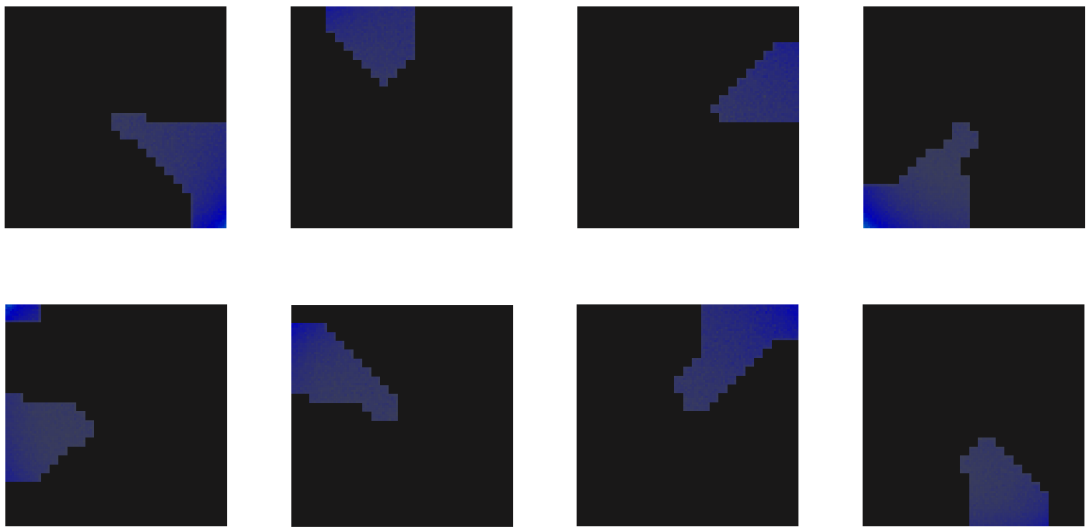


FIGURE 4.9 – Répartition des particules entre les ressources en fin de simulation.

tel qu'il existe un partitionnement acceptable. Nous choisissons alors r le nombre de mailles par sommet selon les conditions de l'équation 4.9.

$$\exists r, \forall s \in S \; \exists \{m_0, m_1, ..., m_r\} \in s, \#S > \#S_{min} N \; . \tag{4.9}$$

Le seuil de déséquilibre, noté τ_p, définit le déséquilibre maximal en nombre de particules que nous considérons comme étant acceptable. Choisir un τ_p élevé n'a pas de sens. De plus, cette valeur est un seuil. En effet, le choix de τ_p ne nous garantit absolument pas que le déséquilibre en nombre de particules sera toujours inférieur à τ_p, mais simplement qu'au delà de cette valeur nous invoquerons le partitionnement. Le choix de X, le nombre d'itérations minimal entre chaque partitionnement, est complémentaire du choix de τ_p. Par exemple, choisir une valeur de X très élevée et une valeur de τ_p très faible serait incohérent pour un cas test très déséquilibré. En effet, il serait fort possible que le seuil de déséquilibre soit régulièrement largement dépassé. Il faut donc choisir X en fonction du déséquilibre maximal que nous ne souhaitons pas dépasser et du cas test.

Le paramétrage de cette approche n'est pas trivial. Il est par ailleurs assez difficile de trouver une configuration optimale pour tous les cas test. Nous avons étudié expérimentalement l'influence de certains de cas paramètres. L'étude expérimentale sera présentée en section suivante.

4.3 Étude expérimentale

L'approche par partitionnement que nous avons proposée, permet de réguler la charge de calcul et les données à chaque partitionnement. La méthode par partitionnement réalise un partitionnement toutes les X itérations au maximum. Nous avons choisi $R = 4$ car nous avons besoin d'un degré de réplication suffisamment élevé pour permettre une régulation de charge efficace. Nous avons également besoin d'utiliser un degré de réplication constant puisqu'un degré de réplication dépendant de N impliquerait une surconsommation mémoire trop importante. La valeur de X est un critère important, dans la mesure où X conditionne largement la qualité de la distribution des charges de calcul et des données ainsi que le coût de la méthode. Nous avons donc choisi de tester

cette méthode avec plusieurs valeurs de X. Nous avons choisi comme valeurs 5, 20, 50 et ∞. La notation ∞ signifie qu'un seul partitionnement au maximum sera réalisé. Nous avons comparé la méthode par partitionnement avec les méthodes de réplication de domaine et hybride avec degré de réplication constant. Nous notons $H_R P_X$ la méthode par partitionnement. Les partitionnements du graphe de la méthode $H_R P_X$ sont réalisés avec une tolérance de 25 % par rapport au nombre de sommets par sous-graphe (τ'_m) et de 5 % par rapport au nombre de particules par sous-graphe (τ'_p) . Nous avons choisi $\tau_p = 0.25$ et $\tau_m = 0.25$. La méthodologie d'expérimentation utilisée est celle que nous avons présentée en section 2.4.

4.3.1 Étude de régulation de charge et de donnée

Nous avons observé, au chapitre 3, que la méthode H_R est très impactée par des déséquilibres en nombre de particules. La méthode $H_R P_X$ a pour objectif de répondre aux limitations de la méthode H_R en permettant une décomposition, sous critères de charge, du domaine. Nous avons donc réalisé une étude expérimentale comparative de répartition de charge de calcul et de donnée sur les cas test présentés en sous-section 2.1.6. Nous allons présenter les résultats en déséquilibre de charge (voir l'équation 2.6) et en surconsommation mémoire (voir l'équation 2.7). Nous avons inséré pour chacun des cas test 128 millions de particules. Nous avons réalisé cette étude comparative pour 32 et 256 ressources de calcul. Nous proposons, dans un premier temps, d'évaluer l'impact de la méthode de partitionnement sur la répartition de la charge.

Étude de régulation de charge

La figure 4.10 présente les résultats obtenus sur le cas test homogène pour 32 et 256 ressources. Nous observons que la méthode $H_R P_X$ est parfaitement équilibrée durant les premières itérations. Cela correspond aux itérations précédant le premier partitionnement. En augmentant le nombre de ressources, on diminue le nombre maximal de mailles, le partitionnement arrive donc plus tôt. Après le partitionnement, des déséquilibres apparaissent. Nous constatons que la méthode $H_4 P_X$ parvient à maintenir une qualité de répartition de la charge à un bon niveau excepté pour la configuration $X = \infty$ avec 32 ressources. En effet, la méthode $H_4 P_\infty$ ne réalise qu'un seul partitionnement et le déséquilibre de charge augmente durant les itérations suivant le partitionnement. Nous observons toutefois que celui-ci reste modéré, il est en effet le plus souvent inférieur à celui de la méthode H_4. La qualité de répartition de la méthode $H_4 P_X$ augmente quand X diminue. En effet, les partitionnements sont plus fréquents.

Avec 256 ressources, nous observons que le déséquilibre de charge de la méthode $H_4 P_X$ s'est largement accentué. En particulier, nous constatons que la méthode $H_4 P_\infty$ devient rapidement plus déséquilibrée que la méthode H_4 et le reste jusqu'à la fin de la simulation. Par ailleurs, l'impact du choix de X est clairement visible tant les écarts en termes de déséquilibre sont importants. En effet, par exemple, la méthode $H_4 P_{50}$ est entre 2 et 5 fois plus déséquilibrée que la méthode $H_4 P_{20}$. Nous observons, à la lecture de la courbe, une fréquence de partitionnement de la méthode $H_4 P_5$ extrêmement élevée. En effet, chaque pic vers le bas correspond à un partitionnement.

La figure 4.11 présente les résultats obtenus sur le cas test hétérogène par insertion centralisée pour 32 et 256 ressources. Nous observons des résultats assez proches de ceux du cas test précédent. Nous observons cependant que le déséquilibre de la méthode $H_4 P_\infty$ est plus élevé par rapport au cas test homogène avec 32 ressources. Nous constatons même que le déséquilibre de charge obtenu par la méthode $H_4 P_\infty$ est plus élevé que celui obtenu par la méthode H_4 à partir de l'itération 400, que ce soit avec 32 ou 256 ressources. Cela illustre bien la non pérennité du partitionnement.

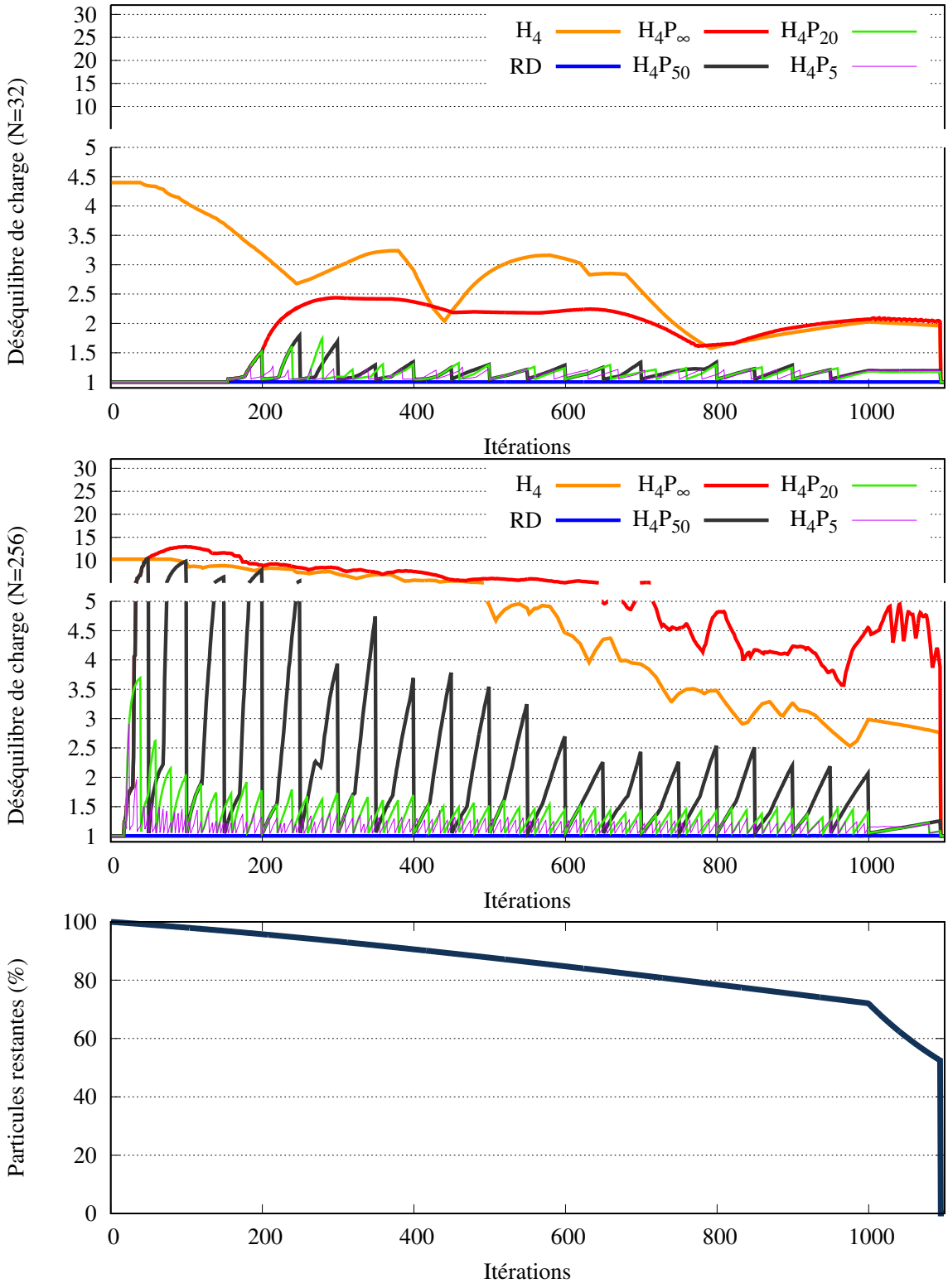


FIGURE 4.10 – Étude du déséquilibre de charge pour le cas test homogène (voir figure 2.3).

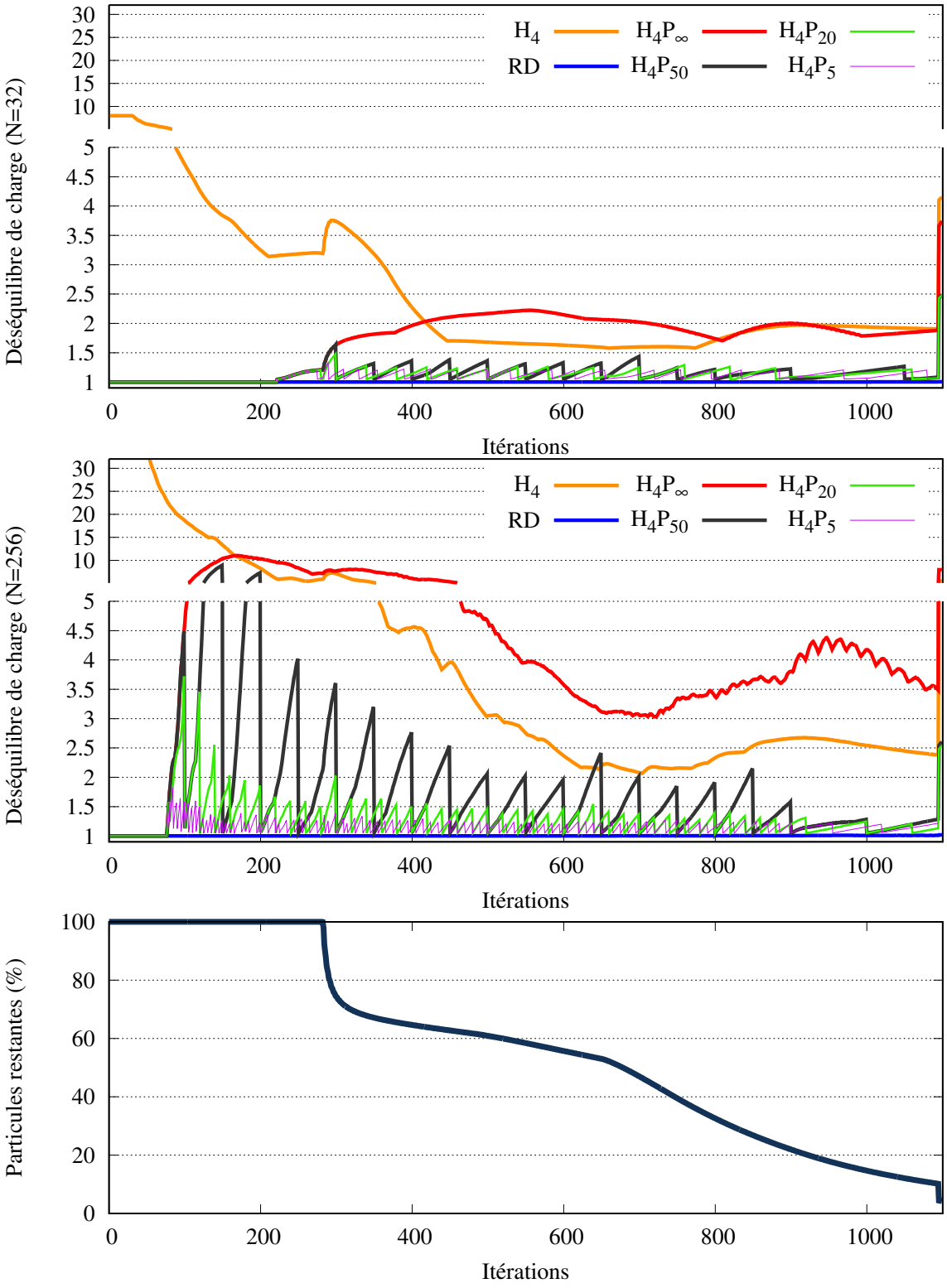


FIGURE 4.11 – Étude du déséquilibre de charge pour le cas test hétérogène par insertion centralisée (voir figure 2.4).

N	Méthodes	H_4	H_4P_∞	H_4P_{50}	H_4P_{20}	H_4P_5	RD
32	Particules	**4.4**	2.28	1.69	1.64	1.22	1.0
	Mailles	**4.2**	**4.8**	**4.8**	**4.8**	**4.8**	**26.0**
256	Particules	**10.2**	**12.7**	**10.3**	**3.67**	2.89	1.0
	Mailles	**4.2**	**5.58**	**4.81**	**4.84**	**4.84**	**74.5**

TABLE 4.1 – Surconsommation mémoire pour le cas homogène.

N	Méthodes	H_4	H_4P_∞	H_4P_{50}	H_4P_{20}	H_4P_5	RD
32	Particules	**8.0**	1.33	1.22	1.21	1.2	1.0
	Mailles	**4.2**	**4.93**	**4.93**	**4.93**	**4.93**	**21.2**
256	Particules	**64.0**	**11.0**	**8.9**	**3.72**	1.83	1.0
	Mailles	**4.2**	**5.09**	**4.84**	**4.84**	**4.84**	**55.1**

TABLE 4.2 – Surconsommation mémoire pour le cas hétérogène par insertion centralisée.

La figure 4.12 présente les résultats obtenus sur le cas test hétérogène par insertion latérale pour 32 et 256 ressources. Nous constatons des disparités de déséquilibre de charge très prononcées. En effet, la méthode H_4P_∞ demeure très déséquilibrée, à l'exception des premières itérations. La méthode H_4P_∞ obtient un déséquilibre d'environ 10 avec 256 ressources. La méthode H_4P_X, pour toutes les autres configurations de X, provoque rapidement un déséquilibre après un partitionnement, si bien que la méthode H_4P_{50} est largement plus déséquilibrée que les méthodes H_4P_{20} et H_4P_5 avec 256 ressources.

Les résultats en équilibrage de charge sont contrastés. La qualité de la répartition de la charge de la méthode H_RP_X, pour les configurations $X = 5$ et $X = 20$, est encourageante. En effet, les résultats obtenus avec les méthodes H_4P_{20} et H_4P_5 sont correctes. En revanche, les résultats obtenus par les méthodes H_4P_{50} et H_4P_∞ mettent en évidence l'importance de ne pas choisir une valeur de X trop élevée.

Étude de régulation de donnée

Nous avons réalisé une étude comparative de surconsommation mémoire en nombre d'éléments de simulation. L'objectif de cette étude consiste à évaluer la capacité de l'approche par partitionnement à limiter la consommation mémoire.

La table 4.1 présente les résultats obtenus en surconsommation mémoire sur le cas test homogène. Nous constatons, premièrement, que la surconsommation mémoire de la méthode H_4P_X diminue quand X diminue. Avec 256 ressources de calcul, H_4P_{50} surconsomme 2.81 fois plus que la méthode H_4P_{20} en termes de particules. Dans ce cas, X est réduit de 2.5 fois pour un gain d'un facteur 2.81. Le gain est alors assez proportionnel à la réduction de X. En revanche, H_4P_{20} ne surconsomme que 1.27 fois plus que H_4P_5 dans ce cas, X est pourtant réduit d'un facteur 4. Ce résultat illustre que le choix de X n'est pas trivial. En effet, le gain obtenu en réduisant X n'est pas linéaire. Nous étudierons plus tard l'impact de la valeur de X sur le coût de la méthode. Ensuite, nous observons que la méthode H_4P_∞ surconsomme plus que la méthode H_R, confirmant ainsi les limitations observées dans l'étude de régulation de charge. Par ailleurs, nous pouvons noter que la surconsommation mémoire en nombre de mailles par ressource dépasse la contrainte donnée en entrée à l'outil de partitionnement. Ce résultat confirme la difficulté rencontrée par l'outil de partitionnement pour établir une partition équilibrée sur tous les critères.

La table 4.2 présente les résultats obtenus en surconsommation mémoire sur le cas test

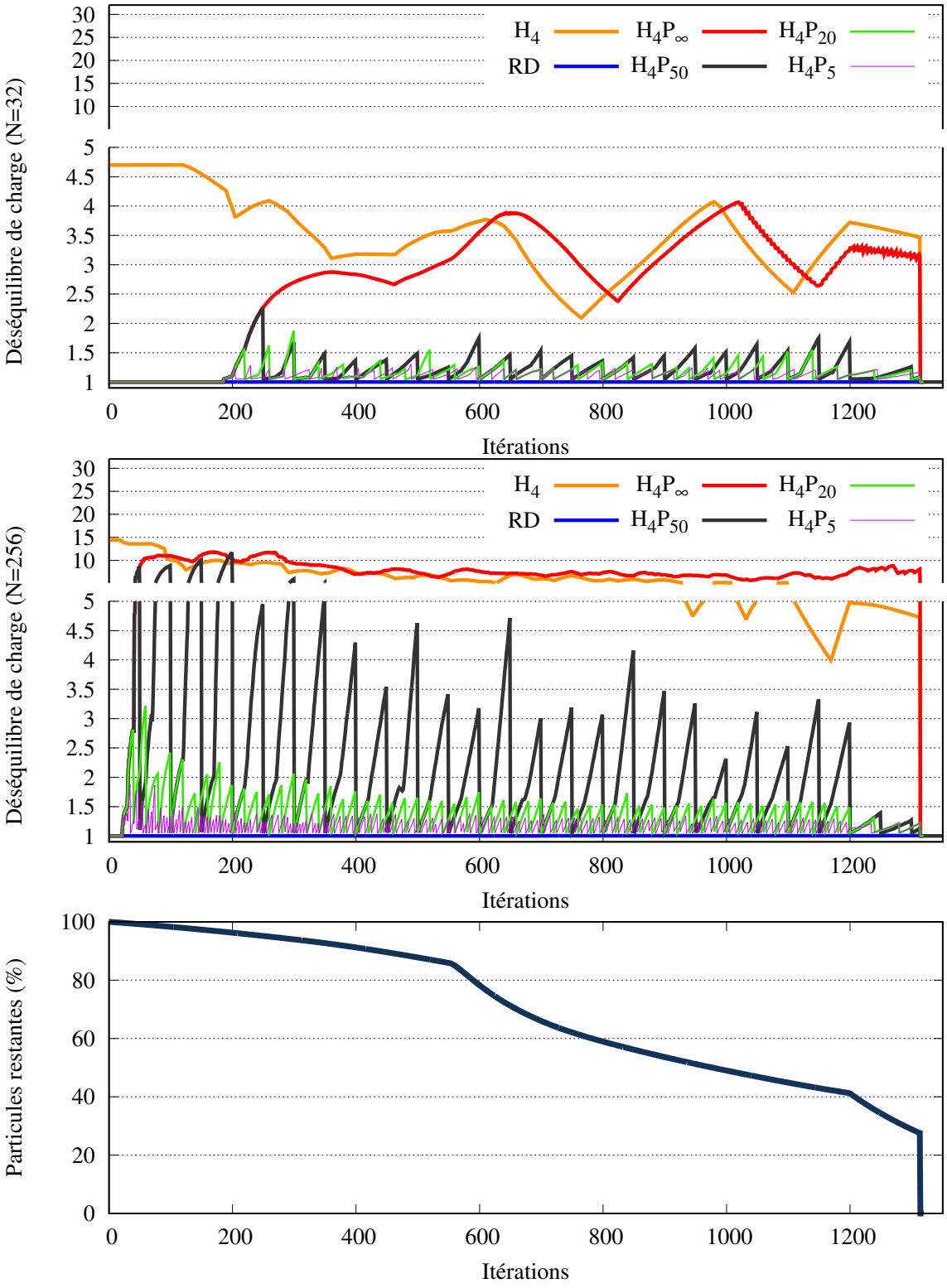


FIGURE 4.12 – Étude du déséquilibre de charge pour le cas test hétérogène par insertion latérale (voir figure 2.5).

N	Méthodes	H_4	H_4P_∞	H_4P_{50}	H_4P_{20}	H_4P_5	RD
32	Particules	**4.7**	2.82	2.16	1.76	1.22	1.0
	Mailles	**4.2**	**4.8**	**4.8**	**4.84**	**4.84**	**17.5**
256	Particules	**14.4**	**11.4**	**11.3**	**3.19**	1.87	1.0
	Mailles	**4.19**	**5.31**	**4.88**	**4.88**	**4.88**	**59.2**

TABLE 4.3 – Surconsommation mémoire pour le cas hétérogène par insertion latérale.

hétérogène par insertion centralisée. Si nous constatons ici aussi l'impact de X sur la surconsommation mémoire, nous observons surtout que la technique de partitionnement différé est très bénéfique pour ce cas test. En effet, alors que la méthode H_4 surconsomme d'un facteur $\frac{N}{4}$, les méthodes par partitionnements obtiennent des résultats sensiblement proches de ceux obtenus avec le cas test précédent, ils sont même meilleurs.

La table 4.3 présente les résultats obtenus en surconsommation mémoire sur le cas test hétérogène par insertion latérale. Nous observons une certaine stabilité dans les résultats obtenus avec la méthode par partitionnement. En effet, l'impact de X sur la surconsommation mémoire se vérifie encore. Nous observons que, globalement, la surconsommation mémoire est à peu près constante concernant les mailles. En revanche, la surconsommation mémoire concernant les particules augmente quand le nombre de ressources augmente. Cette augmentation de la surconsommation mémoire n'est toutefois pas proportionnelle.

Pour conclure, le constat de cette étude expérimentale est que la méthode H_RP_X, bien que ne parvenant pas complètement à satisfaire simultanément les objectifs de régulation de charge et de donnée, est une réelle alternative aux méthodes classiques. Nous faisons également le constat que la configuration $X = \infty$ n'est pas une solution viable aux problématiques de régulation de charge et de donnée, celle-ci obtenant même régulièrement de moins bons résultats que la méthode hybride. La méthode H_RP_X a permis d'améliorer considérablement la répartition de la charge de calcul et des données. Le choix de X joue un rôle primordial dans le gain observé. Les résultats observés nous inciteraient à réduire autant que possible la valeur de X. Nous devons toutefois ne pas occulter le fait que le coût de la méthode dépend largement du nombre de partitionnements. Nous proposons d'étudier les performances en temps de la méthode par partitionnement. Nous commençons par une étude en extensibilité forte.

4.3.2 Extensibilité forte

Nous proposons de réaliser une étude comparative en temps d'exécution de l'approche par partitionnement avec les méthodes RD et H_R en extensibilité forte. Nous allons étudier l'évolution du temps d'exécution de 1 à 256 ressources de calcul. Nous fixons le nombre total de particules insérées à 128 millions. Nous calculons l'efficacité parallèle pour chacune des méthodes en prenant la même référence, à savoir le temps d'exécution avec $N = 1$, en utilisant donc 16 cœurs de calcul.

Efficacité parallèle

La figure 4.13 présente les résultats obtenus en efficacité parallèle sur le cas test homogène jusqu'à 256 ressources. Nous observons que les performances de la méthode H_4P_∞ s'effondrent assez rapidement confirmant ainsi les conclusions faites par l'étude sur la répartition de la charge. Nous observons également une chute des performances de la méthode H_4P_5 à 256 ressources de calcul. Nous supposons que le coût en termes de partitionnement en est la cause. Enfin, la méthode

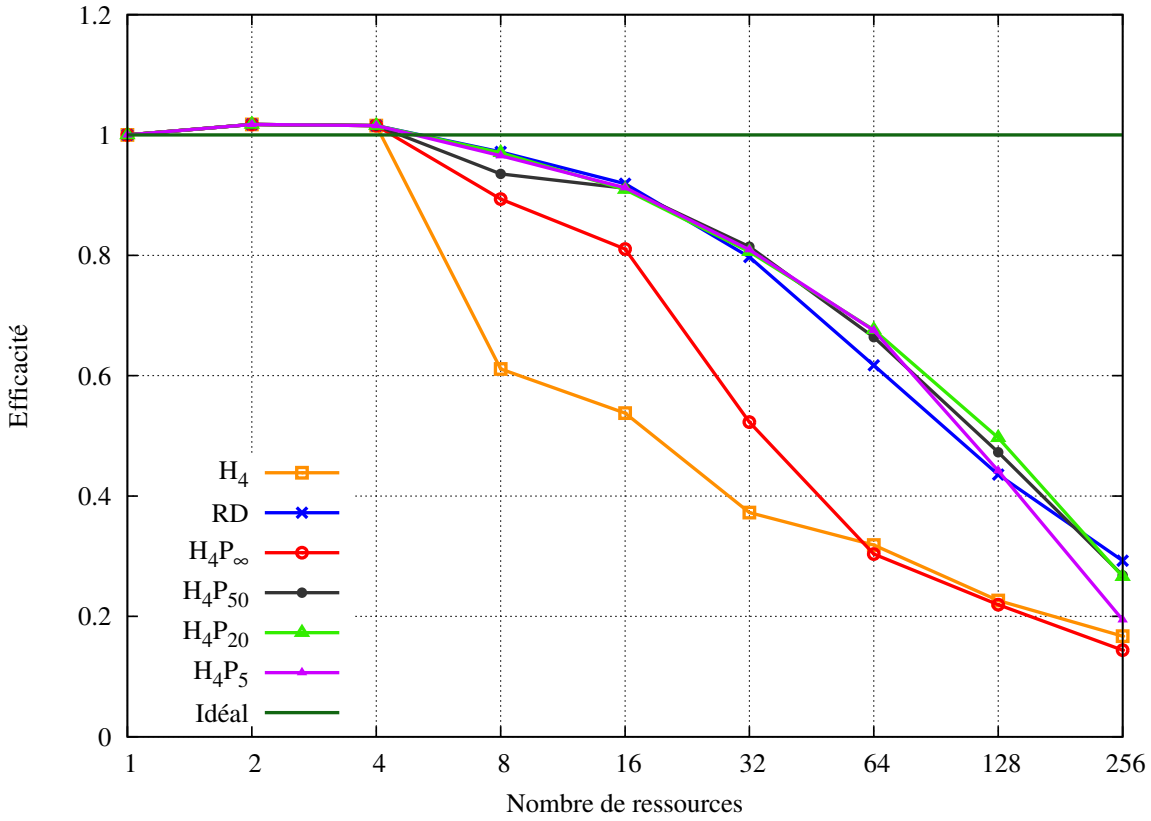


FIGURE 4.13 – Extensibilité forte pour le cas test homogène (meilleur en haut).

H_4P_{50} et la méthode H_4P_{20} obtiennent des performances proches de celles de la méthode RD retombant juste en dessous à 256 ressources.

La figure 4.14 présente les résultats obtenus en efficacité parallèle sur le cas test hétérogène par insertion centralisée jusqu'à 256 ressources. Nous observons que les performances des méthodes par partitionnement sont plus proches de celles de la méthode RD par rapport au cas test précédent. Le partitionnement étant davantage différé que sur le cas test homogène, les pertes de performance sont donc moindres.

La figure 4.15 présente les résultats obtenus en efficacité parallèle sur le cas test hétérogène par insertion latérale jusqu'à 256 ressources. Nous observons que les performances des méthodes par partitionnements sont moins bonnes avec ce cas test qu'avec les précédents. Ces résultats confirment les impressions de déséquilibre plus importants que nous avions observées lors de l'étude sur la répartition de la charge.

La méthode H_4P_{50} et la méthode H_4P_{20} obtiennent les meilleurs résultats des méthodes par partitionnement. Nous avons pourtant observé que la méthode H_4P_5 est la mieux équilibrée. Nous pensons que le coût de partitionnement est la cause de la perte de performance observée. Nous proposons de vérifier cette théorie avec un profilage de l'application.

Profilage

La table 4.4 présente le profilage de l'application pour le cas test homogène en extensibilité forte. Nous observons que les méthodes H_4 et H_4P_∞ obtiennent des temps très élevés pour le

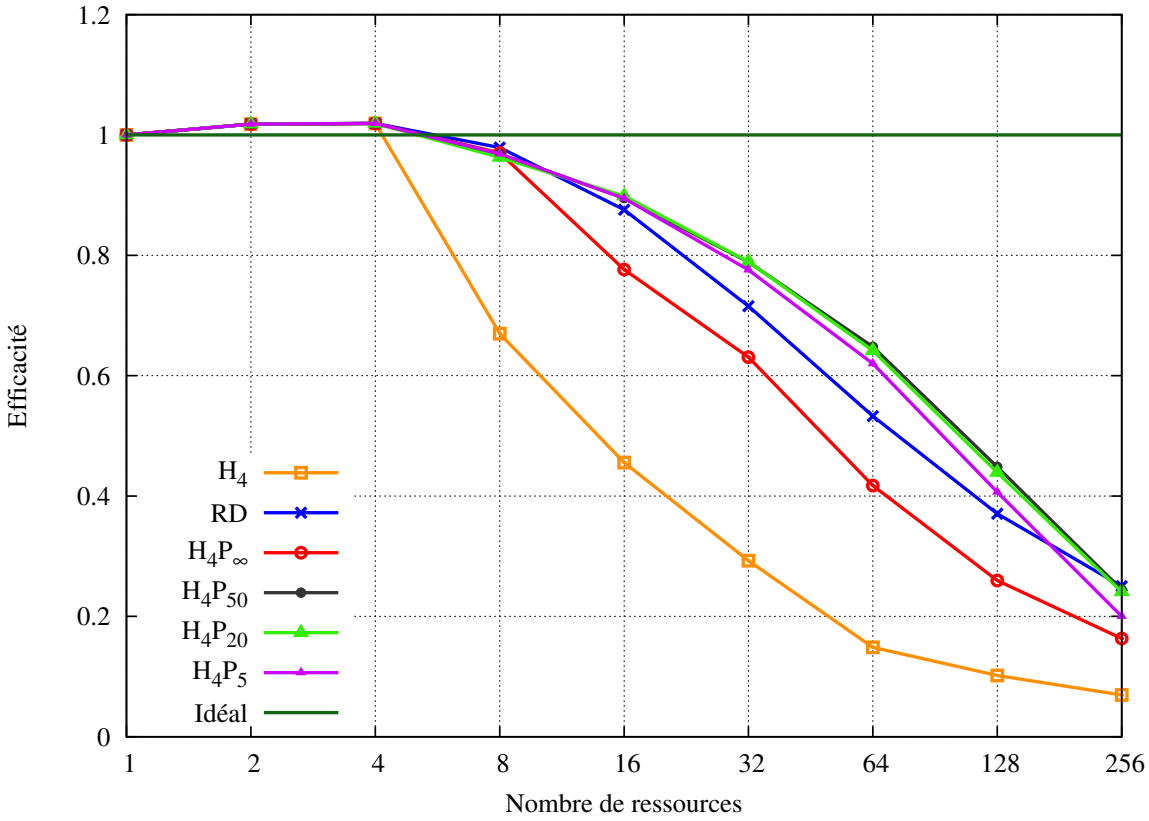


FIGURE 4.14 – Extensibilité forte pour le cas test hétérogène par insertion centralisée (meilleur en haut).

traitement des particules et des communications. Ce résultat confirme les déséquilibres de charge observés lors de l'étude en régulation de charge. Nous remarquons que la H_4P_∞ n'est pas meilleure que la méthode H_4. Le fait de partitionner une seule fois n'est donc pas suffisant. En choisissant $X = 50$, nous réduisons significativement le temps de traitement des particules et des communications. En revanche, nous augmentons le temps de partitionnement. En réduisant la valeur de X, nous réduisons encore le temps de traitement des particules et de communication mais nous augmentons de manière significative le temps de partitionnement. Le gain sur le temps de traitement de particules devient plus faible que l'augmentation du temps de partitionnement lorsque l'on passe de $X = 20$ à $X = 5$. La valeur de X la plus pertinente pour ce cas test en extensibilité forte est $X = 20$ ou $X = 50$.

La table 4.5 présente le profilage de l'application pour le cas test hétérogène par insertion

Méthodes	H_4	H_4P_∞	H_4P_{50}	H_4P_{20}	H_4P_5	RD
Particules + Communications	**176**	**212**	**87.9**	65.6	63.4	38.4
Mailles	9.78	9.21	10.1	10.6	10.3	**71.7**
Partitionnement(s)	0.51	4.18	**22.0**	**44.8**	**61.1**	0.0
Régulation (hors partitionnement)	0.0	0.0	0.0	0.01	0.02	0.0
Temps total	186	225	120	121	135	110

TABLE 4.4 – Profilage (secondes) en extensibilité forte pour le cas homogène.

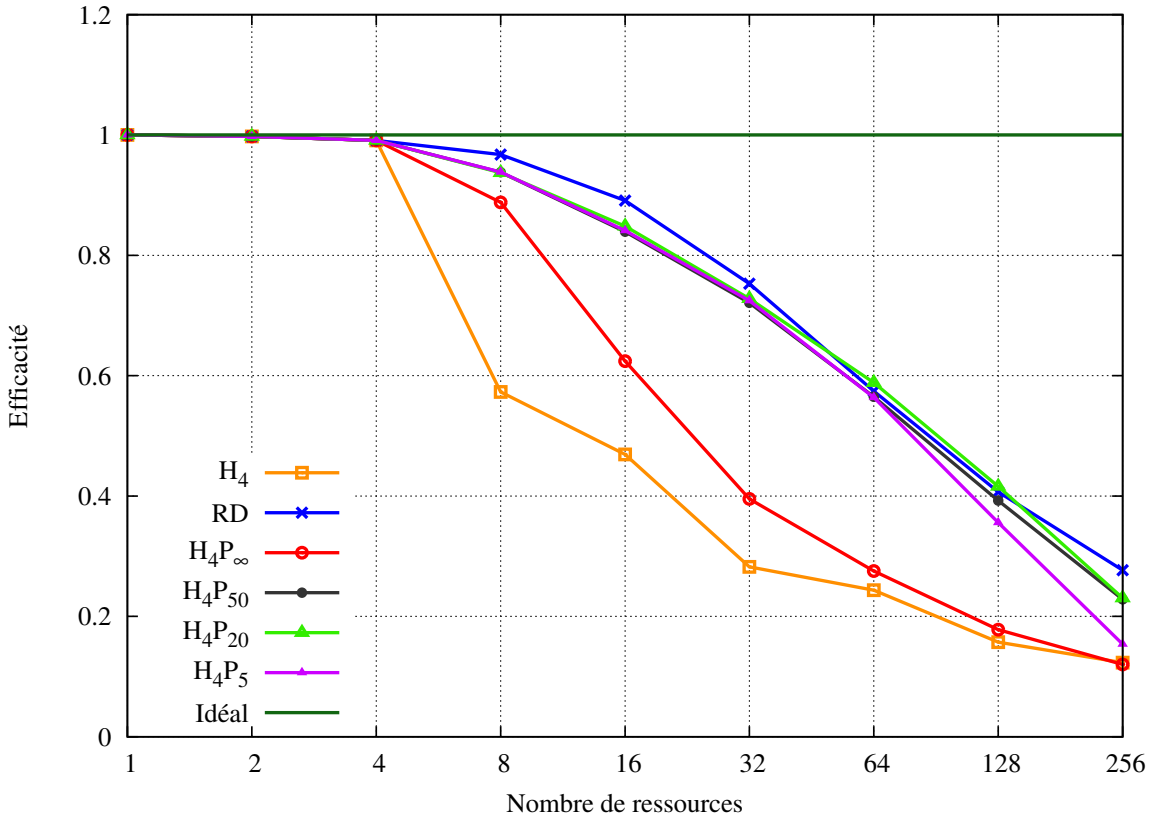


FIGURE 4.15 – Extensibilité forte pour le cas test hétérogène par insertion latérale (meilleur en haut).

centralisée en extensibilité forte. Nous pouvons dresser sensiblement le même bilan que sur le cas test précédent. La valeur de X la plus pertinente pour ce cas test en extensibilité forte est $X = 50$. Le temps total de la configuration $X = 20$ est néanmoins très proche de celui de la configuration $X = 50$.

La table 4.6 présente le profilage de l'application pour le cas test hétérogène par insertion latérale en extensibilité forte. Les résultats obtenus se rapprochent beaucoup de ceux obtenus avec les cas test précédents. La valeur de X la plus pertinente pour ce cas test en extensibilité forte est $X = 50$. Comme pour le cas test précédent, les configurations $X = 20$ et $X = 50$ obtiennent des temps totaux très proches.

Méthodes	H_4	H_4P_∞	H_4P_{50}	H_4P_{20}	H_4P_5	RD
Particules + Communications	**313**	**124**	**60.8**	49.4	49.6	30.2
Mailles	11.3	11.2	11.8	11.8	12.0	**60.6**
Partitionnement(s)	0.51	2.9	**19.3**	**34.7**	**47.3**	0.0
Régulation (hors partitionnement)	0.0	0.0	0.01	0.01	0.02	0.0
Temps total	325	138	92.0	95.9	109	90.7

TABLE 4.5 – Profilage (secondes) en extensibilité forte pour le cas hétérogène par insertion centralisée.

Méthodes	H_4	H_4P_∞	H_4P_{50}	H_4P_{20}	H_4P_5	RD
Particules + Communications	**213**	**243**	**85.4**	60.9	60.1	37.7
Mailles	10.8	10.7	11.0	11.2	11.9	**67.2**
Partitionnement(s)	0.52	3.99	**26.5**	**53.5**	**79.1**	0.0
Régulation (hors partitionnement)	0.0	0.0	0.0	0.01	0.02	0.0
Temps total	224	258	123	126	151	105

TABLE 4.6 – Profilage (secondes) en extensibilité forte pour le cas hétérogène par insertion latérale.

Conclusion

La méthode par partitionnement n'est pas parvenue à dépasser la méthode RD en termes de performance, en efficacité parallèle en extensibilité forte. Le coût en partitionnements est une forte limitation, celui-ci étant parfois supérieur au temps de traitement des particules. Le déséquilibre de charge entraîne également une baisse des performances. La méthode par partitionnement réussit toutefois à dépasser nettement la méthode hybride avec degré de réplication constant. La configuration $X = 50$ a obtenu les meilleurs résultats en extensibilité forte. Notons que la configuration $X = 20$ a obtenu des résultats très proches de ceux de la configuration $X = 50$.

Les conséquences du déséquilibre de charge, en termes de temps d'exécution, deviennent moins importantes lorsque le nombre de particules diminue. Ces déséquilibres seraient potentiellement davantage préjudiciables avec un nombre de particules par ressource constant. Nous proposons donc d'étudier le comportement de la méthode par partitionnement en extensibilité faible.

4.3.3 Extensibilité faible

Nous proposons de réaliser une étude comparative en efficacité parallèle des méthodes par partitionnement avec les méthodes RD et H_R en extensibilité faible. Nous allons étudier l'évolution du temps d'exécution de 1 à 256 ressources de calcul. Nous fixons le nombre moyen de particules insérées par ressource à 4 millions. Nous traitons donc jusqu'à 1,024 milliard de particules avec 256 ressources.

Efficacité parallèle

La figure 4.16 présente les résultats obtenus en efficacité parallèle sur le cas test homogène jusqu'à 256 ressources, en extensibilité faible. Nous rappelons que dans cette étude le nombre de particules augmente proportionnellement au nombre de ressources. Le fait que la méthode par partitionnement obtient une efficacité parallèle supérieure à 1 jusqu'à 64 ressources s'explique donc par la décomposition du domaine. Nous constatons une chute des performances de la méthode H_4P_X à partir de 64 ressources. Notons que les configurations $X = 20$ et $X = 5$ obtiennent les meilleurs performances des méthodes par partitionnement. Elles obtiennent d'ailleurs pratiquement les mêmes résultats. La méthode H_4P_∞, quant à elle, voit ses performances chuter dès $N = 8$. À l'exception de la configuration $X = \infty$, nous observons que la méthode H_4P_X obtient des performances en extensibilité faible largement meilleures comparées à celles de la méthode H_4. Ce gain est probablement la conséquence d'une meilleure répartition des particules que nous avons observée en sous-section 4.3.1.

La figure 4.17 présente les résultats obtenus en efficacité parallèle sur le cas test hétérogène par insertion centralisée jusqu'à 256 ressources en extensibilité faible. Nous constatons une

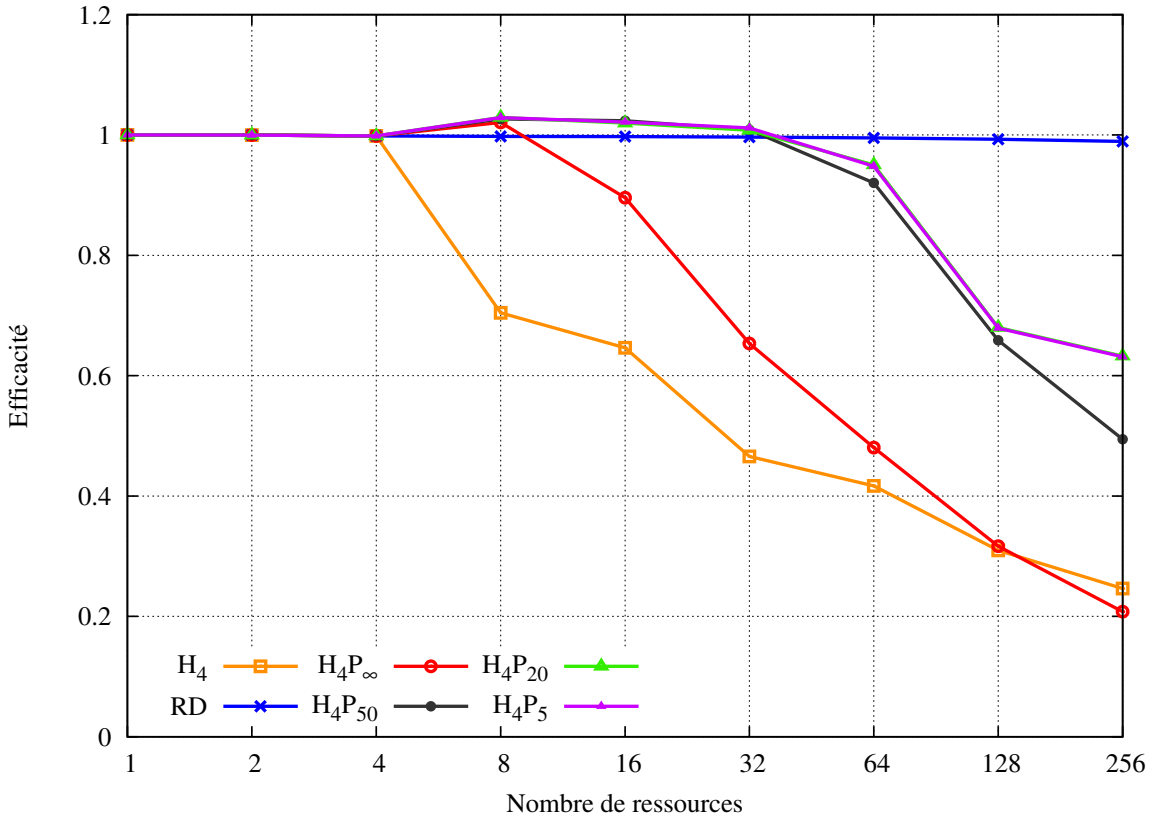


FIGURE 4.16 – Extensibilité faible pour le cas test homogène (meilleur en haut).

amélioration des performances de la méthode H_4P_X en comparaison avec le cas test précédent. Notons que, ici aussi, les configurations $X = 20$ et $X = 5$ obtiennent les meilleurs performances.

La figure 4.18 présente les résultats obtenus en efficacité parallèle sur le cas test hétérogène par insertion latérale jusqu'à 256 ressources en extensibilité faible. Nous observons des performances proches de celles du cas test précédent. Les performances sont globalement légèrement moins bonnes. La configuration $X = 20$ obtient les meilleurs résultats sur ce cas test.

Cette étude a montré que la méthode H_4P_X, avec $X \neq \infty$, obtient des résultats, qui bien que non-optimaux, largement meilleurs que la méthode hybride avec degré de réplication constant. Nous obtenons même des résultats très proches de ceux de la méthode RD jusqu'à $N = 64$. Nous proposons maintenant de profiler l'application afin d'affiner notre étude.

Profilage

La table 4.7 présente le profilage de l'application pour le cas test homogène en extensibilité faible. En comparaison avec le profilage en extensibilité forte, nous observons des écarts importants sur le temps de traitement des particules et des communications. Ce résultat explique le fait que la configuration $X = 50$ ne soit plus la meilleure configuration en extensibilité faible. Nous observons seulement 1 seconde d'écart sur le temps total entre les configurations $X = 20$ et $X = 5$. Pourtant, les résultats du profilage montrent des comportements différents. Similairement au profilage en extensibilité forte, le fait de baisser la valeur de X permet de réduire le temps de traitement des particules et des communications mais implique une augmentation du temps de partitionnement.

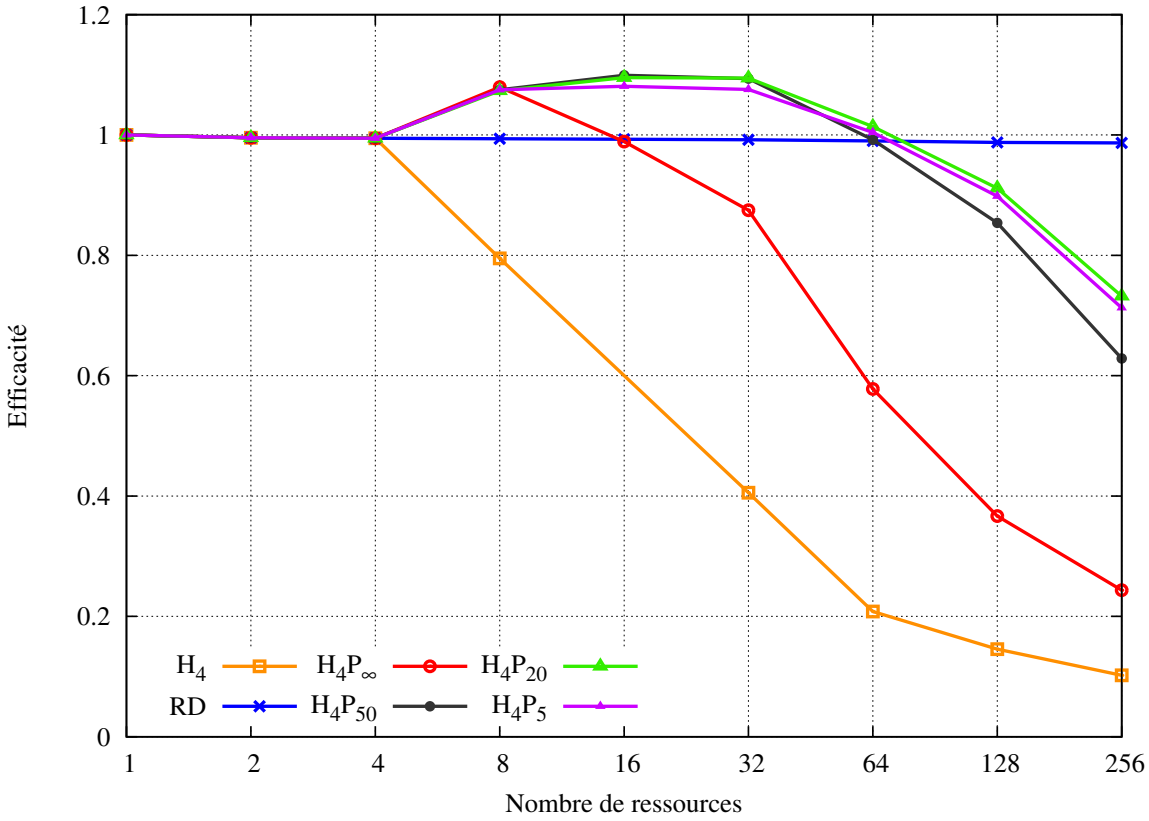


FIGURE 4.17 – Extensibilité faible pour le cas test hétérogène par insertion centralisée (meilleur en haut).

En extensibilité faible, le temps de traitement des particules est plus important, et le gain obtenu en améliorant la répartition des particules permet de compenser l'augmentation de la fréquence de partitionnement.

La table 4.8 présente le profilage de l'application pour le cas test hétérogène par insertion centralisée en extensibilité faible. Nous observons beaucoup de similitudes avec le cas test précédent. Toutefois, le passage de $X = 20$ à $X = 5$ amène un gain nettement inférieur. La configuration $X = 20$ obtient donc le temps total le plus faible.

La table 4.9 présente le profilage de l'application pour le cas test hétérogène par insertion latérale en extensibilité faible. Nous observons, sur ce cas test également, que le gain obtenu en passant de $X = 20$ à $X = 5$ sur le traitement des particules ne permet pas de compenser l'augmentation du temps de partitionnement.

Méthodes	H_4	H_4P_∞	H_4P_{50}	H_4P_{20}	H_4P_5	RD
Particules + Communications	**1326**	**1555**	**600**	423	388	263
Mailles	9.86	10.0	10.2	10.6	10.8	**67.7**
Partitionnement(s)	0.51	4.81	**44.2**	**75.0**	**110**	0.0
Régulation (hors partitionnement)	0.0	0.0	0.0	0.01	0.02	0.0
Temps total	1336	1569	654	508	509	331

TABLE 4.7 – Profilage (secondes) en extensibilité faible pour le cas homogène.

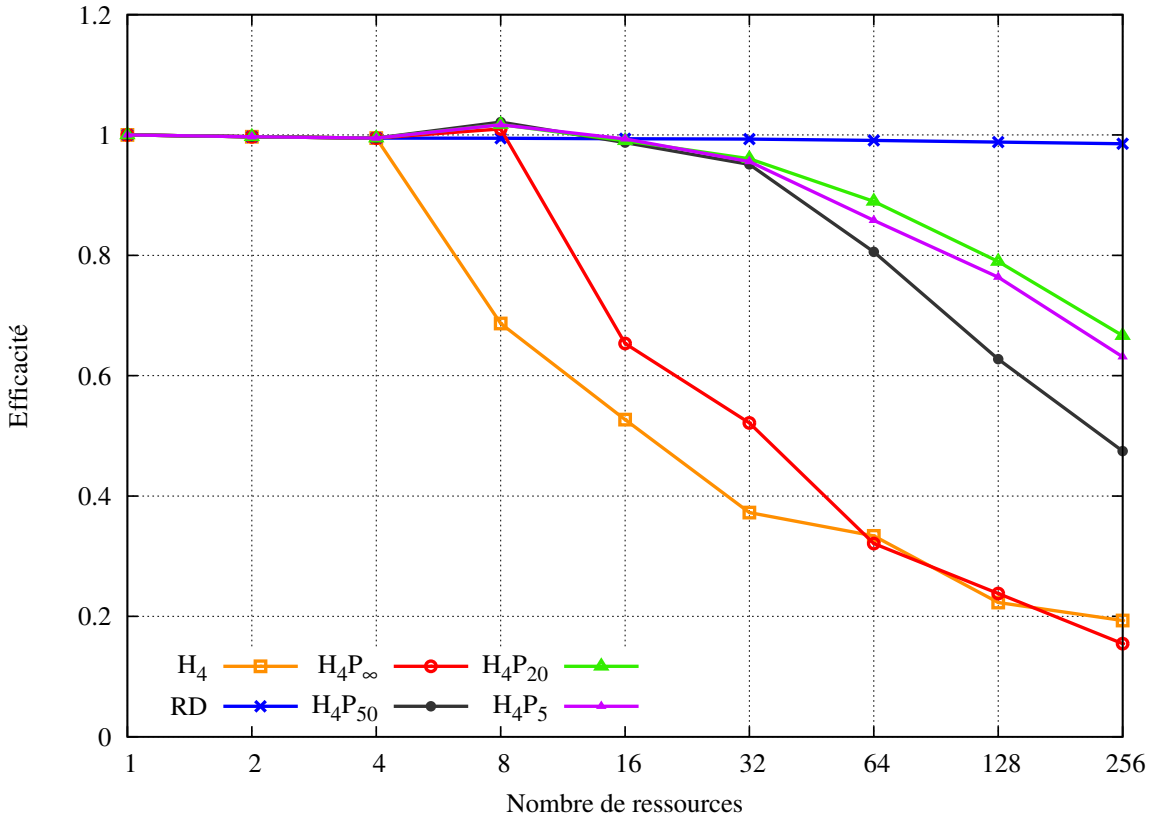


FIGURE 4.18 – Extensibilité faible pour le cas test hétérogène par insertion latérale (meilleur en haut).

Conclusion

En terme d'extensibilité faible, la méthode RD obtient toujours les meilleures performances. Nous constatons que la méthode $H_R P_X$ est cependant parvenue à approcher les performances de la méthode RD surpassant ainsi largement les performances de la méthode hybride avec degré de réplication constant. La configuration $X = 20$ a obtenu les meilleurs résultats en extensibilité faible.

Méthodes	H_4	H_4P_∞	H_4P_{50}	H_4P_{20}	H_4P_5	RD
Particules + Communications	**2530**	**1021**	**352**	268	253	184
Mailles	11.7	12.5	12.0	12.0	12.3	**77.0**
Partitionnement(s)	0.51	4.28	**36.6**	**61.5**	**88.3**	0.0
Régulation (hors partitionnement)	0.0	0.0	0.01	0.01	0.03	0.0
Temps total	2542	1038	400	341	354	261

TABLE 4.8 – Profilage (secondes) en extensibilité faible pour le cas hétérogène par insertion centralisée.

Méthodes	H_4	H_4P_∞	H_4P_{50}	H_4P_{20}	H_4P_5	RD
Particules + Communications	**1606**	**1948**	**567**	351	331	239
Mailles	10.6	10.3	11.4	11.1	11.8	**76.1**
Partitionnement(s)	0.53	4.59	**50.8**	**89.0**	**133**	0.0
Régulation (hors partitionnement)	0.0	0.0	0.0	0.01	0.02	0.0
Temps total	1617	1963	629	451	475	315

TABLE 4.9 – Profilage (secondes) en extensibilité faible pour le cas hétérogène par insertion latérale.

4.4 Conclusion

Les méthodes de décomposition de domaine et hybride avec degré de réplication constant ont montré des performances très éloignées de l'optimal. Ces mauvais résultats sont dus à d'importants déséquilibres de charge. Les méthodes de réplication de domaine et hybride avec degré de réplication dépendant de N ont, quant à elles, montré une surconsommation mémoire relative aux mailles conséquente. De plus, elles obtiennent un surcoût de traitement non négligeable sur les mailles notamment en extensibilité forte. Pour résoudre ces problèmes, nous avons proposé une méthode par partitionnement de graphe. Cette méthode consiste à optimiser la méthode hybride avec degré de réplication constant en éliminant autant que possible les déséquilibres de charge et de donnée en utilisant les techniques de partitionnement de graphe. La configuration $X = \infty$, consistant à ne partitionner qu'une seule fois, a échoué à réduire le déséquilibre de charge de la méthode hybride. Elle a même, le plus souvent, accentué ce déséquilibre de charge. En revanche, les autres configurations ont obtenu des résultats plus intéressants. Elle ont, en particulier, permis une très significative réduction de la consommation mémoire relative aux mailles en comparaison avec la méthode RD tout en augmentant pas de manière trop significative la consommation mémoire relative aux particules. La méthode par partitionnement de graphe est donc une sérieuse avancée dans la résolution de la problématique de régulation de charge et de donnée.

Le paramétrage de la valeur de X joue donc un rôle primordial dans le comportement de cette méthode. La configuration $X = 20$ est celle, parmi les configurations testées, obtenant globalement les meilleurs résultats, extensibilités fortes et faibles confondues. La configuration $X = 20$ sera donc notre référence pour la suite du document.

La méthode H_RP_X a obtenu des résultats en extensibilité faible et forte largement meilleurs par rapport à la méthode hybride avec degré de réplication constant, se rapprochant même de ceux de la méthode RD. Cependant, des déséquilibres demeurent, expliquant pourquoi cette méthode n'atteint pas les performances de la méthode RD. Ces déséquilibres apparaissent entre les partitionnements. Il s'avère donc que l'absence de mécanismes de régulation entre les partitionnements impact de manière significative la qualité de la répartition de la charge de calcul et des données de cette méthode. De plus, le coût de partitionnement, croissant proportionnellement au nombre de ressources, est une limitation très forte de cette méthode. Des mécanismes plus dynamiques ainsi que moins coûteux sont nécessaires. Nous proposons donc de nous orienter vers une méthode de régulation dynamique de charge et de donnée.

Chapitre 5

Régulation dynamique de charge et de donnée

Les limitations de l'approche par partitionnement, que nous avons présentée au chapitre 4 sont essentiellement dues à un manque de régulation de charge entre les partitionnements. Nous avons également observé que l'augmentation de la fréquence de partitionnement impliquait un surcoût en temps supérieur au gain obtenu. Cette approche a toutefois permis de mettre en évidence la nécessité de prendre en compte dynamiquement la répartition de la charge de calcul au moment d'opérer une distribution des données de simulation. La méthode utilisée doit être dynamique pour être capable de tenir compte de l'évolution de la répartition de la charge de calcul sur le domaine de simulation.

Afin de minimiser les coûts de communication et de synchronisation entre les ressources, nous proposons de nous orienter vers une approche de régulation ne nécessitant pas une vision globale de la répartition de la charge de calcul et des données. Nous proposons que chaque ressource ait une connaissance partielle de la répartition de la charge de calcul et des données par une approche de voisinage. Cette approche consiste à ce que chaque ressource choisisse un ensemble de ressources avec lesquelles une régulation de charge et de donnée sera possible. Nous proposons donc une approche de régulation dynamique de charge et de donnée par des opérations de régulation locales.

5.1 Algorithme général

En utilisant une méthode de régulation dynamique et locale, la représentation des charges de calcul et des données de toutes les ressources confondues n'est pas indispensable. La modélisation sous forme de graphe que nous avons proposée au chapitre 4 convient aux techniques de partitionnement. Nous proposons d'adapter ce modèle afin qu'il puisse mieux prendre en compte le principe de localité. L'idée est que chaque ressource possède son propre sous-graphe, contenu dans le graphe global introduit au chapitre 4, représentant ses propres mailles et ses propres particules. Nous proposons également que la composition en sommets de ces sous-graphes puisse évoluer dynamiquement.

L'approche de voisinage consiste à ce que chaque ressource ait accès à ses données et aux données de quelques autres ressources préalablement sélectionnées. Nous proposons concrètement que les ressources s'échangent leurs sous-graphes et nombres de particules respectifs.

Le processus de régulation de notre approche dynamique est réalisé en deux étapes bien distinctes. La première consiste à limiter le nombre de mailles par ressource et la seconde consiste à

redistribuer les particules en cas de déséquilibre. L'intérêt de procéder en deux étapes distinctes est de réduire la complexité de la problématique de régulation de charge et de donnée en la scindant en deux problématiques moins complexes. Ces algorithmes se baseront sur le modèle de graphe. La régulation de charge et de donnée s'effectuera par le transfert de sommets entre les sous-graphes.

L'algorithme 8 présente les étapes principales de la simulation lorsque l'approche dynamique est utilisée. À chaque itération, nous commençons par le traitement des particules (ligne 2). Ensuite, le sous-graphe appartenant à la ressource est mis à jour (ligne 3). Cette étape sera détaillée en section 5.2. L'algorithme de voisinage est invoqué en ligne 4. Cet algorithme sera détaillé en section 5.3. La limitation du nombre de mailles est effectuée en ligne 5. L'algorithme de limitation du nombre de mailles sera présenté en section 5.4. Enfin, l'équilibrage en nombre de particules est réalisé en ligne 6. L'algorithme de régulation de charge sera détaillé en section 5.5.

Algorithme 8 : Algorithme général.

```
   /* Tant que la condition de fin de simulation n'est pas atteinte        */
1  tant que !fin_application faire
       /* Phase de Transport                                               */
2      Monte_carlo_transport( Maillage, Particules )
       /* Mise à jour du sous-graphe local (section 5.2)                   */
3      Mise_a_jour_sous_graphe( Graphe, Maillage, Particules )
       /* Établissement du voisinage (section 5.3)                         */
4      Construction_voisinage( Graphe)
       /* Limitation du nombre de sommets (section 5.4)                    */
5      Limitation_mailles( Graphe, Maillage, Particules )
       /* Régulation de charge (section 5.5)                               */
6      Regulation_particules( Graphe, Maillage, Particules )
7  fin
```

5.2 Construction du sous-graphe

Chaque ressource possède son propre sous-graphe représentant ses propres données. Nous proposons que la composition des sous-graphes puisse être dynamique. Nous proposons que chaque sous-graphe puisse ajouter ou retirer dynamiquement des sommets. Nous commencerons par présenter quelques règles de composition des sous-graphes en termes de sommets en sous-section 5.2.1. Puis, nous définirons une nouvelle pondération capable de tenir compte du principe de localité. Nous redéfinirons la pondération des arêtes en sous-section 5.2.2. Enfin, nous redéfinirons la pondération des sommets en sous-section 5.2.3.

5.2.1 Sous-graphe dynamique et instances de sommets

Nous proposons de représenter la charge de calcul et les données de chaque ressource de manière indépendante. Concrètement, nous proposons d'attribuer à chaque ressource, notée i, son propre sous-graphe noté $G'_i = (S'_i, A'_i)$.

Chaque sommet du graphe global représente un ensemble de mailles. Ces mailles ont des coordonnées permettant de les situer dans le domaine. Nous proposons de localiser chaque sommet par des coordonnées obtenues à partir des coordonnées des mailles sous-jacentes. Deux sous-graphes différents peuvent posséder des sommets ayant les mêmes coordonnées. Nous proposons d'introduire la notation s_i^x, où s_i^x désigne le sommet s^x appartenant au sous-graphe G'_i de coordonnées $x = (x_1, x_2, ..., x_d) \in X^d$ (d désignant le nombre de dimensions). Nous nommons s_i^x l'instance du

sommet s^x du sous-graphe G'_i. Deux instances de sommet, ayant les mêmes coordonnées, désignent le même sommet du graphe global et donc les mêmes mailles. Dans le cas où les mailles nécessitent un stockage mémoire, même lorsqu'elles n'ont pas de particules, il doit exister au moins une instance pour tous les sommets appartenant au graphe global. La condition de l'équation 5.1 doit alors être respectée.

$$\forall x = (x_1, x_2, ..., x_d) \in X^d \; \exists i \in \{0, 1, ..., N-1\} \; s_i^x \in S'_i \; . \tag{5.1}$$

Nous proposons de distribuer l'ensemble des sommets $s \in S$ sur les ressources en utilisant une méthode de partitionnement équilibré en nombre de sommets. Ainsi, nous définissons $\overset{\circ}{S}_i$ l'ensemble des sommets correspondant au sous-graphe de i. Ainsi, $\forall s \in S \; \exists i s \in \overset{\circ}{S}_i$.

La pondération des sommets a pour principal objectif d'exprimer une répartition de la charge de calcul et des données. L'approche par sous-graphe local et dynamique amène à considérer la pondération des sommets et des arêtes. Nous allons commencer par redéfinir la pondération des arêtes.

5.2.2 Pondération des arêtes

Les arêtes permettaient de quantifier les communications aux frontières des sous-domaines pour les méthodes par partitionnement. Nous proposons que désormais l'on puisse ajouter des sommets dynamiquement aux sous-graphes. Par conséquent, les arêtes ne représentent plus des communications potentielles. En revanche, une arête demeure un accès d'un sommet à un autre pour les particules. Lorsqu'une arête lie deux sommets s et s' tels que $s \in S'_i$ et $s' \in S'_i$, elle informe des possibles migrations de particules entre les deux sommets. Supposons que nous puissions construire une arête entre un sommet $s \in S$ et un sommet $s' \in S'_i$, c'est-à-dire quand l'un des deux sommets n'appartient pas au sous-graphe de la ressource i. Cette arête a des propriétés intéressantes puisque la migration de particules de s' vers s impliquera la création d'un nouveau sommet dans le sous-graphe G'_i. Nous proposons donc en premier lieu de catégoriser les arêtes selon les types de sommets qu'elles relient.

Catégorisation des arêtes

Les arêtes reliant deux sommets appartenant à S'_i appartient par définition à A'_i. Nous proposons de considérer les sommets appartenant à $\overset{\circ}{S}_i$ de la même façon que les sommets S'_i. En effet, ces sommets représentent des mailles que la ressource doit potentiellement stocker en mémoire. Une arête a appartient alors à A'_i si a vérifie l'équation 5.2.

$$\begin{cases} a = (s^x, s^y) \in A, s^x \in S'_i, s^y \in S'_i \implies a \in A'_i \; ; \\ a = (s^x, s^y) \in A, s^x \in S'_i, s^y \in \overset{\circ}{S}_i \implies a \in A'_i \; ; \\ a = (s^x, s^y) \in A, s^x \in \overset{\circ}{S}_i, s^y \in \overset{\circ}{S}_i \implies a \in A'_i \; . \end{cases} \tag{5.2}$$

Considérons maintenant les arêtes reliant un sommet $s \in S$ à un sommet $s' \in S'_i \cup \overset{\circ}{S}_i$. Nous notons A''_i cet ensemble. Une arête a appartient à A''_i si a vérifie l'équation 5.3. Notons qu'une arête ne peut pas appartenir à la fois à A'_i et à A''_i (équation 5.4).

$$a = (s^x, s^y) \in A, s^x \in S'_i, s^y \in S \setminus (\overset{\circ}{S}_i \cup S'_i) \implies a \in A''_i \; . \tag{5.3}$$

$$A'_i \cap A''_i = \varnothing \ . \tag{5.4}$$

Contrairement aux arêtes appartenant à A'_i, les arêtes de A''_i désignent de potentielles créations de sommets. Nous proposons de pondérer les arêtes en fonction de la catégorie auxquelles elles appartiennent.

Pondération par catégorie

La création de nouveaux sommets implique la création de nouvelles mailles. Nous ciblons comme objectifs de limiter l'empreinte mémoire et répartir la charge de calcul. Cela dépend en partie du nombre de mailles que possède la ressource. La création de nouvelles mailles est donc contraire à ces objectifs. Par conséquent, une arête désignant la création potentielle d'un sommet exprime un surcoût en termes de mémoire et de calcul. En revanche, une arête reliant deux sommets appartenant déjà au graphe G'_i n'implique pas ce surcoût. Le surcoût supposé étant à priori indépendant du nombre de particules, nous proposons donc de pondérer les arêtes de manière binaire tel que défini par l'équation 5.5.

$$\begin{cases} \forall a_i \in A'_i & w'(a_i) = 0 \ ; \\ \forall a_i \in A''_i & w'(a_i) = 1 \ . \end{cases} \tag{5.5}$$

Nous avons redéfini la pondération des arêtes. Nous proposons maintenant de redéfinir la pondération des sommets.

5.2.3 Pondération des sommets

Notre approche consiste à considérer un sommet de manière locale à la ressource. Pour le graphe global, introduit au chapitre 4, nous n'avions pas cet aspect à prendre en considération. La pondération des sommets était alors uniquement déterminée par le nombre de particules. Nous proposons de considérer d'autres critères de pondération pour les sommets des sous-graphes dynamiques. La pondération multi-critères a pour objectif de tenir compte de plusieurs propriétés. L'idée est donc d'évaluer un sommet selon plusieurs critères et de déterminer globalement le poids qui lui correspond le mieux. Il existe plusieurs méthodes de pondération multi-critères. Il y a la pondération ordonnée où l'on décompose de manière récursive les sommets en plusieurs groupes. Les critères sont alors classés par ordre prioritaire. Il y a également la pondération par méthode de moyenne pondéréee. Nous avons choisi cette méthode car elle présente l'avantage d'être facilement paramétrable. Nous avons également considéré des cas particuliers.

Méthode de pondération

Une moyenne pondéréee consiste à considérer un ensemble de valeurs et de coefficients associés tels que, plus un coefficient est élevé, plus la valeur associée impactra la moyenne.

Soit f une fonction de pondération multi-critères par moyenne pondéréee et $w^k(x)$ une fonction de pondération pour le $k^{\text{ème}}$ critère. f est alors définie par l'équation 5.6.

$$f(x) = \frac{\sum\limits_{k=1}^{N_w} w^k(x) c^k}{\sum\limits_{i=k}^{N_w} c^k} \ . \tag{5.6}$$

Nous admettons que, pour que cette moyenne pondéréee ait un sens, toutes les fonctions w^k doivent être du même ordre de grandeur. Nous proposons donc de borner ces fonctions. Connaissant $\underline{w}$ et $\overline{w}$, nous proposons l'intervalle de valeurs des poids des sommets $s \in S'_i$ tel que défini par l'équation 5.7. Le choix de $\underline{w}$ et $\overline{w}$ permet de définir la finesse de la pondération.

$$\forall s \in S'_i \, \forall k \in \{1, 2, ..., N_w\} \, \underline{w} \leq w^k(s) \leq \overline{w} \, . \tag{5.7}$$

Chaque sommet $s \in \overset{\circ}{S}_i$ peut appartenir ou non à S'_i. Dans tous les cas, ces sommets doivent en priorité appartenir au sous-graphe de la ressource i. Nous proposons de définir une pondération particulière telle que le définit l'équation 5.8. Ainsi, nous donnons à ces sommets une pondération maximale.

$$\forall s \in \overset{\circ}{S}_i \, w'(s) = \overline{w} + 1 \, . \tag{5.8}$$

Nous avons pris en compte dans la fonction de pondération du sous-graphe d'une ressource tous les sommets appartenant à ce sous-graphe ou susceptible de lui appartenir. Nous proposons de définir la pondération par défaut consistant à donner un poids à chaque sommet possiblement possédé par une autre ressource. Cette pondération par défaut est donnée par l'équation 5.9.

$$\forall s \in S, s \notin (S'_i \cup \overset{\circ}{S}_i) \, w'(s) = 0 \, . \tag{5.9}$$

Pour résumer, nous avons distingué trois cas : (i) le sommet appartient au graphe de la ressource ; (ii) le sommet appartient au graphe de la ressource issue du partitionnement noté $\overset{\circ}{S}_i$; ou (iii) le sommet n'appartient à aucun des graphes de la ressource mais uniquement au graphe global S. Chacun des cas suit une règle de pondération différente. L'équation 5.10 donne la pondération des sommets pour la ressource i.

$$\begin{cases} \forall s \in S'_i \setminus \overset{\circ}{S}_i & w'(s) = f(s) = \dfrac{\sum\limits_{k=1}^{N_w} w^k(s) c^k}{\sum\limits_{i=k}^{N_w} c^k} \, ; \\ \forall s \in \overset{\circ}{S}_i & w'(s) = \overline{w} + 1 \, ; \\ \forall s \in S \setminus S'_i \cup \overset{\circ}{S}_i & w'(s) = 0 \, . \end{cases} \tag{5.10}$$

Nous proposons maintenant de définir les critères de pondération permettant de pondérer les sommets $s \in S'_i$. Le volume de charge de calcul et de donnée d'un sommet est fonction du nombre de particules de ce sommet. Nous proposons donc de considérer la densité de particules comme premier critère dans la pondération.

Critère n° 1 : densité de particules

Le premier critère que nos présentons permet de pondérer en fonction du nombre de particules de chaque sommet. Deux étapes sont alors nécessaires. Il faut d'abord calculer le nombre de particules par sommet puis calculer le poids correspondant.

Le nombre de particules représentées par un sommet est obtenu par l'équation 5.11. Nous souhaitons obtenir un poids permettant de quantifier ce nombre. Il existe plusieurs approches. Une approche de type clé-valeur consiste à ce qu'un nombre de particules corresponde à un poids donné. Une approche de type classification consiste à pondérer les sommets les uns par rapport aux autres.

Nous avons opté pour la seconde. En effet, elle permet d'exprimer la charge de calcul d'un sous-graphe indépendemment de tous les autres et de l'ordre de grandeur du nombre de particules.

$$p_i^x = \sum \rho_m,\ \forall m \in s_i^x\ . \tag{5.11}$$

Soit s_i^{pmax} le sommet ayant le plus de particules obtenu au moyen de l'équation 5.12. Attribuer un poids $w^1 \in [\underline{w}, \overline{w}]$ en fonction du nombre de particules pour chaque sommet $p_i^x \in [0, p_i^{pmax}]$ consiste à opérer une transformation affine. Soit $I_1 = [a, b]$ l'intervalle de départ et $I_2 = [c, d]$ l'intervalle d'arrivée. La fonction $y = t(x), x \in I_1, y \in I_2$, transformation affine de I_1 vers I_2, est définie par l'équation 5.13. Il nous suffit donc, pour obtenir le poids selon le premier critère, d'appliquer cette fonction avec $a = 0$, $b = p_i^{pmax}$, $c = \underline{w}$ et $d = \overline{w}$. $w^1(s_i^x)$, le poids d'un sommet $s_i^x \in S_i'$ selon le premier critère, est donné par l'équation 5.14.

$$\exists s_i^{pmax} \in S_i', \forall s_i^x \in S_i'\ p_i^x \leq p_i^{pmax}\ . \tag{5.12}$$

$$y = (\frac{d-c}{b-a})(x-a) + c\ . \tag{5.13}$$

$$w^1(s_i^x) = (\frac{\overline{w}-\underline{w}}{p_i^{max}})p_i^x + \underline{w}\ . \tag{5.14}$$

Le critère du nombre de particules est clairement incontournable et doit sans doute avoir la plus forte pondération de tous les critères, car il permet d'exprimer la répartition de la charge de calcul relatives aux particules. Si le traitement des particules est dans la majorité des cas dominant en termes de temps de calcul, il ne faut toutefois pas exclure le traitement des mailles dans la considération de la répartition de la charge de calcul. Il est donc pertinent d'inclure dans la pondération des informations concernant les mailles. C'est pourquoi nous proposons de considérer le nombre de mailles par sommet comme deuxième critère.

Critère n° 2 : densité de mailles

Chaque maille stockée en mémoire représente une consommation mémoire. La réduction de la consommation mémoire est l'un de nos objectifs. Il est donc important de considérer le nombre de mailles nécessitant un stockage mémoire. De plus, chaque maille nécessite un traitement indépendamment du nombre de particules (mise à jour de listes, etc.). Nous devons prendre en compte le coût de ce traitement. Tous les sommets représentent le même nombre de mailles. En revanche, les mailles n'ayant pas de particules n'impliquent pas les mêmes charges de calcul et les mêmes volumes de donnée que les mailles chargées en particules. Nous proposons de considérer dans le calcul de la densité de mailles uniquement les mailles chargées en particules. Soit dm_i^x la densité de mailles, chargées en particules, représentées par le sommet $s_i^x \in S_i'$. dm_i^x est obtenu au moyen de l'équation 5.15. Soit r le degré de contraction d'un sommet. Nous avons $dm_i^x \in]0, r]$. Nous proposons de déterminer le poids, selon ce critère, en utilisant le même procédé que pour le premier critère, à savoir une transformation affine de $]0, r]$ vers $[\underline{w}, \overline{w}]$.

$$dm_i^x = \#\{m \in s_i^x, \rho_m \neq 0\}\ . \tag{5.15}$$

Nous souhaitons pondérer un sommet selon le deuxième critère de manière proportionnelle au nombre de mailles chargées en particules qui sont représentées par ce sommet. Nous obtenons alors directement le poids par une transformation affine au moyen de l'équation 5.16.

$$w^2(s_i^x) = \left(\frac{\overline{w} - \underline{w}}{r}\right) dm_i^x + \underline{w} \,. \tag{5.16}$$

L'expression des charges de calcul et des données de simulation par la pondération est notre objectif initial. Les critères que nous avons proposés jusqu'à maintenant permettent de pondérer un sommet en fonction de ses caractéristiques de calcul et de donnée à une itération t donnée. Nous n'avons toutefois pas considéré de critères de pondération permettant de prendre en compte l'aspect itératif. En effet, la constitution du sous-graphe de l'itération $t+1$ est le résultat du sous-graphe de l'itération t et des évènements, liés aux particules, survenus lors de l'itération t.

D'une itération à l'autre, les particules peuvent migrer d'une maille à une autre et donc d'un sommet à un autre. Ces migrations peuvent être la cause de créations de sommets supplémentaires. Ces créations impliquent des charges de calcul et des consommations mémoire supplémentaires. C'est pourquoi nous proposons de considérer dans la pondération l'adjacence de chaque sommet.

Critère n° 3 : adjacence

Chaque sommet appartenant à S_i' peut être lié par une arête avec un autre sommet. Nous avons défini une catégorisation de ces arêtes et une pondération associée telle que le poids d'une arête désigne un surcoût potentiel en nombre de sommets. Nous proposons de prendre en compte ce surcoût potentiel également dans la pondération des sommets. Pour cela, nous proposons de considérer dans la pondération d'un sommet les types d'arêtes, dont le sommet est l'une des composantes, et, en particulier, les arêtes appartenant à A''.

Nous avons défini deux catégories d'arêtes. A'' définit les arêtes impliquant une potentielle création de sommets. Lorsque les arêtes, dont un sommet est l'une des extrémités, appartiennent majoritairement à cette catégorie, le sommet est considéré comme étant indésirable selon le critère d'adjacence. La pondération d'un sommet selon ce critère est donc proportionnelle au nombre d'arêtes appartenant à A'.

Le sommet s_i^x est le sommet de coordonnées x dans le sous-graphe de la ressource i. $\#\{a = (s^x, s^y) \in A\}$ désigne le nombre d'arêtes connectées à s^x. Nous obtenons alors ra_i^x, la proportion d'arêtes n'entraînant pas de créations de sommets, dont l'instance s_i^x est l'une des composantes, au moyen de l'équation 5.17. Par nature, $ra_i^x \in [0,1]$. Nous proposons de déterminer le poids selon ce critère en utilisant une transformation affine de $[0,1]$ vers $[\underline{w}, \overline{w}]$.

$$ra_i^x = \frac{\#\{a = (s_i^x, s_i^y), a \in A'\}}{\#\{a = (s^x, s^y) \in A\}} \,. \tag{5.17}$$

Nous souhaitons pondérer un sommet selon le troisième critère en conservant la proportionnalité. Nous obtenons alors directement le poids par une transformation affine au moyen de l'équation 5.18.

$$w^3(s_i^x) = (\overline{w} - \underline{w}) ra_i^x + \underline{w} \,. \tag{5.18}$$

Ce troisième et dernier critère permet de définir un sommet selon d'autres propriétés que simplement la charge de calcul et les données représentées. Ce critère permet en particulier de prendre en compte l'aspect itératif de l'application. Nous proposons maintenant d'illustrer le processus de pondération par un exemple. Dans cet exemple, nous ne considérerons pas la pondération des sommets $s \in \overset{\circ}{S}_i$.

Exemple

Nous définissons les coefficients de pondérations suivant : $c^1 = 60$, $c^2 = 5$ et $c^3 = 35$. Le critère de pondération désignant le nombre de particules par sommet est celui représentant le mieux la charge de calcul et les données de chaque sommet. C'est pourquoi il sera majoritaire. Le critère d'adjacence est également important, dans la mesure où il permet de quantifier le volume de création de sommets. Nous définissons l'intervalle de pondération tel que $\underline{w} = 1$ et $\overline{w} = 7$. La figure 5.1 donne une représentation du graphe et du maillage.

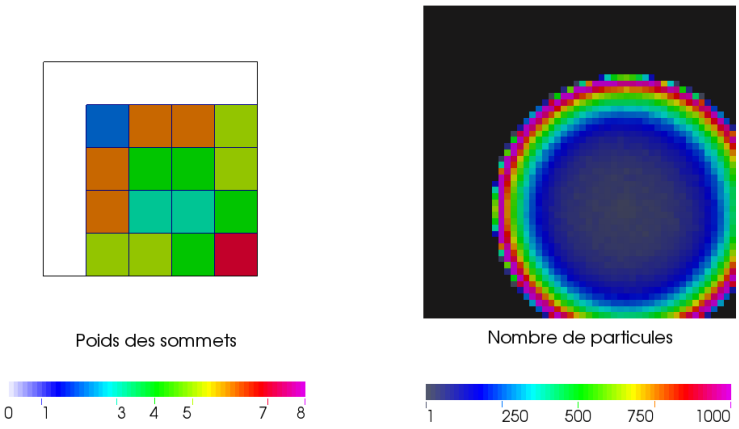
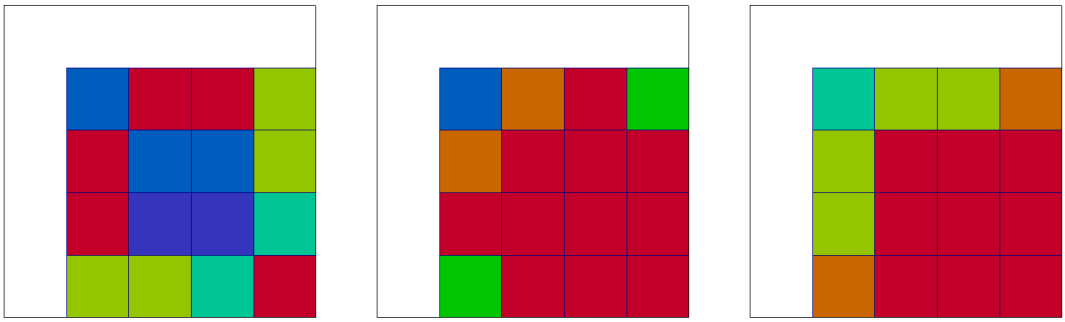


FIGURE 5.1 – Représentation du graphe et du maillage.

La pondération de chaque sommet est le résultat d'une moyenne pondéréee selon trois critères : la densité de particules, la densité de mailles chargées en particules et le voisinage. Afin de comprendre la pondération présentée en figure 5.1, nous proposons d'étudier séparément les pondérations selon tous les critères. la figure 5.2 présente toutes les pondérations partielles. La sous-figure 5.2a présente la pondération en fonction de la densité de particules. La corrélation entre les poids des sommets et la répartition des particules est évidente. La sous-figure 5.2b donne la pondération en fonction de la densité de mailles. Nous observons que les sommets placés au centre du graphe ont un poids maximal. À l'inverse, les sommets placés en "bordure" de graphe sont pour la plupart sous-pondérés. Cela s'explique par le fait que les sommets placés en "bordure" viennent d'être créés et certaines de leurs mailles ne sont pas encore chargées en particules. Enfin, la sous-figure 5.2c présente la pondération en fonction de l'adjacence. Nous constatons que, comme prévu, les sommets ayant peu de sommets adjacents sont sous-pondérés. Nous remarquons que les sommets placés au centre du graphe sont davantage pondérés sur le deux derniers critères mais possèdent à l'arrivée des poids assez faible. Cette propriété est la conséquence dans le fait que ces mêmes sommets sont largement sous-pondérés selon le premier critère. Or, ce dernier possède un coefficient de pondération plus élevé que les deux autres critères réunis. Cet exemple illustre parfaitement l'importante des coefficients de pondérations dans le calcul de la moyenne.

5.2.4 Conclusion

Nous avons défini une méthode de pondération multi-critères permettant de caractériser les sommets selon un ensemble de propriétés. Cette pondération a pour objectif d'optimiser la sélection des sommets à transférer quand cela s'avère nécessaire. Les opérations de transfert s'effectuent entre un sous-graphe et des sous-graphes voisins. La pertinence du choix des voisins est une donnée



(a) Densité de particules. (b) Densité de mailles. (c) Voisinage.

FIGURE 5.2 – Pondérations selon les différents critères.

importante. Nous proposons donc de définir maintenant une méthode de sélection des sous-graphes voisins.

5.3 Sélections de voisins par affinités

Nous proposons d'introduire la notation N_i, définissant l'ensemble des ressources avec lesquelles une ressource, notée i, peut réaliser des opérations de régulation de charge et de donnée. Nous appelons le sous-graphe de la ressource $n \in N_i$ un sous-graphe voisin du sous-graphe G'_i. Nous proposons que chaque ressource puisse accéder aux sous-graphes voisins.

Les ressources dont les sous-graphes sont voisins sont amenées à échanger des particules et des mailles. Dans la mesure où $\#N_i < N$, la question de la composition de N_i se pose alors naturellement. En effet, les sous-graphes voisins doivent être choisis dans l'optique de faciliter autant que possible les opérations de régulation de charge et de donnée.

Nous devons donc proposer une méthode permettant de déterminer la composition de l'ensemble des voisins de chaque sous-graphe dans le but d'optimiser les transferts de sommets.

5.3.1 Principe

Le choix des sous-graphes voisins peut être réalisé de plusieurs façons. Une approche naïve consisterait à choisir aléatoirement, pour chaque sous-graphe, des sous-graphes voisins. Nous obtiendrons avec cette méthode un choix de sous-graphes voisins loin d'être optimal. Un choix optimal serait obtenu en échangeant entre toutes les ressources toutes les informations nécessaires, et ce à chaque itération. En effet, il serait alors possible de calculer la meilleure combinaison possible. Toutefois, cette méthode serait extrêmement coûteuse à la fois en volume de donnée et en temps de traitement. Nous proposons donc une méthode de sélection de voisins nécessitant le moins d'informations possible mais aussi le moins de calcul possible.

La définition d'un ensemble de voisins dont le nombre d'éléments est restreint permet d'éviter les opérations de régulation entre toutes les ressources afin d'éviter un surcoût en communication. Étant donné que le nombre de dimensions du domaine définit le nombre d'interfaces entre les sous-domaines et que le degré de réplication détermine le nombre moyen de ressources se partageant un sommet donné, nous choisissons le nombre de voisins en fonction du nombre de dimensions du domaine et du degré de réplication. Afin d'éviter des déséquilibres en termes de communication,

nous avons fait le choix d'imposer si possible que $\forall i, j \ \#N_i = \#N_j$. Dans l'optique d'éviter que le coût de communication augmente proportionnellement au nombre de ressources, nous proposons de maintenir constant le nombre de voisins maximal N'. Nous avons alors $\forall i \ \#N_i \leq N'$.

Le choix des sous-graphes appartenant à N_i doit prendre en compte le fait que les sous-graphes ainsi sélectionnées réaliseront avec le sous-graphe G'_i des échanges de sommets. Le transfert d'un sommet d'un sous-graphe à l'autre est meilleur lorsque le sous-graphe de destination possède déjà une instance de ce sommet. En effet, aucun sommet supplémentaire ne sera créé dans le sous-graphe de destination : il s'agira simplement d'une mise à jour des propriétés du sommet. Nous admettons donc que deux sous-graphes ayant beaucoup d'instances de sommets en commun (mêmes coordonnées) ont tout intérêt a être voisins. Nous proposons donc de créer les relations de voisinage en fonction du nombre d'instances de sommets en commun. Ces relations ne peuvent être construites sans la connaissance de tous les sous-graphes. Les graphes sont une abstraction haut niveau des données de simulation : un graphe représente un volume de donnée assez restreint. En revanche, pour obtenir une vision globale des sous-graphes, nous devons échanger N sous-graphes. Le volume de donnée à transférer risque alors de devenir une sérieuse limitation lorsque N augmente. Nous proposons donc de réduire le volume des données à s'échanger en utilisant un modèle d'un graphe quotient.

5.3.2 Modèle graphe quotient

Un graphe quotient est une représentation réduite d'un graphe. La taille du graphe quotient doit être choisie de telle sorte que le volume de donnée associé soit le plus faible possible tout en garantissant le moins de perte d'informations possible. Soit $G^*_i = (S^*_i, A^*_i)$, le graphe quotient de la ressource i. Étant donné r' le ratio entre la taille du graphe et la taille du graphe quotient en nombre de sommets, nous obtenons le nombre de sommets du graphe quotient au moyen de l'équation 5.19.

$$\#S^*_i = \frac{\#S'_i}{r'} \ . \tag{5.19}$$

Étant donné D le nombre de dimensions du graphe, nous pouvons trouver un ensemble de ratio, entre la taille du graphe et la taille du graphe quotient en nombre de sommets, sur chacune des dimensions tel que le définit l'équation 5.20.

$$\exists \{r'_1, r'_2, ..., r'_d\}, r'_1 r'_2 ... r'_d = r' \ . \tag{5.20}$$

L'agglomération des sommets du graphe G' que nous proposons est cartésienne. En effet, nous choisissons de grouper les sommets entre eux en fonction de leurs coordonnées. La construction des sommets de G^* en utilisant les coordonnées permet de définir la localisation des sommets de G'. Il suffit d'un sommet appartenant à G' dans un intervalle de coordonnées donné pour que la création du sommet associé du graphe quotient ait lieu. Pour garantir une représentation fidèle du graphe, les sommets du graphe quotient doivent être construits différemment selon le nombre de sommets de G' qu'ils représentent. Nous proposons donc de définir comme poids de sommet du graphe quotient le nombre de sommets du graphe représentés. Nous obtenons alors $w^*(s^x_i)$, le poids du sommet $s^x_i \in S^*_i$, au moyen de l'équation 5.21. La figure 5.3 donne un exemple de création d'un graphe quotient.

$$w^*(s^x_i) = \#\{s^y_i \in S', y = (y_1, y_2, ..., y_d) \in X^d, \forall k \in [1, d] \ y_k \in [x_k r'_k, (x_k + 1) r'_k [\} \ . \tag{5.21}$$

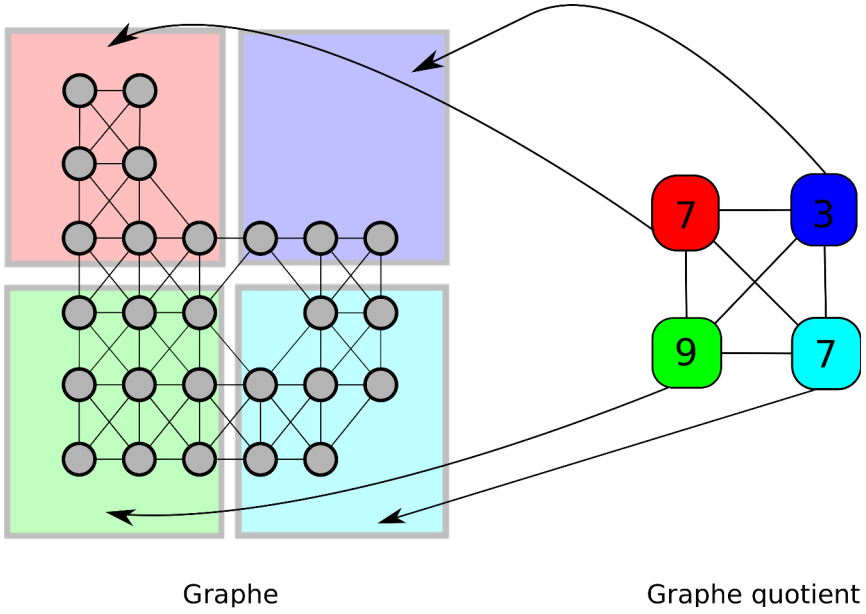


FIGURE 5.3 – Exemple d'un graphe quotient cartésien.

Nous constatons qu'en utilisant le modèle d'un graphe quotient illustré en figure 5.3, deux sommets adjacents dans G' peuvent ne pas être représentés par le même sommet dans G^*. Cette propriété pose problème dans la mesure où la relation entre les sommets adjacents peut être perdue. Nous proposons donc d'élargir l'intervalle de coordonnées de manière à créer un recouvrement entre les sommets adjacents du graphe quotient. Nous obtenons alors $w^*(s_i^x)$, le nouveau poids du sommet $s_i^x \in S_i^*$, au moyen de l'équation 5.22. La figure 5.4 présente alors le nouvel graphe quotient, en utilisant le même exemple que la figure 5.3.

$$w^*(s_i^x) = \#\{s_i^y \in S', y = (y_1, y_2, ..., y_d) \in X^d, \forall k \in [1,d]\ y_k \in [x_k r'_k - 1, (x_k+1)r'_k + 1[\}\ . \tag{5.22}$$

Chaque ressource possède son graphe quotient G_i^* représentant le graphe G'_i. Nous proposons une méthode permettant de déterminer les relations d'affinités entre deux graphe quotients. Dans ce but, nous proposons une méthode de calcul de score d'affinités.

5.3.3 Scores d'affinités

Nous désignons deux ressources ayant beaucoup d'affinités comme étant deux ressources dont les sous-graphes ont beaucoup d'instances de sommets en commun. Nous pouvons appliquer le même raisonnement avec leurs graphe quotients. En effet, des instances communes entre deux graphe quotients impliquent des instances communes entre les sous-graphes sous-jacents. Nous proposons de quantifier les affinités entre deux sous-graphes en comparant leurs graphe quotients respectifs. En particulier, nous nous focalisons sur les instances communes. Les poids des sommets du graphe quotient représentent un nombre de sommets du graphe sous-jacents. Nous proposons donc de calculer un score d'affinités en fonction des poids des sommets que les graphe quotients ont en commun.

Soit $\overset{*}{X}^d$ l'ensemble des coordonnées possibles des sommets du graphe quotient. Le calcul du score d'affinités entre le graphe quotient G_i^* et le graphe quotient G_j^* est donné par l'équation 5.23.

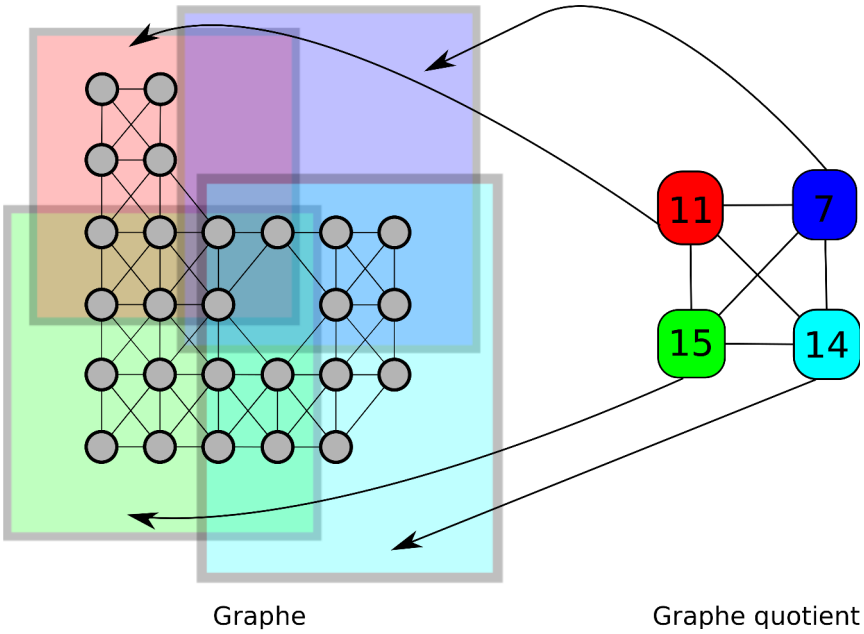


FIGURE 5.4 – Exemple d'un graphe quotient cartésien avec recouvrement.

$$ScoreAff(i,j) = \sum MIN(w^*(s_i^x), w^*(s_j^x)), \forall x \in \overset{*}{X}^d . \tag{5.23}$$

Il s'agit donc d'une somme des poids minimaux des instances de sommets communes entre les graphe quotients. Notons que le score $ScoreAff(i,i)$ est une métrique intéressante permettant de déterminer approximativement la taille du graphe quotient G_i^*. Précisons également que cette méthode de calcul donne $ScoreAff(i,j) = ScoreAff(j,i), \forall i,j$.

Les scores obtenus permettent de déterminer quelles sous-graphes ont des affinités. Nous proposons maintenant une méthode de sélection de sous-graphes basée sur les scores d'affinités.

5.3.4 Algorithme de sélection de voisins

L'objectif de l'algorithme de sélections de voisins est de permettre à chaque ressource de choisir les ressources avec lesquelles leurs sous-graphes respectifs ont le plus d'affinités. Nous rappelons que tous les sous-graphes doivent posséder approximativement le même nombre de voisins. De plus, nous devons faire en sorte qu'aucun sous-graphe ne se retrouve avec un voisinage avec lequel elle n'aurait que très peu d'affinités. Nous proposons donc d'utiliser une approche tour par tour pour permettre à chaque ressource de choisir chacune à son tour un nouveau voisin.

L'approche tour par tour basique présente l'inconvénient que la ressource dont le tour arrive en dernier n'aura potentiellement qu'un ensemble de choix possibles assez restreint. Il est donc évident que nous devons choisir un ordre de priorité qui s'adapte au fur et à mesure des sélections de voisins. De plus, les graphes n'ont pas nécessairement les mêmes tailles. Or, plus le nombre de sommets est élevé plus la ressource devra en envoyer. La taille du graphe quotient étant fonction de la taille du graphe, il est alors pertinent de considérer une ressource i, ayant un score $ScoreAff(i,i)$ élevé, prioritaire dans le choix des voisins.

Nous définissons par qualité de voisin d'une ressource la probabilité que la ressource puisse échanger facilement des sommets avec ce voisin. Nous proposons de tenir compte de la qualité

des voisins choisis dans l'estimation de la priorité. Pour cela, nous proposons de mettre à jour $ScoreAff(i,i)$ lorsque i a choisi un voisin en fonction de la qualité de celui-ci. La qualité d'un voisin est symbolisé par le score d'affinités entre le sous-graphe et ce voisin. Ce score est une estimation du nombre de sommets en commun entre les graphes des ressources. Nous supposons que certains de ces sommets seront effectivement transférés. Nous définissons alors $\tau^*, \tau^* \in [0,1]$, comme l'estimation du ratio de sommets pouvant effectivement être transférés. Nous proposons donc de mettre à jour $ScoreAff(i,i)$ et $ScoreAff(j,j)$ lorsque G'_i a choisi comme voisin G'_j en déduisant $\tau^*\ ScoreAff(i,j)$.

L'algorithme 9 présente l'algorithme de sélection des voisins. En premier lieu, chaque ressource doit commencer par créer son graphe quotient (ligne 1). Nous devons ensuite échanger l'ensemble des graphe quotients afin d'obtenir une vision globale des graphe quotients (ligne 2). Nous terminons ensuite la phase d'initialisation.

Algorithme 9 : Sélection des voisins.

```
   /* Créations des graphe quotients                                   */
1  Creation_HyperGraphes( G*, G')
   /* Échanges des graphe quotients entre les ressources               */
2  Echanges_HyperGraphes( G*, N)
   /* Nous créons l'ensemble des relations de voisinage pour tous les
    sous-graphes                                                       */
3  Npaires ← N'N / 2
4  N_i ← ∅, ∀i
5  choisi_(i,j) ← Faux, ∀i, j, i ≠ j
   /* Les sous-graphes ne peuvent pas se choisir eux-mêmes             */
6  choisi_(i,i) ← Vrai, ∀i
   /* Tant que les voisinages ne sont pas complets                     */
7  tant que Npaires > 0 faire
       /* Sélection du sous-graphe ayant la plus haute priorité        */
8      i ← (k, #N_k < N', ScoreAff(k,k) > ScoreAff(p,p) ∀p)
       /* Sélection du sous-graphe avec lequel il a le plus d'affinités */
9      j ← (k, choisi_(i,j) = Faux, #N_k < N', ScoreAff(i,k) > ScoreAff(i,p) ∀p)
       /* Si un sous-graphe voisin a été trouvée                       */
10     si j ≠ ∅ alors
           /* Mise à jour des voisins                                  */
11         N_i ← N_i ∪ j
12         N_j ← N_j ∪ i
           /* La paire ne pourra plus être formée                      */
13         choisi_(i,j) = Vrai
14         choisi_(j,i) = Vrai
           /* On met à jour la priorité                                */
15         ScoreAff(i,i) ← ScoreAff(i,i) − τ* ScoreAff(i,j)
16         ScoreAff(j,j) ← ScoreAff(j,j) − τ* ScoreAff(j,i)
17     fin
18     Npaires ← Npaires − 1
19 fin
```

La boucle de la ligne 7 permet de créer toutes les paires de voisins pour tous les sous-graphes. À chaque itération de boucle, nous commençons par chercher le sous-graphe le plus prioritaire que nous notons i (ligne 8). Ensuite, nous cherchons le sous-graphe ayant le plus d'affinités avec i (ligne 9). Notons qu'il est possible qu'il n'y ait plus aucun voisin disponible. Si nous avons trouvé un voisin, nous créons alors la paire de voisinage (lignes 11 et 12). Nous désactivons alors la possibilité pour les sous-graphes de se choisir de nouveau (lignes 13 et 14). Enfin, nous mettons

à jour les priorités (lignes 15 et 16).

Cette méthode permet d'obtenir des relations de voisinages en fonction des affinités et des priorités. Cette méthode a plusieurs limitations. La première réside dans le fait que le voisinage est fixe. Cette attribution statique est limitée dans la mesure où l'on utilise potentiellement assez vite tous les transferts de sommets envisageables. Il serait donc judicieux de pouvoir changer les voisins d'une itération à l'autre. La deuxième limitation concerne les priorités. En effet, l'ordre de priorité obtenu est très limité en taille puisque le nombre de voisins est très restreint. Certains sous-graphes peuvent alors ne jamais devenir prioritaires malgré un nombre de sommets conséquent. Nous proposons alors une méthode capable de créer des relations de voisinages différentes entre les itérations.

5.3.5 Adaptation inter-itérations

Permettre aux sous-graphes de changer la composition du voisinage d'une itération à l'autre est judicieux. Toutefois, imposer le changement pourrait être contre-productif. En effet, lorsque deux sous-graphes ont beaucoup d'affinités, il parait judicieux de conserver leur relation de voisinage pendant de nombreuses itérations. Nous proposons alors de n'imposer aucune règle supplémentaire pour le choix d'un voisin. Nous proposons d'étendre l'ordre prioritaire, non plus seulement au nombre de voisins, mais également au nombre d'itérations. Nous proposons de calculer un voisinage sur X itérations. Calculer un voisinage sur un nombre d'itérations trop élevé amènerait des solutions trop approximatives.

Soit $t \in [0, X-1]$. Nous notons N_i^t le $t^{ème}$ voisinage du sous-graphe i. Nous avons alors $N_i = N_i^t$ lorsque $iter\_courante \equiv t \pmod X$. L'algorithme de sélections de voisins modifié est présenté en tant qu'algorithme 10. Cet algorithme sera illustré par exemple dans la sous-section suivante.

5.3.6 Exemple

Soit $N = 6$, $X = 3$, $N' = 3$ et $\tau^* = 0.1$. La figure 5.1 donne une représentation des 6 graphes correspondant aux 6 ressources. Nous avons les graphes des ressources 0, 1 et 2 (dans l'ordre en partant de la gauche) sur la première ligne et les graphes des ressources 3, 4 et 5 (dans l'ordre en partant de la gauche) sur la deuxième ligne. Nous pouvons constater que les graphes de la première ligne ont beaucoup de sommets en commun. Nous pouvons constater la même chose pour les graphes de la seconde ligne.

Les scores d'affinités des graphe quotients, construits à partir de ces graphes, sont donnés dans la table 5.1. Nous avons sur la diagonale les scores désignant la somme des poids de chacun des graphe quotients. Ces sommes donnent un ordre prioritaire, mentionné dans la table entre parenthèses précédé du caractère #. Nous notons que les ressources 0 et 2 sont nettement prioritaires par rapport à la ressource 4. Nous observons que les scores obtenus sont conformes aux impressions concernant les affinités entre les graphes.

Les voisinages après le premier tour de sélection sont donnés dans la table 5.2. Pour chaque ressource, les scores coloriés en verts correspondent aux voisins choisis. Sur la diagonale, nous affichons le nouveau score, et entre parenthèse, combien lui a été soustrait. Nous remarquons plusieurs choses. Nous observons que les sous-graphes de fortes priorités ont pu choisir les sous-graphes voisins avec lesquelles ils ont le plus d'affinités (scores d'affinité élevés). Ce résultat concorde parfaitement avec ce que nous souhaitons obtenir. En revanche, nous remarquons que la ressource 4 n'a choisi qu'un seul voisin. En effet, il semblerait que, au moment de choisir des voisins, tous les autres sous-graphes étaient indisponibles (leurs voisinages étaient déjà complets). Ce phénomène

Algorithme 10 : Sélection itérative des sous-graphes voisins.

```
   /* Créations des graphe quotients                                   */
1  Creation_HyperGraphes( G*, G' )
   /* Échanges des graphe quotients entre les ressources               */
2  Echanges_HyperGraphes( G*, N)
   /* Nous créons l'ensemble des relations de voisinage pour tous les
    sous-graphes                                                       */
3  t ← 0
4  tant que t ≠ X faire
5      Npaires ← N'N / 2
6      N_i^t ← ∅, ∀i
7      choisi_(i,j) ← Faux, ∀i, j, i ≠ j
       /* Les sous-graphes ne peuvent pas se choisir eux-mêmes         */
8      choisi_(i,i) ← Vrai, ∀i
       /* Tant que les voisinages ne sont pas complets                 */
9      tant que Npaires > 0 faire
           /* Sélection du sous-graphe ayant la plus haute priorité    */
10         i ← (k, #N_k < N', ScoreAff(k,k) > ScoreAff(p,p) ∀p)
           /* Sélection du sous-graphe avec lequel il a le plus d'affinités */
11         j ← (k, choisi_(i,j) = Faux, #N_k < N', ScoreAff(i,k) > ScoreAff(i,p) ∀p)
           /* Si un sous-graphe voisin a été trouvée                   */
12         si j ≠ ∅ alors
               /* Mise à jour des voisins                              */
13             N_i^t ← N_i^t ∪ j
14             N_j^t ← N_j^t ∪ i
               /* La paire ne pourra plus être formée                  */
15             choisi_(i,j) = Vrai
16             choisi_(j,i) = Vrai
               /* On met à jour la priorité                            */
17             ScoreAff(i,i) ← ScoreAff(i,i) − τ* ScoreAff(i,j)
18             ScoreAff(j,j) ← ScoreAff(j,j) − τ* ScoreAff(j,i)
19         fin
20         Npaires ← Npaires − 1
21     fin
22     t ← t + 1
23 fin
```

scoreAff	0	1	2	3	4	5
0	84 (#1)	67	76	45	44	48
1	67	73 (#3)	73	41	40	44
2	76	73	83 (#2)	42	41	45
3	45	41	42	68 (#4)	55	59
4	44	40	41	55	57 (#6)	57
5	48	44	45	59	57	65 (#5)

TABLE 5.1 – Matrice des scores d'affinités avant les sélections des voisins.

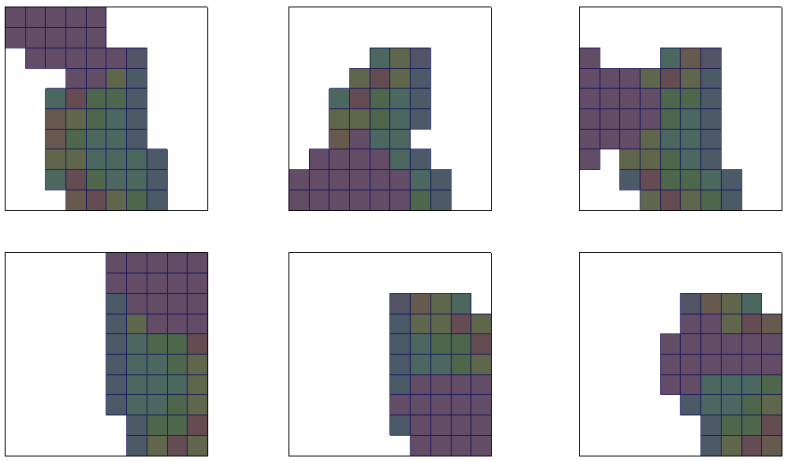


FIGURE 5.5 – Représentation des graphes de 6 ressources.

scoreAff	0	1	2	3	4	5
0	64.9 (-19,1)	67	76	45	44	48
1	67	54.9 (-18,1)	73	41	40	44
2	76	73	63.6 (-19.4)	42	41	45
3	45	41	42	52.5 (-15.5)	55	59
4	44	40	41	55	51.5 (- 5.5)	57
5	48	44	45	59	57	49.8 (-15.2)

TABLE 5.2 – Sélections des voisins (t=0).

peut être assez dérangeant dans la mesure où des relations de voisinage sont perdues. Toutefois, il faut nuancer par rapport au fait que le sous-graphe en question est largement moins prioritaire que les autres. La nécessité de transférer des sommets sera donc moins importante. Enfin, nous observons que les scores sur la diagonale se sont sensiblement uniformisés.

Les voisinages après le deuxième tour de sélections sont donnés dans la table 5.3. Pour commencer, nous remarquons que, contrairement au tour précédent, tous les sous-graphes ont formé un voisinage complet. Enfin, nous observons que le processus d'uniformisation des scores sur la diagonale s'est poursuivi. En effet, les écarts entre les sous-graphes prioritaires et les autres se sont largement réduits.

Les voisinages après le troisième et dernier tour de sélections sont donnés dans la table 5.4. Nous constatons que les voisinages sont strictement identiques à ceux du tour précédent. Cela

scoreAff	0	1	2	3	4	5
0	45.8 (-19,1)	67	76	45	44	48
1	67	36.9 (-18,0)	73	41	40	44
2	76	73	44.5 (-19.1)	42	41	45
3	45	41	42	36.9 (-15.6)	55	59
4	44	40	41	55	36.3 (-15.2)	57
5	48	44	45	59	57	33.4 (-16.4)

TABLE 5.3 – Sélections des voisins (t=1).

scoreAff	0	1	2	3	4	5
0	26.7 (-19,1)	67	76	45	44	48
1	67	18.9 (-18,0)	73	41	40	44
2	76	73	25.4 (-19.1)	42	41	45
3	45	41	42	21.3 (-15.6)	55	59
4	44	40	41	55	21.1 (-15.2)	57
5	48	44	45	59	57	17 (-16.4)

TABLE 5.4 – Sélections des voisins (t=2).

s'explique par le fait que nous avons peu de sous-graphes et peu de voisins par sous-graphe. En effet, le nombre de voisinages possibles n'est pas extrêmement élevé. Or, les sous-graphes prioritaires ont pu choisir les sous-graphes avec lesquelles ils ont le plus d'affinités. Cela ne laisse donc que très peu de possibilités ensuite. Le fait que les voisinages soient identiques d'une itération à l'autre ne pose pas de problème, d'autant plus si ces derniers sont proches de l'optimal comme sur notre exemple. Cet exemple illustre bien que l'ordre prioritaire de sélection joue un rôle décisif dans le processus de sélections des voisins.

5.3.7 Conclusion

Nous avons donc proposé une méthode permettant d'obtenir les N_i pour tout i. Les voisinages des sous-graphes sont déterminées en fonction des affinités entre leurs graphes respectifs. En effet, les échanges de sommets entre deux sous-graphes sont optimaux lorsque leurs graphes ont beaucoup d'affinités. Ces échanges de sommets ont pour objectifs de limiter les données par ressource et de répartir la charge de calcul. Nous allons maintenant présenter une méthode de limitation du nombre de mailles.

5.4 Limitation du nombre de mailles

La restriction en nombre de sommets de chaque sous-graphe implique directement une limitation du nombre de mailles. La limitation de la consommation mémoire relative aux mailles peut donc être obtenue en limitant le nombre de sommets par sous-graphe. Étant donné R le degré de réplication, nous définissons $\#S'_{max}$ le nombre maximal de sommets par sous-graphe au moyen de l'équation 5.24.

$$\#S'_{max} = R\frac{M}{N}\,. \tag{5.24}$$

Le transfert de sommets d'un sous-graphe à un autre sous-graphe permet de réduire le nombre de sommets du sous-graphe source. Toutefois, ces transferts doivent être organisés pour être efficaces. En effet, si ces transferts ont pour conséquence de grossir de manière démesurée le sous-graphe destination, nous perdrions tout intérêt à réaliser ces transferts. Nous proposons donc une méthode de limitation du nombre de sommets de chaque sous-graphe.

5.4.1 Introduction

$\#S'_{max}$ désigne le nombre maximal de sommets qu'un sous-graphe donné est autorisé à posséder. L'idée est donc d'envoyer autant de sommets que nécessaire afin de respecter cette contrainte. La règle permettant de décider combien de sommets un sous-graphe doit transférer à un

autre sous-graphe pour garantir le respect de la contrainte n'est pas trivial. En effet, il est difficile d'anticiper avec précision le nombre de sommets qui seront créés, par la migration des particules, au cours de l'itération. De plus, un sous-graphe donné ne connaît pas à l'avance les sommets qu'il recevra de ses voisins. Nous proposons une approche de seuil qui consiste à envoyer des sommets tant que le nombre de sommets est supérieur à ce seuil. Nous proposons de définir $\#S'_{seuil}$ comme étant le nombre de sommets à partir duquel nous considérons que nous devons envoyer des sommets pour respecter la contrainte. Étant donné τ_s, avec $0 < \tau_s < 1$, une marge par rapport à la contrainte en nombre de sommets, nous obtenons $\#S'_{seuil}$ au moyen de l'équation 5.25.

$$\#S'_{seuil} = (1 - \tau_s)\#S'_{max} \ . \tag{5.25}$$

Dans l'optique de limiter le nombre de transferts de sommets non pertinents, nous proposons d'instaurer des critères. Ces critères ont pour objectif de déterminer quels transferts de sommets sont pertinents et lesquels ne le sont pas.

5.4.2 Critères de transfert

Chaque sous-graphe possède ses propres propriétés (taille du graphe, connexité du graphe, caractéristiques des sommets, etc.). L'approche consistant à transférer des sommets d'un sous-graphe à un autre se doit de tenir compte de ces différentes propriétés. L'une de ces propriétés est la taille du sous-graphe. Nous proposons donc de considérer la taille de chaque sous-graphe comme critère de transfert de sommets.

Nombre de sommets

Les sous-graphes n'ont pas nécessairement le même nombre de sommets. Par conséquent, les sous-graphes n'ont pas la même nécessité d'envoyer des sommets. En effet, une ressource i telle que $\#S'_i = \#S'_{max}$ doit absolument être prioritaire par rapport à une ressource n telle que $\#S'_n = \#S'_{seuil}$. Nous définissons r_i^S quantifiant la nécessité d'envois de sommets de la ressource i. r_i^S est défini par l'équation 5.26.

$$r_i^S = \frac{\#S'_i}{\#S'_{max}} \ . \tag{5.26}$$

Nous proposons de prendre en compte r_i^S et r_n^S dans la décision d'envoyer un sommet du sous-graphe de la ressource i vers le sous-graphe de la ressource n. Nous définissons $r_{(i,n)}^S \in [-1, 1]$, permettant de déterminer quelle ressource entre i et n est prioritaire sur l'envoi de sommets, au moyen de l'équation 5.27. Si $r_{(i,n)}^S = -1$, cela signifie que n est largement prioritaire sur i. Si $r_{(i,n)}^S = 1$, c'est exactement l'inverse.

$$r_{(i,n)}^S = \frac{r_i^S - r_n^S}{\tau_s} \ . \tag{5.27}$$

Chaque sommet a son propre poids. Les instances de sommets communes entre deux sous-graphes voisins n'ont pas forcément les mêmes poids. Nous proposons de considérer comme critère de transfert le poids des instances de sommets du sous-graphe voisin.

Poids de sommets

Nous rappelons que les poids attribués aux sommets permettent de quantifier la charge de calcul et les données qui y sont associés. Un sommet de faible poids implique donc le transfert de peu de donnée et de charge de calcul. Nous proposons donc de transférer les sommets de faibles poids en priorité. En appliquant cette stratégie, un sommet dont le poids est élevé a de fortes chances de ne pas être transféré. Par conséquent, si un sous-graphe et son sous-graphe voisin possèdent une instance de sommet en commun et que le poids de l'instance du sous-graphe voisin est élevé, le transfert du sommet est pertinent. Lorsque le poids du l'instance de sommet du sous-graphe distant est élevé, nous qualifions le sous-graphe distant de "bon candidat". Soit $r^W_{(x,n)}$, exprimant dans quelle mesure le sous-graphe de la ressource n est un bon candidat. Nous obtenons $r^W_{(x,n)}$ au moyen de l'équation 5.28.

$$\begin{cases} Si\ w'(s^x_n) = 0,\ r^W_{(x,n)} & = -1\ ; \\ Si\ w'(s^x_n) \neq 0,\ r^W_{(x,n)} & = \dfrac{\overline{w} - w'(s^x_n)}{\overline{w}}\ . \end{cases} \tag{5.28}$$

Une limitation du critère de choix en fonction des poids des sommets existe lorsque toutes les instances de sommets sont faiblement pondérés. En effet, dans ce cas, aucun sous-graphe ne serait un bon candidat. Nous devons donc ajouter un autre critère permettant de décider quel sous-graphe conservera quel sommet. Nous avons défini $\overset{\circ}{S}_i$ comme étant les sommets attribués statiquement à la ressource i afin de garantir au minimum une instance par sommet. Il s'agit d'un partitionnement des sommets du graphe global. Chaque sommet ayant ses propres coordonnées, nous proposons de prendre en compte la localisation de chaque sommet par rapport aux différents sous-graphes.

Localisation de sommets

L'attribution statique d'instances de sommets, permettant de garantir au minimum une instance par sommet, offre la possibilité de garantir le stockage mémoire d'une maille donnée sur au moins une ressource. Cela peut s'avérer nécessaire pour certaines applications Monte-Carlo. Il est donc pertinent pour une ressource de posséder les particules associées aux sommets $s \in \overset{\circ}{S}_i$ lorsqu'elles existent. Par conséquent, s'il existe une instance de sommet s de poids supérieur ou égal à $\underline{w}$ telle que $s \in \overset{\circ}{S}_i$, la ressource i a tout intérêt à posséder cette instance. L'idée est donc d'orienter l'attribution des instances de sommets en fonction du partitionnement.

Nous souhaitons qu'une ressource i conserve si possible les sommets $s \in \overset{\circ}{S}_i$. Nous avons mis en place une pondération spéciale pour les sommets en question. Étant donné que les particules se déplacent, la conservation des sommets voisins aux sommets $s \in \overset{\circ}{S}_i$ doit être aussi envisagée. Pour évaluer cette proximité, nous proposons d'évaluer une distance, entre chaque sommet $s \in S'_i$ et l'ensemble des sommets $s \in \overset{\circ}{S}_i$, en utilisant les coordonnées des sommets. Pour exprimer la localisation de tous les sommets $s \in \overset{\circ}{S}_i, \forall i$, nous proposons de procéder par le calcul des barycentres de chaque sous-graphe.

Le calcul d'un barycentre consiste en une moyenne pondérée de plusieurs coordonnées. Dans notre cas, les coordonnées sont celles des sommets $s \in \overset{\circ}{S}_i$. Nous considérons tous les éléments avec le même poids. Nous obtenons alors $\overset{\circ}{\underline{x}}_i$ le barycentre de $\overset{\circ}{S}_i$.

Nous avons exprimé, à l'aide de leur barycentre $\overset{\circ}{\underline{x}}_i$, l'ensemble des sommets $s \in \overset{\circ}{S}_i$. Nous

proposons maintenant de déterminer la proximité d'un sommet donné avec $\overset{\circ}{S}_i$, en calculant une distance entre ce sommet et $\overset{\circ}{\underline{x}}_i$. Nous utilisons pour cela les coordonnées du sommet et les coordonnées du barycentre. Nous notons $D(x, \overset{\circ}{\underline{x}}_i)$ la distance entre un sommet s^x et le barycentre du sous-graphe appartenant à la ressource i.

Soit i une ressource et n une ressource dont les sous-graphes sont voisins. Nous pouvons définir $r^D_{(i,n)}$, au moyen de l'équation 5.29, permettant de déterminer si un sommet s^x_i est plus proche $\overset{\circ}{\underline{x}}_i$ ou de $\overset{\circ}{\underline{x}}_n$.

$$\begin{cases} Si\ D(x, \overset{\circ}{\underline{x}}_i) \geq D(x, \overset{\circ}{\underline{x}}_n),\ r^D_{(x,i,n)} & = 1\ ; \\ Si\ D(x, \overset{\circ}{\underline{x}}_i) < D(x, \overset{\circ}{\underline{x}}_n),\ r^D_{(x,i,n)} & = -1\ . \end{cases} \tag{5.29}$$

Conclusion

Nous avons proposé un ensemble de critères dans l'optique de réglementer les transferts de sommets entre les sous-graphes. Il s'agit, désormais, de regrouper l'ensemble de ces critères. Pour cela, nous proposons de mettre en œuvre une méthode de calcul de score. Cette méthode consiste en une moyenne pondérée. Ce score permettra de déterminer si l'envoi d'un sommet donné vers un sous-graphe donné est pertinent ou non.

5.4.3 Calcul d'un score

Nous définissons une fonction $score(x, i, n)$ permettant de déterminer dans quelle mesure le sommet de coordonnées x peut être envoyé par la ressource i à la ressource n. Étant donné que $c^S \in \mathbb{R}$, $c^W \in \mathbb{R}$ et $c^D \in \mathbb{R}$ sont des coefficients strictement positifs tels que $c^S + c^W + c^D = 1$, l'équation 5.30 donne l'expression de la fonction $score$. Le choix des coefficients n'est pas trivial. Selon le cas test, il peut être pertinent d'adapter les coefficients pour obtenir une meilleure régulation de charge et de donnée. Néanmoins ces trois critères sont importants et complémentaires, leurs coefficients doivent donc être relativement proches.

$$score(x, i, n) = c^S r^S_{(i,n)} + c^W r^W_{(x,i,n)} + c^D r^D_{(x,i,n)}\ . \tag{5.30}$$

5.4.4 Limitation d'envois de particules

Même si la limitation du nombre de sommets n'a pas pour objectif d'améliorer la qualité de la distribution des particules, elle ne doit pas être dégradée de manière excessive. Concrètement, envoyer massivement des sommets fortement pondérés vers une ressource ayant déjà beaucoup de particules n'est clairement pas une solution acceptable. Une ressource ne doit donc envoyer qu'une petite quantité de particules et de manière équitable aux ressources avec qui elle peut communiquer. Soit $r^p_{max} \in [0, 1]$, le ratio maximal de particules pouvant être envoyé par une ressource. Nous admettons que, plus une ressource doit envoyer de sommets, plus elle enverra des particules. Nous proposons donc de définir r^p_i, le ratio de particules pouvant être envoyé par la ressource i à une autre ressource, tel que celui-ci dépende du nombre de sommets de la ressource i. Ainsi, nous proposons que r^p_i soit proportionnel à r^S_i. r^p_i est alors obtenu au moyen de l'équation 5.31. Soit $\overline{P}^n_i$ le nombre de particules concernées par un transfert de la ressource i vers la ressource n. $\overline{P}^n_i$ est soumis à l'équation 5.32.

$$r_i^p = \frac{r_i^S r_{max}^p}{N'} \,. \tag{5.31}$$

$$\overline{P}_i^n \leq r_i^p P_i \,. \tag{5.32}$$

5.4.5 Algorithme de limitation du nombre de sommets

L'algorithme de limitation du nombre de sommets est détaillé par l'algorithme 11. Nous envoyons des sommets tant que le seuil n'est pas atteint ou qu'il ne reste plus de sommets susceptibles d'être envoyés (ligne 2), en commençant par les sommets de poids faibles. À chaque itération de la boucle principale correspond un poids w donné. Nous parcourons donc tous les sommets du sous-graphe de poids w de la ressource (ligne 4). Pour chaque sommet sélectionné, nous parcourons la liste des sous-graphes potentiellement receveurs (ligne 6). Nous choisissons le sous-graphe ayant le score le plus élevé à condition que ce score soit positif (ligne 8). Dans l'hypothèse où un sous-graphe destinataire a été trouvé, nous pouvons procéder au transfert (lignes 14 à 16). Si le seuil est atteint après le transfert d'un sommet, nous interrompons la boucle principale (ligne 17). Si à la fin d'une itération de boucle principale, le seuil n'est toujours pas atteint, nous recommençons le procédé avec les sommets de poids suivant (ligne 22).

La limitation du nombre de sommets que nous proposons n'engendre pas de déséquilibre excessifs en termes de particules entre les ressources. Cependant, l'équilibre n'est pas garanti. C'est pourquoi nous proposons maintenant des mécanismes d'équilibrages dynamiques de la répartition de la charge de calcul et plus particulièrement des particules.

5.5 Équilibrage de charge de calcul

Les mécanismes d'équilibrages de charge que nous proposons sont invoqués après l'algorithme de limitation du nombre de sommets. Les transferts locaux réalisés par les autres ressources pendant l'algorithme de limitation du nombre de sommets n'est pas connu. En effet, bien que cela soit tout à fait possible, mettre à jour toutes les informations (nombre de sommets, nombre de particules, poids sur les sommets, etc.) et les transférer aux ressources dont les sous-graphes sont voisins amènerait probablement un surcoût non négligeable. De plus, la restriction d'envois de particules énoncée dans la sous-section 5.4.4 permet de minimiser les déséquilibres engendrés par l'algorithme de limitation du nombre de sommets.

5.5.1 Introduction

L'équilibrage de la répartition des particules dans un contexte dynamique où seules les opérations locales sont autorisées consiste à échanger des particules entre ressources dont les sous-graphes sont voisins. Les ressources dont les sous-graphes sont voisins ont connaissance de leurs nombres de particules respectifs.

Dans le cas où une surcharge est constatée, nous devons rééquilibrer la charge. La première étape consiste alors à déterminer les écarts de charge entre la ressource i et toutes les autres. Il est nécessaire de comparer le nombre de particules de la ressource i avec le nombre de particules de chacune des ressources avec lesquelles elle possède une relation de voisinage. En effet, une ressource peut être en moyenne surchargée tout en étant sous-chargée par rapport à une autre ressource. Nous obtenons alors l'ensemble des écarts de charge, exprimés en nombre de particules que l'on doit envoyer pour rétablir l'équilibre, au moyen de l'équation 5.33.

Algorithme 11 : Limitation du nombre de sommets.

$w \leftarrow \underline{w}$
/* Tant que le nombre de sommets de la ressource notée i est supérieur au seuil */
tant que $\#S'_i > \#S'_{seuil} \wedge w \leq \overline{w}$ **faire**
  $n' \leftarrow \emptyset$
  /* Pour tous les sommets de poids w */
  pour chaque $s_i^x \in S'_i, w'(s_i^x) = w$ **faire**
    $max\_score \leftarrow 0$
    /* Sélection du voisin */
    pour chaque $n \in N_i$ **faire**
      $candidat\_score \leftarrow score(x, i, n)$
      /* Le sous-graphe de la ressource n est-il notre meilleur candidat ? */
      si $candidat\_score \geq max\_score \wedge \overline{P}_i^n + p_i^x \leq r_i^p P_i$ **alors**
        $n' \leftarrow n$
        $max\_score \leftarrow candidat\_score$
      fin
    fin
    /* Avons nous trouvé un sous-graphe destinataire ? */
    si $n' \neq \emptyset$ **alors**
      /* Transfert du sommet */
      $S'_{n'} \leftarrow S'_{n'} \cup \{s_i^x\}$
      $S'_i \leftarrow S'_i \setminus \{s_i^x\}$
      /* On met à jour le nombre de particules concernées par un transfert de i vers n' */
      $\overline{P}_i^{n'} \leftarrow \overline{P}_i^{n'} + p_i^x$
      /* Suffisamment de sommets envoyés ? */
      si $\#S'_i \leq \#S'_{seuil}$ **alors**
        $STOP$
      fin
    fin
  fin
  $w \leftarrow w + 1$
fin
/* Remise à zéro du nombre de particules envoyées */
pour chaque $n \in N_i$ **faire**
  $\overline{P}_i^n \leftarrow 0$
fin

$$\forall n \in N_i \ \underline{P_i^n} = \frac{P_i - P_n}{2} \ . \tag{5.33}$$

Si $\underline{P_i^n}$ est strictement supérieur à zéro, cela signifie que la ressource i doit donc envoyer $\underline{P_i^n}$ à la ressource n pour rétablir l'équilibre entre les deux ressources. Le rétablissement de l'équilibre en nombre de particules entre deux ressources doit être suffisamment restreint pour limiter à un ensemble pertinent de sommets les transferts entre deux ressources. Les règles que nous souhaitons mettre en place dans la régulation des particules ont une philosophie assez proche de celles que nous avions définies pour la limitation du nombre de sommets. En effet, l'idée est ici aussi d'éviter d'imposer un sommet inadapté à un sous-graphe. Concrètement, nous proposons de restreindre les déplacements en fonction des poids de sommets dans les sous-graphes source et destination.

Nous admettons que les sommets de poids élevé, en plus de permettre une régulation plus rapide, ont de fortes chances d'être conservés par le sous-graphe destinataire. Nous définissons w_{seuil}, comme le poids seuil à partir duquel nous admettons que le transfert d'un sommet n'a plus de nécessité d'être restreint. En revanche, si le poids du sommet est inférieur à ce seuil, le transfert doit être soumis à certaines conditions. Nous proposons de prendre en compte le poids de l'instance du sommet dans le sous-graphe destination, de telle sorte que, si ce poids est suffisamment élevé, le transfert sera possible. Nous proposons également de prendre en compte le poids du sommet dans le sous-graphe source, de telle sorte que, plus le poids est faible, moins le transfert sera possible. Nous proposons donc d'autoriser le transfert du sommet s_i^x de la ressource i vers la ressource n si la condition définie par l'équation 5.34 est vérifiée.

$$w'(s_n^x) \geq w_{seuil} - w'(s_i^x) \ . \tag{5.34}$$

Soit $\overline{P_i^n}$ le nombre de particules maximal pouvant effectivement être transféré de i vers n en respectant les restrictions définies par l'équation 5.34. La valeur de $\overline{P_i^n}$ est soumise à la condition de l'équation 5.35.

$$\forall n \in N_i \ \overline{P_i^n} \leq \underline{P_i^n} \ . \tag{5.35}$$

Nous avions défini un ordre prioritaire dans l'envoi des sommets pour l'algorithme de limitation du nombre de sommets (algorithme 11). Nous proposons d'en définir un autre pour l'algorithme d'équilibrage du nombre de particules. L'objectif de cet algorithme est de rééquilibrer en termes de particules des paires de ressources. Nous souhaitons atteindre cet objectif en transférant le moins de sommets possible. Pour cela, nous proposons de sélectionner prioritairement les sommets de plus hautes densités de particules. Rappelons que la densité d'un sommet est noté $w^1(s_i^x)$.

5.5.2 Algorithme principal d'équilibrage du nombre de particules

L'algorithme 12 décrit la méthode que nous proposons pour équilibrer les ressources entre elles. Cet algorithme a pour objectif de permettre à chaque ressource de se rééquilibrer avec les ressources avec lesquelles elle possède une relation de voisinage. Chaque ressource ayant un accès à la structure complète de son sous-graphe et de ses données de simulation, l'opération de régulation est effectuée par la ressource la plus chargée. Cet algorithme nécessite au préalable que la ressource i ait calculé l'ensemble des $\overline{P_i^n}, \forall n \in N_i$. Nous expliquerons ultérieurement la méthode proposée pour calculer ces valeurs.

L'algorithme consiste à ce que la ressource i tente de réaliser des opérations de régulation de charge tant qu'il existe une ressource susceptible de permettre cette régulation, c'est-à-dire une ressource disponible pour une opération de régulation ayant moins de particules que la ressource i

(ligne 2). Pour cela, nous devons élire la ressource avec laquelle nous estimons avoir plus de chance d'opérer une régulation efficace (lignes 5 à 8).

Permettre que deux ressources ou plus puissent utiliser un voisin commun pour faire de la régulation de charge impliquerait très probablement que cette ressource devienne alors malencontreusement surchargée. En effet, une opération de régulation entre une ressource i et une ressource n ne tient pas compte des éventuels transferts que pourrait réaliser une autre ressource. Pour résoudre ce problème, nous proposons un système de jeton (voir chapitre 2, figure 2.8). Chaque ressource i possède alors son jeton noté J_i. La ressource i tente donc de prendre le jeton de n' (lignes 12 et 13). Dans l'hypothèse où le jeton est pris, la ressource i peut alors essayer d'envoyer des sommets à n' dans l'optique d'équilibrer le nombre de particules des deux ressources (ligne 15). Pour réaliser une régulation de charge efficace, nous proposons d'envoyer en priorité les sommets les plus fortement pondérés selon le critère du nombre de particules. En effet, cette technique permet de minimiser le nombre de sommets à envoyer, minimisant les risques de surconsommation mémoire de la ressource destinataire. De plus, cette technique permet de transférer des sommets qui, probablement, seront conservés par le sous-graphe destinataire (le poids du sommet étant, à fortiori, élevé) rendant cet équilibrage pérenne dans le temps.

Le transfert d'un sommet est considéré valide selon plusieurs conditions. Premièrement, il faut que son poids, selon le critère de densité de particules, corresponde à la valeur de w^1, le poids de l'itération courante (ligne 17). Deuxièmement, il faut que la condition de l'équation 5.34 soit satisfaite et que le nombre de particules du sommet soit inférieur ou égal au nombre de particules restant à transférer (ligne 18). Si ces conditions sont satisfaites, le transfert peut avoir lieu (lignes 19 à 21). Ensuite, le nombre de particules de la ressource i ayant diminué, il faut mettre à jour les nombres de particules restants à transférer (lignes 22 à 23). Soit $\tau'_p \in [0, 1]$, proche de zéro, le ratio déterminant si un équilibrage est significatif ou non. Si un équilibrage n'est pas jugé significatif (ligne 29), nous relâchons le jeton pour permettre à une autre ressource d'utiliser n' pour se rééquilibrer. La tentative de régulation de charge de i avec n' achevée, les valeurs de $\underline{P}_i^{n'}$ et $\overline{P}_i^{n'}$ sont mises à zéro afin d'exclure n' des potentielles futures candidates (lignes 33 et 34).

L'algorithme se termine à partir du moment où la régulation des charges de calcul a été obtenue ou que toutes les tentatives pour y parvenir ont été effectuées.

5.5.3 Algorithme d'initialisation des potentiels de régulation

L'algorithme 13 présente la méthode de calcul des $\overline{\underline{P}}_i^n, \forall n \in N_i$. La proposition consiste à simuler l'algorithme de régulation proprement dit en prenant les sous-graphes voisins comme destinataire à tour de rôle. Pour cela, nous parcourons l'ensemble des sous-graphes voisins (ligne 1). Pour chaque voisin $n \in N_i$, nous calculons l'écart en nombre de particules comme défini en utilisant l'équation 5.33. Nous stockons cet écart dans la variable δ_{P_n}. Ensuite, le principe est le même que pour l'algorithme 12, à la différence qu'aucun sommet n'est transféré. En effet, lorsqu'un sommet s_i^x est valide, nous incrémentons $\overline{\underline{P}}_i^n$ de p_i^x (ligne 5). Notons qu'un sommet donné peut être valide pour plusieurs sous-graphes alors que l'algorithme 12 oblige qu'un sommet ne puisse être transféré que vers un seul sous-graphe. Ce n'est pas gênant, dans le mesure où la valeur de $\overline{\underline{P}}_i^n$ désigne un potentiel de régulation et, en aucun cas, une prédiction. Ainsi, nous obtenons à la fin les $\overline{\underline{P}}_i^n, \forall n \in N_i$ qui sont utilisés par l'algorithme 12.

Algorithme 12 : Équilibrage de la répartition des particules.

```
1  candidat_valide ← VRAI
2  tant que candidat_valide = VRAI faire
3      P_i^{n'} = 0
4      candidat_valide ← FAUX
       /* Sélection du meilleur candidat valide                          */
5      pour chaque n ∈ N_i faire
6          si P_i^n > P_i^{n'} alors
7              n' ← n
8              candidat_valide ← VRAI
9          fin
10     fin
       /* S'il existe un candidat valide                                 */
11     si candidat_valide = VRAI alors
           /* Le jeton n'a-t-il pas déjà été pris ? Si non, prise du jeton */
12         si J_{n'} = LIBRE alors
13             J_{n'} ← PRIS
14             w^1 ← w̄
               /* Tant qu'il y a du déséquilibre                         */
15             tant que (P_i^{n'} > 0) ∧ (w^1 ≥ w) faire
                   /* Sélection des sommets en commençant par les plus fortement
                      pondérés selon le premier critère                  */
16                 pour chaque s_i^x ∈ S'_i, w^1(s_i^x) = w^1 faire
17                     si w^1(s_i^x) = w^1 alors
                           /* Le candidat peut-il recevoir ce sommet ?   */
18                         si (w'(s_n^x) ≥ w_seuil − w'(s_i^x)) ∧ (p_i^x ≤ P_i^{n'}) alors
                               /* Transfert du sommet                    */
19                             S'_{n'} ← S'_{n'} ∪ {s_i^x}
20                             S'_i ← S'_i \ {s_i^x}
21                             P̄_i^{n'} ← P̄_i^{n'} + p_i^x
                               /* Moins d'équilibrages à faire           */
22                             P_i^n ← P_i^n − p_i^x, ∀n ∈ N_i
23                             P̄_i^n ← P̄_i^n − p_i^x, ∀n ∈ N_i
24                         fin
25                     fin
26                 fin
27                 w^1 ← w^1 − 1
28             fin
               /* Si une quantité trop peu significative de particules a été
                  transférée, on libère le jeton de n'                   */
29             si P̄_i^{n'} < P_n ∗ τ'_p alors
30                 J_{n'} ← LIBRE
31             fin
32         fin
           /* n' ne peut plus être utilisé                               */
33         P_i^{n'} ← 0
34         P̄_i^{n'} ← 0
35     fin
36 fin
```

Algorithme 13 : Initialisation des potentiels de régulation de charge.

```
   /* Pour tous les sous-graphes voisins de i                          */
1  pour chaque n ∈ N_i faire
2      δ_{P_n} ← P_i^n
3      P̄_i^n ← 0
4      w^1 ← w̄
       /* Tant qu'il y a du déséquilibre                               */
5      tant que (δ_{P_n} > 0) ∧ (w^1 ≥ w) faire
6          pour chaque s_i^x ∈ S'_i faire
               /* Sélections des sommets en commençant par les plus fortement
                  pondérés                                             */
7              si w^1(s_i^x) = w^1 alors
                   /* Le sous-graphe n peut-il recevoir ce sommet ?   */
8                  si (w'(s_n^x) ≥ w_seuil − w'(s_i^x)) ∧ (p_i^x ≤ δ_{P_n})) alors
                       /* Mise à jour du déséquilibre                 */
9                      δ_{P_n} ← δ_{P_n} − p_i^x
                       /* Mise à jour de la valeur du potentiel d'équilibrage */
10                     P̄_i^n ← P̄_i^n + p_i^x
11                 fin
12             fin
13         fin
14         w^1 ← w^1 − 1
15     fin
16 fin
```

5.5.4 Conclusion

Nous avons développé des méthodes de régulation de charge et de donnée. Ces méthodes sont complémentaires entre elles. Ces méthodes sont des heuristiques dans le sens où les solutions proposées ne sont pas nécessairement optimales. Elles sont aussi moins coûteuses que des solutions exactes dont les complexités sont exponentielles. En effet, nos méthodes ont une complexité en $\mathcal{O}(n)$ où n est le nombre de sommets du sous-graphe. Néanmoins, nous souhaitons garantir que les objectifs ciblés soient atteints avec une tolérance raisonnable. Nous proposons d'optimiser les solutions obtenues par l'approche dynamique en instaurant des mécanismes de contrôle.

5.6 Optimisations

L'approche que nous avons présentée ne peut pas garantir que les objectifs soient atteints. Nous parlons dans ce cas de solution invalide. L'utilisation de mécanismes de contrôles doit permettre de détecter lorsqu'une solution obtenue n'est pas pleinement satisfaisante et d'apporter une correction. Nous commencerons par présenter une méthode de détection d'invalidité. Nous présenterons ensuite des techniques permettant de corriger ces invalidités.

5.6.1 Détections d'invalidités par anticipation

Les solutions obtenues par l'approche dynamique peuvent être de plus ou moins bonne qualité. Nous admettons que nous pouvons définir une qualité en dessous de laquelle nous ne pouvons pas être satisfait. Cette qualité minimale correspond à une solution valide mais se rapprochant suffisamment d'une solution invalide pour que des mécanismes supplémentaires doivent être mis en œuvre. Nous pouvons ainsi détecter des solutions invalides avant qu'elle ne se produisent réellement.

Nous avons défini $\#S'_{max}$ le nombre de sommets maximal qu'un sous-graphe peut posséder. Nous avons défini $\#S'_{seuil}$ le nombre de sommets à partir duquel nous jugeons pertinent d'opérer des transferts de sommets. Il est possible qu'une ressource ne puisse pas envoyer suffisamment de sommets pour diverses raisons. Nous proposons de détecter ce dysfonctionnement avant qu'il n'atteigne un état critique, correspondant à $\exists i, \#S'_i > \#S'_{max}$. Nous proposons de placer notre seuil de tolérance à mi-chemin entre l'état critique et l'optimal. Nous définissons alors $\#S'_{ref}$, le seuil de détection de dysfonctionnement, au moyen de l'équation 5.36.

$$\#S'_{ref} = \frac{\#S'_{max} + \#S'_{seuil}}{2} \ . \tag{5.36}$$

Nous avons défini P_{max}, dépendant de τ_p une tolérance de déséquilibre, comme étant le seuil de partitionnement de la méthode par partitionnement (chapitre 4, sous-section 4.2.2, équation 4.5). Le déséquilibre en nombre de particules de la méthode par partitionnement a tendance à être croissant au fur et à mesure des itérations. Il est donc pertinent de définir ce seuil assez bas pour éviter d'être beaucoup trop déséquilibré les $X-1$ itérations suivantes. L'approche dynamique permet de maîtriser le déséquilibre à chaque itération. Même si le déséquilibre peut être croissant, sa croissance sera freinée. Nous proposons donc de définir une nouvelle tolérance de déséquilibre τ''_p pour la méthode dynamique. Nous obtenons alors P_{ref} le nombre de particules seuil à partir duquel une ressource est jugée trop surchargée.

$$P_{ref} = \frac{P}{N} \times (1 + \tau''_p) \ . \tag{5.37}$$

Soit τ_m une tolérance de surconsommation mémoire. Nous définissons par $M_{max} = (1 + \tau_m)(R\frac{M}{N})$ le nombre de mailles maximal qu'une ressource peut posséder en utilisant la méthode par partitionnement. Soit τ''_m, une nouvelle tolérance de déséquilibre relative à la limitation du nombre de mailles. Étant donné τ''_m, nous définissons M_{ref}, le nombre de mailles maximal à partir duquel la solution n'est plus satisfaisante au moyen de l'équation 5.38.

$$M_{ref} = R\frac{M}{N} \times (1 + \tau''_m) \ . \tag{5.38}$$

Nous avons défini plusieurs constantes permettant de déterminer si une solution est acceptable ou non. Nous proposons maintenant des méthodes permettant de corriger ces solutions quand elles ne sont pas satisfaisantes. Un voisinage inadapté peut être une cause de dysfonctionnements, en particulier en ce qui concerne la limitation du nombre de mailles. Pour éviter ces dysfonctionnements, nous proposons une méthode de mise à jour des voisinages des ressources.

5.6.2 Invocation de mise à jour de voisinages

Les erreurs potentielles de l'approche dynamique peuvent provenir de la qualité de ses voisins. En effet, un sous-graphe ayant exploité le maximum des affinités avec ses sous-graphes aura de grandes difficultés à envoyer suffisamment de sommets. L'équilibrage en nombre de particules pourrait aussi être impacté par un voisinage obsolète. En effet, une ressource peut être équilibrée avec les ressources, avec lesquelles elle possède une relation de voisinage, sans pour autant être équilibrée avec toutes les autres. Nous proposons donc de mettre à jour les voisinages lorsque la qualité du voisinage d'un sous-graphe n'est plus suffisante.

Nous proposons de recalculer les voisins toutes les X itérations si nécessaire. Nous choisissons le même X que pour la méthode par partitionnement présentée au chapitre 4. En effet, le choix

de X est soumis à la même contrainte, à savoir réduire le coût tout en garantissant une solution acceptable.

La détection de la baisse de la qualité du voisinage peut paraître assez complexe. Dans les faits, il suffit de vérifier si le nombre de sommets demeure limité. Ainsi, si la condition de l'équation 5.39 est vérifiée, l'invocation de la mise à jour des voisinages est opérée.

$$\exists i, \#S'_i > \#S'_{ref} \, . \tag{5.39}$$

L'algorithme 14 présente la méthode d'invocation utilisée. À chaque itération, chaque ressource teste localement si le nombre de sommets qu'elle possède est conforme avec les objectifs fixés (ligne 1). Soit $\tilde{N}_i$ un booléen déterminant si la ressource i souhaite une mise à jour des voisins. Dans le cas où une anomalie est détectée par la ressource i, la ressource met à jour le booléen $\tilde{N}_i$ à Vrai (ligne 2). Nous rappelons que la mise à jour des voisins ne peut être opérée que toutes les X itérations. C'est pourquoi nous testons la condition d'invocation de la mise à jour des voisins que tous les X itérations (ligne 4). Pour tester la condition d'invocation, nous devons en premier lieu échanger les $\tilde{N}_i$ de chaque ressource (ligne 5). Ensuite, la condition d'invocation est testée (ligne 6). Dans le cas où la condition est vérifiée, la mise à jour des voisins est alors réalisée en utilisant la méthode de sélections itératives (ligne 7).

Algorithme 14 : Invocation de la mise à jour des voisins.

```
   /* Le nombre de sommets est-il dans le domaine de validité ?          */
1  si #S'_i > #S'_ref alors
2  |  Ñ_i ← Vrai
3  fin
   /* Point de synchronisation atteint                                   */
4  si iter_courante ≡ 0 (mod X) alors
5  |  Echanger_demande_maj(Ñ_i, N)
   |  /* Y a-t-il au moins une ressource qui exige la mise à jour des
   |   voisins ?                                                        */
6  |  si ∃k, Ñ_k = Vrai alors
7  |  |  Mise_a_jour_iterative_voisins(N)
8  |  |  Ñ_i ← Faux
9  |  fin
10 fin
```

La mise à jour des voisins permet d'améliorer la répartition de la charge et des données en améliorant la qualité des voisinages. Cependant, Il est possible que, dans certains cas, cela ne suffise pas. Typiquement, si l'algorithme de limitation du nombre de sommets a divergé de telle sorte qu'une ressource est incapable d'envoyer plus de sommets qu'elle n'en crée en une itération, alors un mécanisme plus robuste devient nécessaire. Nous proposons alors d'utiliser une technique de partitionnement.

5.6.3 Invocation de partitionnement

Nous avons vu au chapitre 4 que le partitionnement était coûteux. Cependant nous avons vu qu'il permettait d'obtenir une répartition valide dans la plupart des cas. La condition d'invocation du partitionnement doit être clairement moins stricte que celle de la mise à jour des voisins. Toutefois, elle doit être suffisamment contraignante pour qu'au moment de partitionner la répartition ne soit pas mauvaise, en particulier sur les données. En effet, une surconsommation mémoire trop importante pourrait dépasser la capacité mémoire des ressources.

Nous avons défini P_{ref} comme étant le nombre de particules à partir duquel nous considérons que la répartition des particules n'est pas satisfaisante. Lorsque l'équation 5.40 est vérifiée, nous proposons d'invoquer le partitionnement du graphe global afin de rétablir l'équilibre.

$$\exists i, P_i > P_{ref} \ . \tag{5.40}$$

Nous avons défini M_{ref} comme étant le nombre de mailles à partir duquel nous considérons que la répartition des mailles n'est pas satisfaisante. Lorsque l'équation 5.41 est vérifiée, nous proposons ici aussi d'invoquer le partitionnement du graphe global afin de rétablir l'équilibre.

$$\exists i, M_i > M_{ref} \ . \tag{5.41}$$

L'algorithme 15 présente la méthode d'invocation du partitionnement que nous proposons. Cette méthode ressemble beaucoup à celle de l'invocation de la mise à jour des voisins. Le critère d'invocation dépend non plus d'un seul paramètre mais de deux (M_i et P_i, ligne 1). Nous réalisons le partitionnement, si au moins une ressource l'a exigé, puis nous mettons à jour les voisinages (ligne 8). En effet, le partitionnement va rendre obsolète les voisinages.

Algorithme 15 : Invocation du partitionnement.

```
   /* Le nombre de particules et le nombre de mailles sont-ils dans le
    domaine de validité ?                                              */
1  si P_i > P_ref ∨ M_i > M_ref alors
2  |  P̃M_i ← Vrai
3  fin
   /* Point de synchronisation atteint                                  */
4  si iter_courante ≡ 0 (mod X) alors
5  |  Echanger_demande_partitionnement(P̃M_i, N)
   |  /* Y a-t-il au moins une ressource qui exige le partitionnement ? */
6  |  si ∃k, P̃M_k = Vrai alors
7  |  |  Partitionnement(N)
8  |  |  Mise_a_jour_iterative_voisins(N)
9  |  |  P̃M_i ← Faux
10 |  fin
11 fin
```

Nous avons défini une technique de partitionnement multi-critères dans le chapitre 4. Cette technique permet de distribuer les mailles et les particules de manière équitable, le tout en minimisant les interfaces entre les sous-domaines. Dans un contexte dynamique, les contraintes sont quelque peu différentes. En effet, la considération des interfaces telle que définie dans le chapitre 4 est assez peu pertinente pour l'approche dynamique. Nous allons donc voir comment adapter le partitionnement pour que celui-ci corresponde mieux aux propriétés de l'approche dynamique.

5.6.4 Technique de partitionnement adaptée à l'approche dynamique

Le partitionnement consiste à décomposer un graphe en sous-graphes. Dans un contexte polygraphes, pour pouvoir réaliser un partitionnement, nous devons, en premier lieu, constituer un graphe unique. Nous devons alors construire les sommets et arêtes à partitionner.

Ensemble de sommets et d'arêtes à partitionner

Le partitionnement que nous proposons d'opérer consiste à répartir les sommets de tous les graphes après avoir fusionné les instances entre elles. Cela revient donc en principe à mettre à jour le graphe $G = (S, A)$ que nous avions défini au chapitre 4. S désigne l'ensemble des sommets de G, y compris ceux n'ayant pas de particules. Nous étions obligé de prendre en compte l'intégralité des sommets avec les méthodes par partitionnement. Dans un contexte dynamique, les sommets n'ayant pas de particules n'ont plus lieu d'être. Soit $\hat{S}$ tel que $\exists i, s_i^x \in S'_i \implies s^x \in \hat{S}$. Nous proposons alors de ne partitionner que les sommets appartenant à $\hat{S}$. Les sommets sur lesquels le partitionnement doit être appliqué possèdent des arêtes. Le partitionnement coupe des arêtes donnant lieu à des interfaces entre les sous-domaines. Ces interfaces impliqueront de potentielles créations de sommets. Nous proposons donc de prendre en considération toutes les arêtes dont au moins un des sommets qui la compose appartient à $\hat{S}$. Nous définissons $\hat{A}$ tel que $a = (s^x, s^y), s^x \in \hat{S} \implies a \in \hat{A}$.

Fonctions de pondération

La fonction de pondération des sommets appartenant à S est autant adaptée à l'approche statique qu'à l'approche dynamique, dans la mesure où ils représentent les mailles et les nombres de particules sur chacune d'elles. Nous proposons donc d'utiliser cette fonction de pondération pour les sommets appartenant à $\hat{S}$. En revanche, la fonction de pondération des arêtes A indique à l'outil de partitionnement les coûts de communication potentiels. Ces coûts s'appliquent sur les transferts de sommets avec l'approche dynamique. Nous proposons donc une autre fonction de pondération des arêtes notée $\hat{w}()$ définie par l'équation 5.42.

$$\hat{w}(a) = 1, \forall a \in \hat{A} \, . \tag{5.42}$$

Partitionnement et attributions de sous-graphes

Nous souhaitons réaliser un partitionnement multi-critères du graphe global. Le premier critère porte sur le nombre de sommets par sous-graphe et le second concerne le nombre de particules par sous-graphe. Nous proposons d'utiliser la même éthologie que celle du partitionnement du graphe global détaillée au chapitre 4, sous-section 4.2.3. En revanche, nous utiliserons le même principe pour la pondération des arêtes que celui des sous-graphes dynamiques (sous-section 5.2.2). En effet, les arêtes seront pondérés binairement. Nous pondérons à 1 les arêtes reliant des sommets possédant des particules, à 0 sinon. Le graphe global dont la pondération des arêtes a été adaptée est noté $\hat{G}$. Le graphe global étant déjà partitionné en sous-graphes, nous avons défini $\overset{\circ}{S}_i, \forall i$. L'idée est donc de ne partitionner que les sommets possédant des particules. Nous devons déterminer le nombre de sous-graphes. Nous proposons de déterminer le nombre de sous-graphes en fonction de $\#S'_{seuil}$. L'équation 5.43 donne le calcul du nombre de sous-graphes n. Nous pouvons alors partitionner le graphe $\hat{G} = (\hat{S}, \hat{A})$ en n sous-graphes équilibrés.

$$n = max(\{n_p = 2^p, \frac{\#\hat{S}}{n_p} \leq \#S'_{seuil}\}) \, . \tag{5.43}$$

Le partitionnement donne l'ensemble des sommets pour les n sous-graphes. Nous supposons que ces sous-graphes sont équilibrés en nombre de sommets ainsi qu'en nombre de particules représentées. Nous notons $\hat{S}_{k,n}$ l'ensemble des sommets du kème sous-graphe. Nous devons maintenant attribuer l'ensemble des $\hat{S}_{k,n}$ aux ressources. Soit une ressource notée i, nous devons trouver

k tel que $S'_i \leftarrow \hat{S}_{k,n}$. Nous avons $n \leq N$, nous devons donc attribuer potentiellement plusieurs ressources à une même sous-graphe. Nous proposons de répartir de manière équitable les ressources sur les sous-graphes. Le nombre de ressources maximal à attribuer à chaque sous-graphe est donné par l'équation 5.44. Nous avions défini $\overset{\circ}{\underline{x}}_i$ le barycentre du sous-graphe noté i issu du partitionnement en N sous-graphes équilibrés en nombre de sommets. Nous avons défini au chapitre 4, sous-section 4.2.3, $\overline{\underline{x}}_{k,n}$ le barycentre du sous-graphe noté k issu du partitionnement en n sous-graphes équilibrés en nombre de sommets et en particules. Nous proposons de choisir pour chaque ressource i le sous-graphe k telle que la distance entre $\overset{\circ}{\underline{x}}_i$ et $\overline{\underline{x}}_{k,n}$ soit la plus faible possible.

$$n' = \lceil \frac{N}{n} \rceil \,. \tag{5.44}$$

L'algorithme 16 donne la méthode utilisée pour attribuer les ressources aux sous-graphes. Nous recherchons le sous-graphe à assigné à chaque ressource. Pour cela, nous parcourons l'ensemble des ressources (ligne 2). Pour chaque ressource, nous choisissons le sous-graphe k, parmi ceux à qui moins de n' ressources ont été attribuées (ligne 5), dont la distance entre son barycentre $\overline{\underline{x}}_{k,n}$ et le barycentre $\overset{\circ}{\underline{x}}_i$ soit la plus faible possible. Nous affectons alors l'ensemble des sommets du graphe de i aux sommets du kème sous-graphe (ligne 12). Les arêtes du graphe de i seront obtenues par construction.

Algorithme 16 : Attribution des ressources aux sous-graphes.

```
1  n'_j ← n'
   /* Pour chaque ressource                                          */
2  pour chaque i ∈ [0, N − 1] faire
3      D_{i,min} ← D_max /* Recherche du sous-graphe le plus proche  */
4      pour chaque j ∈ [0, n − 1] faire
5          si n'_j > 0 alors
6              si D(x°_i, x̄_{j,n}) ≤ D_{i,min} alors
7                  k ← j
8                  D_{i,min} ← D(x°_i, x̄_{j,n})
9              fin
10         fin
11     fin
12     S'_i ← Ŝ_{k,n} n'_k ← n'_k − 1
13 fin
```

Nous avons proposé un ensemble de méthodes et d'algorithmes permettant une régulation dynamique des charges de calcul et des données. Les mécanismes de régulation proposés permettent de maintenir une distribution des charges de calcul et des données a un niveau d'équilibre tout en nécessitant un coût modéré. Nous proposons de récapituler l'ensemble des mécanismes mis en œuvre puis de les illustrer par un exemple et enfin de rendre une analyse critique de notre approche.

5.7 Conclusion

Nous avons présenté des algorithmes avec chacun des objectifs différents. Les algorithmes sont liés les uns aux autres, dans la mesure où aucun algorithme n'a vraiment de sens indépendamment des autres. Nous proposons donc de détailler l'algorithme général regroupant l'ensemble des mécanismes que chaque ressource exécute.

5.7.1 Algorithme général détaillé

Notre approche dynamique nécessite la mise en place de plusieurs algorithmes. Ces algorithmes sont invoqués successivement à chaque itération par chaque ressource. L'algorithme 17 présente le déroulement de la simulation pour la ressource i en utilisant l'approche dynamique. En début de simulation, après le calcul des $\overset{\circ}{S}_i$ (ligne 3), nous initialisons les voisins (ligne 4). À chaque itération de la boucle principale, nous suivons le même procédé. Nous commençons par appeler l'algorithme d'invocation de l'outil de partitionnement (ligne 7). Si celui-ci décide d'opérer un partitionnement, nous devons appeler l'algorithme d'attribution des ressources aux sous-graphes ainsi créés (ligne 9). Dans le cas contraire, nous mettons à jour le sous-graphe G'_i (ligne 12). Ensuite, nous testons si une mise à jour des voisins est nécessaire. Pour cela, nous appelons l'algorithme d'invocation de mise à jour des voisins (ligne 13). Les échanges de sous-graphes sont alors réalisés (ligne 14). L'algorithme de limitation du nombre de sommets est ensuite invoqué (ligne 15). Puis, l'algorithme de régulation du nombre de particules est appelé à son tour (lignes 16 et 17). Afin d'éviter l'envoi de nombreux messages sur le réseau, les transferts de sommets d'une ressource à une autre sont rassemblés dans un seul et unique message. Enfin, les transferts de sommets sont effectués en fin d'itération (ligne 19).

L'algorithme que nous venons de détaillé implique des modifications des sous-graphes à chaque itération. Nous proposons d'illustrer ces changements par un exemple sur les opérations de transfert de sommets entre sous-graphes.

5.7.2 Exemple

Soit $N = 8$, $\underline{w} = 1$, $\overline{w} = 7$, $c^1 = 60$, $c^2 = 5$, $c^3 = 35$ et $R = 2$. Nous proposons de suivre l'évolution des graphes des 8 ressources pendant la durée d'une simulation. Le cas test choisi est le cas test hétérogène avec insertion centralisée permettant de faire le lien avec le chapitre précédent et l'exemple présenté en sous-section 4.2.5. La légende des couleurs associées aux poids des sommets des graphes est donnée en figure 5.6.

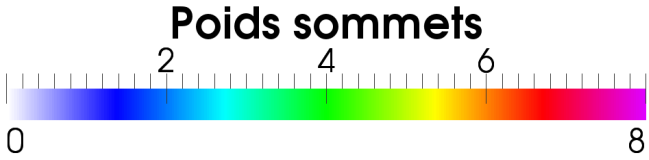


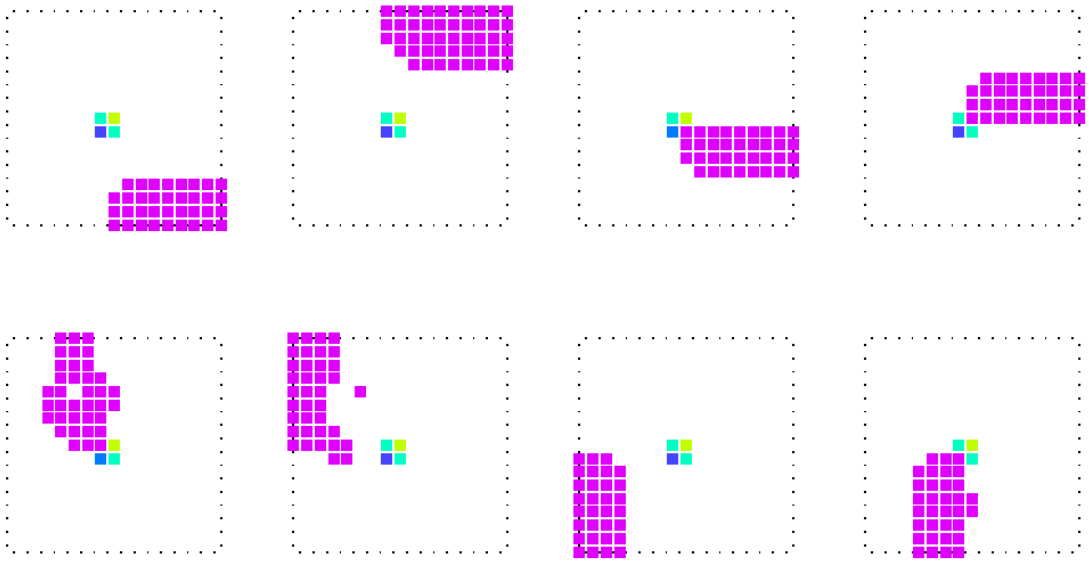
FIGURE 5.6 – Légende des couleurs utilisées dans les figures 5.7 et 5.8.

La figure 5.7 illustre l'évolution des graphes avant que les opérations de transfert de sommets n'aient lieu. À la fin de la première itération (sous-figure 5.7a), chaque ressource possède un graphe de 4 sommets situés au centre auquel s'ajoute les sommets $\overset{\circ}{S}_i$. Nous constatons d'ailleurs que les poids des sommets sont les mêmes pour toutes les ressources à l'exception des ressources dont les sommets centraux ont dans leur voisinage les sommets $\overset{\circ}{S}_i$. Cela est dû au critère de pondération du

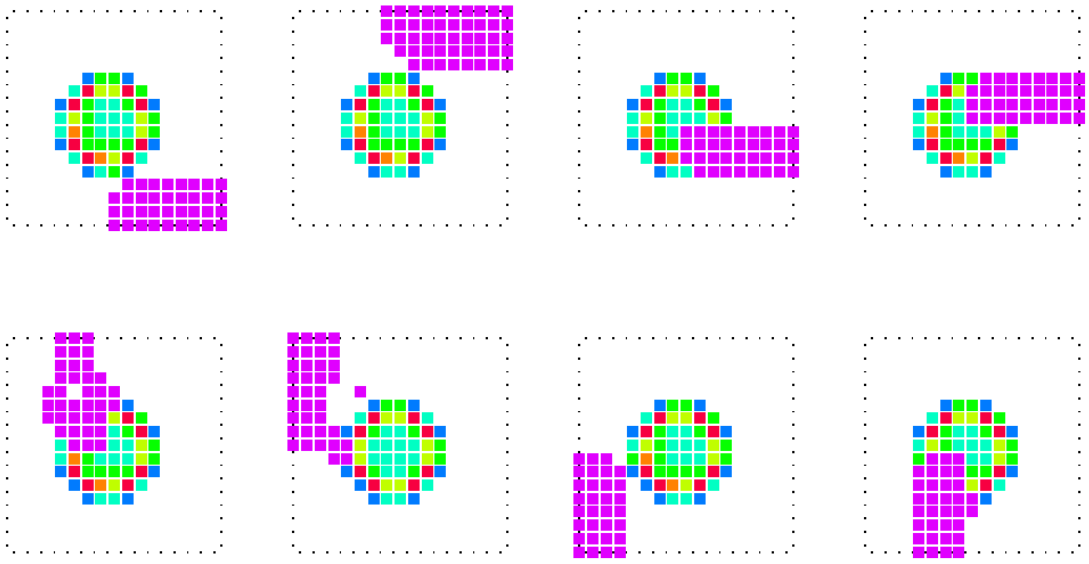
Algorithme 17 : Algorithme général détaillé.

```
   /* Initialisation                                                    */
1  Initialisation( Maillage, Particules )
   /* Création d'une version non pondérée du graphe G = (S, A)          */
2  Creation_graphe_global( Graphe, Maillage )
   /* Obtention des °S_i                                                */
3  Partitionnement( S, N, 0.05 )
   /* Initialisations des voisins 10                                    */
4  Mise_a_jour_iterative_voisins( N )
   /* Tant que la condition de fin de simulation n'est pas atteinte     */
5  tant que !fin_application faire
       /* Phase de Transport                                           */
6      Monte_carlo_transport( Maillage, Particules )
       /* Partitionnement en cas de nécessité (algorithme 15)          */
7      Test_partitionnement()
8      si Partitionnement fait alors
           /* Partitionnement réalisé, nous assignons les ressources aux
              sous-graphes (algorithme 16)                              */
9          Assignation_sous_graphes()
10     fin
11     sinon
           /* Création du sous-graphe avec les fonctions de pondérations citées
              en section 5.2                                            */
12         Mise_a_jour_sous_graphe( G'_i, Maillage_i, Particules_i )
           /* Mise à jour des voisins en cas de nécessité (algorithmes 10 et
              14)                                                       */
13         Test_maj_voisins()
           /* Communications de graphes entre voisins                  */
14         Echanges_sous_graphes( i , N_i)
           /* Limitation du nombre de sommets (algorithme 11)          */
15         Limitation_sommets( i , N_i)
           /* Initialisation des potentiels de régulations (algorithme 13) */
16         Init_regulation_particules( i , N_i)
           /* Régulation de charge (algorithme 12)                     */
17         Regulation_particules( i , N_i)
18     fin
       /* Envoi et réception des sommets entre les sous-graphes voisins */
19     Echanges_sommets( i , N_i)
20 fin
```

voisinage. Quelques itérations plus tard, les graphes se sont étendus (sous-figure 5.7b).



(a) Représentation des sommets des graphes au début de la simulation.



(b) Représentation des sommets des graphes avant les premières opérations de transfert.

FIGURE 5.7 – Évolution des graphes de 8 ressources de calcul avant les opérations de transfert pour le cas hétérogène par insertion centralisée.

La figure 5.8 illustre l'évolution des graphes à partir du moment où les opérations de transfert de sommets ont commencé. Après les premières opérations de transfert de sommets (sous-figure 5.8a), nous constatons que très clairement chaque ressource a appliqué une politique de conservation des

sommets différente. Cela semble largement déterminé par la localisation des sommets par rapport aux $\overset{\circ}{S}_i$. Cette tendance est confirmée plusieurs itérations plus tard (sous-figure 5.8b). Nous observons, en effet, que les ressources choisissent, en priorité, les sommets proches de leur $\overset{\circ}{S}_i$. C'est exactement le comportement voulu. Nous pouvons constater que certains graphes possèdent des sommets isolés des autres. Cela s'avère être totalement normal dans la mesure où la régulation du nombre de particules nécessite de transférer des sommets que le graphe destinataire ne possède pas forcément.

Cet exemple nous permet d'observer le comportement de notre approche. La question qui se pose maintenant est la suivante : notre méthode est-elle robuste et efficace ? En effet, la question de robustesse se pose dans la mesure où notre approche repose sur des heuristiques et plus généralement des méthodes calculant une solution approchée à un problème complexe. Nous proposons donc de mener une analyse critique de notre approche.

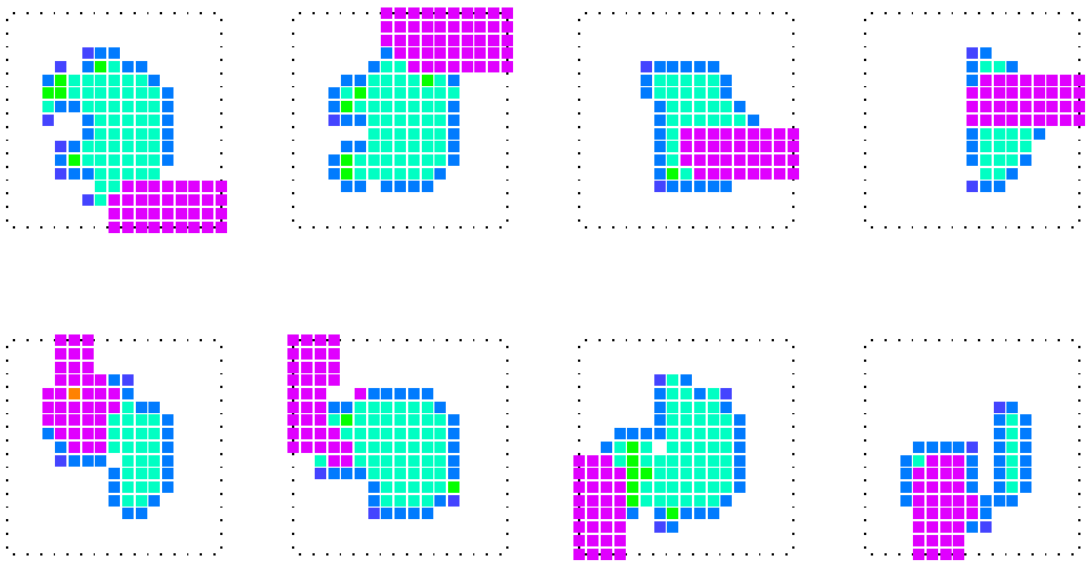
5.7.3 Analyse critique

L'approche dynamique que nous avons mis en place a pour objectif de garantir une bonne répartition de la charge de calcul et des données à un coût raisonnable. Pour cela, nous avons mis en place un ensemble de méthodes.

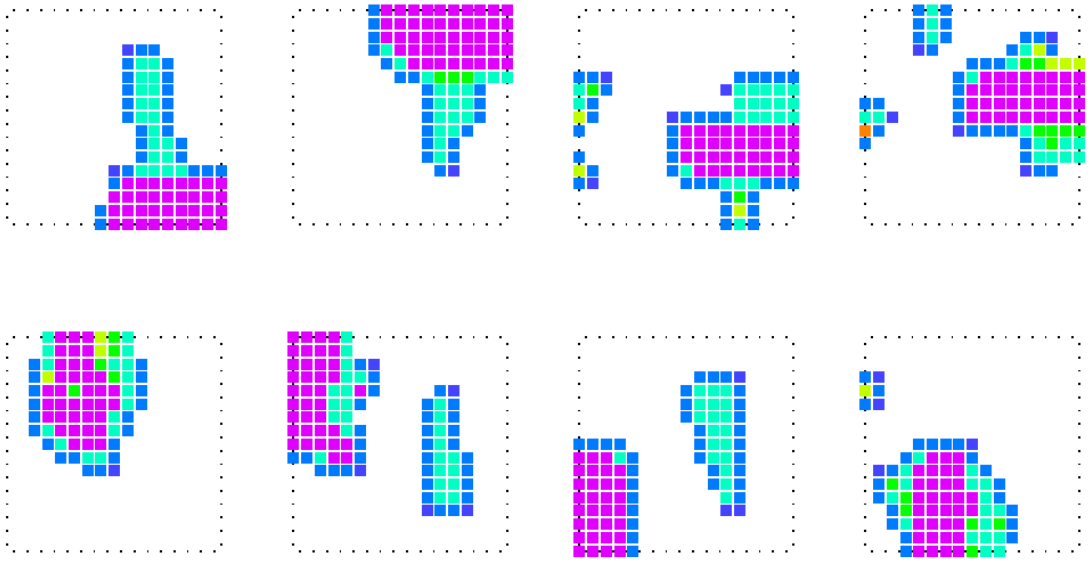
Le coût en communication de notre approche n'est pas nul. Les communications engendrées sont toutefois assez localisées, dans le sens où chaque ressource communique avec un ensemble de ressources restreint en taille. Le coût des communications point à point n'augmente donc pas, en théorie, avec le nombre de ressources. Certains mécanismes de notre approche sont collectifs (partitionnement, mise à jour des voisins), et leur coût constitue potentiellement une limitation. Il faut toutefois rappeler que ces mécanismes ne sont invoqués qu'en cas d'absolue nécessité. Dans le cas idéal, où nos méthodes fourniraient des résultats optimaux, ces mécanismes ne seraient jamais invoqués. Il est, par ailleurs, assez difficile de déterminer à l'avance quel sera le comportement de chacune des méthodes proposées et donc de savoir si elles fourniraient des résultats optimaux ou non. Il est également assez difficile de déterminer la pire répartition de la charge de calcul et des données pouvant être obtenue par notre approche, ainsi que le coût de régulation de charge et de donnée nécessaire à son fonctionnement.

Notre approche repose sur le fait que chaque ressource puisse créer à la volée des sommets et donc des mailles. Dans notre application, la physique simulée est simple. Les données géométriques peuvent être calculées simplement à partir des coordonnées. Les interactivités de chaque maille peuvent être obtenues dynamiquement à partir du calcul des données géométriques et les sections efficaces sont calculées à partir des interactivités. Dans une simulation plus complexe, ces données peuvent ne pas être calculées à la volée. Cette problématique pourrait être gérée en dupliquant les données géométriques et les sections efficaces de chaque maille. Cette duplication impliquerait néanmoins une surconsommation mémoire. À titre d'exemple, les sections efficaces peuvent nécessiter jusqu'à 100 GB [38]. Notre approche permet de limiter le nombre de mailles à traiter par ressource. Elle ne nécessite donc pas de dupliquer les données géométriques et les sections efficaces de toutes les mailles.

Notre approche est très paramétrable. Cela a des avantages comme des inconvénients. Le fait de pouvoir paramétrer nous permet de nous adapter aux besoins facilement. En revanche, cela soulève un problème de fond. En effet, comment savoir si la configuration des paramètres choisis est la meilleure possible ? Pour apporter des premiers éléments de réponses à toutes ces questions, nous allons mener, dans le chapitre 6, une étude expérimentale.



(a) Représentation des sommets des graphes après quelques opérations de transfert.



(b) Représentation des sommets des graphes lorsque les particules se sont dispersées dans presque tout le maillage.

FIGURE 5.8 – Évolution des graphes de 8 ressources de calcul dès lors que les opérations de transfert ont débuté pour le cas hétérogène par insertion centralisée.

Chapitre 6

Étude expérimentale de la régulation dynamique

Nous avons présenté une approche dynamique au chapitre 5. Cette approche consiste à invoquer des opérations de régulation de charge et de donnée, dynamiquement et localement. Les algorithmes que nous avons détaillés permettent de limiter le nombre de mailles stockées en mémoire et de répartir les particules entre les ressources. Cette approche a pour objectif de répondre aux limitations de l'approche par partitionnement.

Nous évaluerons dans ce chapitre l'approche dynamique en comparant les résultats obtenus par celle-ci avec les résultats de l'approche hybride, avec degré de réplication constant, ceux de l'approche réplication de domaine et ceux de l'approche par partitionnement. Nous commencerons par une étude en équilibrage de charge et de donnée, puis nous analyserons les résultats en extensibilité forte et enfin nous terminerons par une étude en extensibilité faible. Pour l'ensemble des résultats présentés, l'approche dynamique sera notée $RCDD$.

Les algorithmes de l'approche dynamique sont paramétrables. La table 6.1 liste les différentes valeurs, données aux paramètres, qui ont été utilisées pour réaliser les expériences de ce chapitre. Les descriptions des paramètres sont également rappelées. Nous notons que c^1 est majoritaire (60%). Nous avons choisi $w_{seuil} = 4$. Nous avons choisi une valeur intermédiaire exactement à mi-chemin entre $\underline{w}$ et $\overline{w}$ car cette valeur n'est pas trop permissive. Les transferts des sommets de faibles poids ne sont alors pas facilement admis. Cette valeur n'est pas non plus excessivement contraignante. Nous avons défini $\tau_p^{''} = 0,3$, la ressource la plus chargée peut donc avoir jusqu'à 1,3 fois plus de particules que la moyenne au delà duquel un partitionnement sera invoqué. Nous avons fixé $N' = 16$. Cette valeur est suffisamment élevée pour garantir des voisinages de bonne qualité. Nous avons choisi de minimiser la taille du graphe dans le but de minimiser le volume de communication correspondant à l'échange des graphes. Nous avons donc opté pour $r = 100$. Pour les mêmes raisons, nous avons choisi $r' = 16$.

6.1 Étude de régulation de charge et de donnée

L'approche dynamique a pour objectif de garantir une répartition relativement équitable des charges de calcul et des données à chaque itération. Nous avons donc réalisé une étude expérimentale comparative de répartition des charges de calcul et des données sur les cas test présentés au chapitre 2, sous-section 2.1.6. Nous allons présenter les résultats en déséquilibre de charge et en surconsommation mémoire. Nous rappelons que le déséquilibre de charge est donné

Nom	**Description**	**Valeur**
c^1	Coefficient de pondération du critère de densité de particules	60
c^2	Coefficient de pondération du critère de densité de mailles	5
c^3	Coefficient de pondération du critère de voisinage	35
$\underline{w}$	Poids de sommets minimal	1
$\overline{w}$	Poids de sommets maximal	7
w_{seuil}	Poids de sommets seuil pour l'équilibrage en nombre de particules	4
$\tau_p^{''}$	Tolérance de déséquilibre en particules	0,3
$\tau_m^{''}$	Tolérance de déséquilibre en mailles	0,2
τ_s	Marge par rapport à la limite en nombre de sommets	0,3
τ^*	Coefficient de mise à jour du score d'un graphe quotient	0,025
c^S	Coefficient du critère de nombre de sommets dans le calcul du score	0,275
c^W	Coefficient du critère de poids de sommets dans le calcul du score	0,45
c^D	Coefficient du critère de localisation de sommets dans le calcul du score	0,275
N'	Nombre maximal de sous-graphes voisins	16
r_{max}^p	Ratio maximal de particules pouvant être envoyées	0,125
r	Rapport entre le nombre de mailles et le nombre de sommets	100
r'	Rapport entre la taille du graphe et la taille du graphe quotient	16
R	Degré de réplication	4

TABLE 6.1 – Configuration de la régulation dynamique.

par la formule $\Delta_{charges} = \frac{P^{max}}{P^{moyen}}$ et que la surconsommation mémoire est donnée par la formule $\Delta_{mem} = \frac{Mem^{max}}{Mem^{optimal}}$ (voir chapitre 2, sous-section 2.4.2). Nous avons inséré pour chacun des cas test 128 millions de particules. Nous avons réalisé cette étude comparative pour 32 et 256 ressources de calcul.

6.1.1 Étude de régulation de charge

La figure 6.1 présente les résultats obtenus sur le cas test homogène pour 32 et 256 ressources. Nous observons que la méthode dynamique est très faiblement déséquilibrée avec 32 ressources. En effet, les déséquilibres oscillent entre 5 et 15 %. Le déséquilibre de charge est largement mieux maîtrisé avec l'approche dynamique par rapport à l'approche par partitionnement. Nous observons que, contrairement à l'approche par partitionnement, l'augmentation du nombre de ressources ne cause pas de déséquilibre significatifs pour l'approche dynamique. En effet, les déséquilibres restent stables aux alentours de 10 %. Nous observons tout de même un pic de déséquilibre à 1,2 dans les premières itérations. Ces déséquilibres sont causés par l'algorithme de limitation en nombre de sommets. En effet, toutes les mailles sources sont répliquées ce qui amène un nombre de sommets de graphe, à l'initialisation, assez conséquent. L'algorithme de limitation en nombre de sommets requiert l'envoi de nombreux sommets, entraînant des déséquilibres en nombre de particules.

La figure 6.2 présente les résultats obtenus sur le cas test hétérogène par insertion centralisée pour 32 et 256 ressources. Nous observons que la méthode dynamique est ici aussi très faiblement déséquilibrée. Toutefois, nous observons que l'augmentation du nombre de ressources engendre un léger déséquilibre pour l'approche dynamique. Les déséquilibres augmentent mais restent relativement stables, aux alentours de 15-20 %. Nous observons à l'itération 698 un pic de déséquilibre à 1,34. Nous rappelons que nous avons mis en place des mécanismes de contrôle qui sont appelés tous les X itérations. Lors d'un contrôle, si le déséquilibre en nombre de particules est supérieur à une tolérance choisie, nous invoquons un partitionnement. Pour cette étude expérimentale, nous avons fixé le déséquilibre maximal toléré à 1,3. Le pic de déséquilibre de l'itération 698 est un cas intéressant. En effet, alors que l'algorithme de contrôle est invoqué au début de l'itération 700, le déséquilibre repasse en dessous de la barre de 1,3 à l'itération 699, soit juste avant l'appel de l'algorithme de contrôle. Ensuite, le déséquilibre continue de baisser à l'itération 700 sans faire appel à un outil de partitionnement. Autrement dit, à une itération près un partitionnement aurait été invoqué pour rétablir l'équilibre alors que finalement l'algorithme d'équilibrage de particules y est parvenu. Ce phénomène soulève une question intéressante sur les mécanismes de contrôle. En effet, comment être certain de la pertinence d'un partitionnement ? Répondre à cette question est difficile, d'autant plus que nous savons pas exactement quelles conséquences, sur les performances, auraient eu le partitionnement.

La figure 6.3 présente les résultats obtenus sur le cas test hétérogène par insertion latérale pour 32 et 256 ressources. Nous observons que la méthode dynamique demeure très faiblement déséquilibrée. Nous observons que les pics de déséquilibre sont moins nombreux et plus faibles que sur le cas test précédent. Les déséquilibres sont assez comparables à ceux du cas test homogène. En effet, les déséquilibres se stabilisent aux alentours de 10-15 %. Les résultats obtenus par la régulation dynamique montrent que l'hétérogénéité du domaine ne provoque pas nécessairement des déséquilibres de charge entre les ressources.

6.1.2 Étude de régulation de donnée

La table 6.2 présente les résultats obtenus en surconsommation mémoire sur le cas test homogène. Nous observons que, sur ce cas test, l'approche dynamique obtient le meilleur compromis.

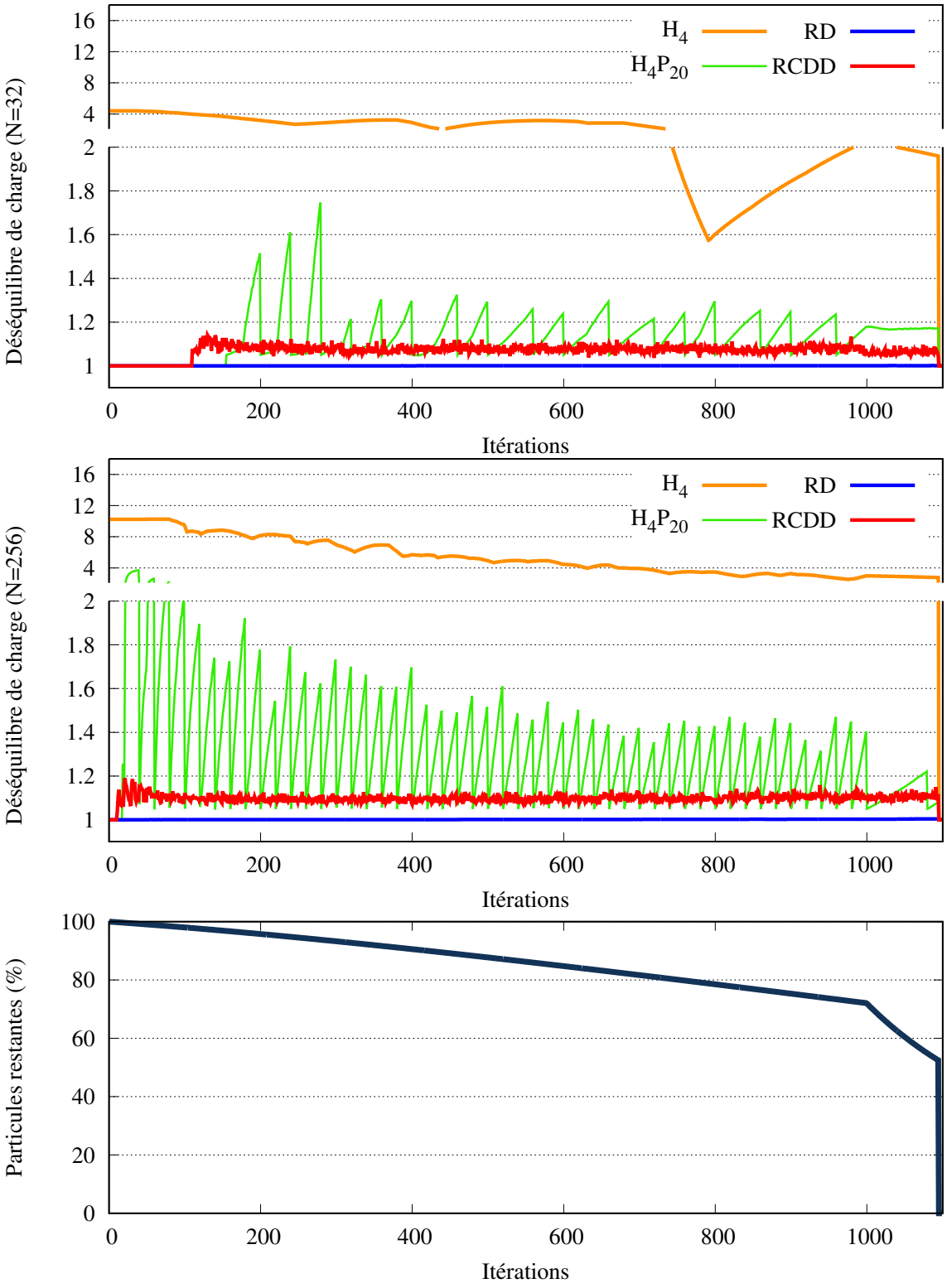


FIGURE 6.1 – Étude du déséquilibre de charge pour le cas test homogène (voir figure 2.3).

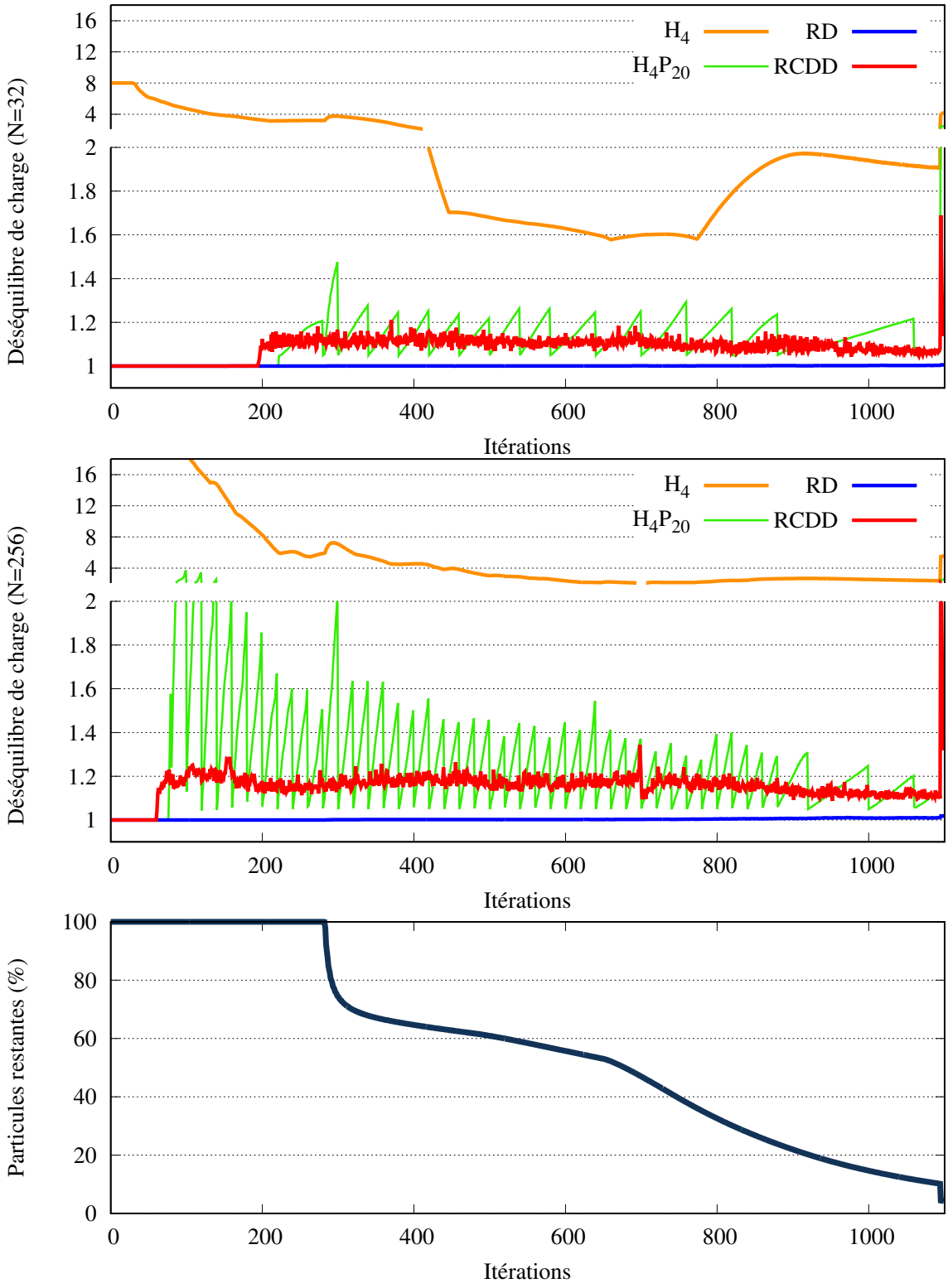


FIGURE 6.2 – Étude du déséquilibre de charge pour le cas test hétérogène par insertion centralisée (2.4).

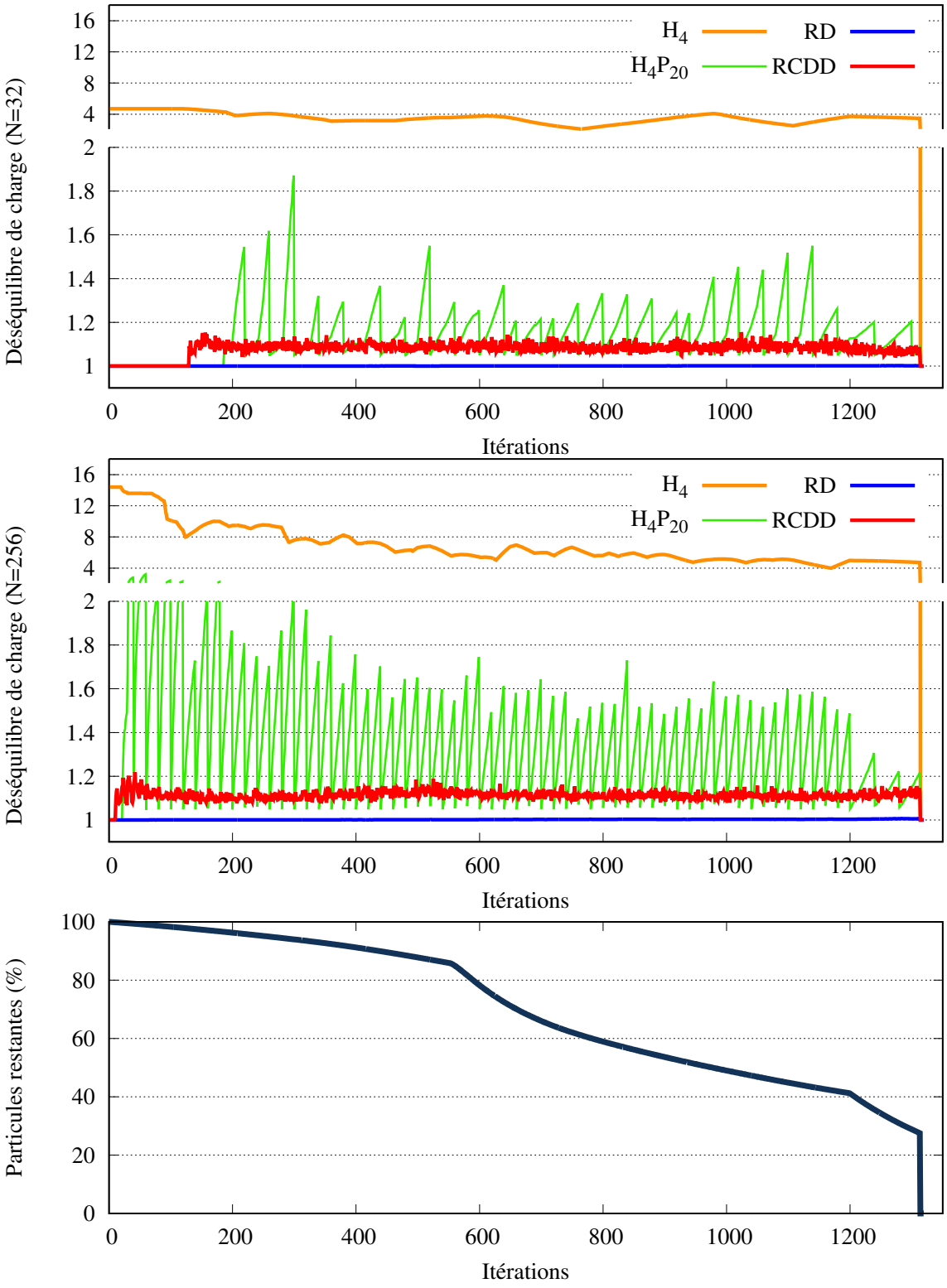


FIGURE 6.3 – Étude du déséquilibre de charge pour le cas test hétérogène par insertion latérale (2.5).

N	Méthodes	H_4	H_4P_{20}	$RCDD$	RD
32	Particules	**4.4**	1.64	1.11	1.0
	Mailles	**4.2**	**4.8**	**3.7**	**26.0**
256	Particules	**10.2**	**3.67**	1.19	1.0
	Mailles	**4.2**	**4.84**	**3.99**	**74.5**

TABLE 6.2 – Surconsommation mémoire pour le cas homogène.

N	Méthodes	H_4	H_4P_{20}	$RCDD$	RD
32	Particules	**8.0**	1.21	1.0	1.0
	Mailles	**4.2**	**4.93**	**3.72**	**21.2**
256	Particules	**64.0**	**3.72**	1.25	1.0
	Mailles	**4.2**	**4.84**	**4.67**	**55.1**

TABLE 6.3 – Surconsommation mémoire pour le cas hétérogène par insertion centralisée.

En effet, elle obtient une surconsommation mémoire maîtrisée relative aux mailles, correspondant au degré de réplication, et une surconsommation mémoire très faible relative aux particules, avec moins de 20 % de surconsommation mémoire.

La table 6.3 présente les résultats obtenus en surconsommation mémoire sur le cas test hétérogène par insertion centralisée. Nous observons que l'approche dynamique demeure le meilleur compromis. Cependant, les surconsommations mémoire obtenues par la régulation dynamique, sont légèrement plus élevées sur ce cas test. Cette augmentation est dans le respect des objectifs de l'approche, les surconsommations mémoire obtenues étant inférieures aux tolérances fixées. Étant donné $\tau_m^{''} = 0,2$ et $R = 4$, la tolérance de surconsommation mémoire maximale relative aux mailles est de 4,8, or nous obtenons 4,67 avec 256 ressources.

La table 6.4 présente les résultats obtenus en surconsommation mémoire sur le cas test hétérogène par insertion latérale. Ce cas test confirme une nouvelle fois que la régulation dynamique est le meilleur compromis. Les surconsommations mémoire obtenues par la régulation dynamique sont légèrement plus élevées par rapport au cas test homogène mais moins élevées que sur le cas test précédent.

Les résultats obtenus par l'approche dynamique en régulation de charge et de donnée sont très encourageants. En effet, les déséquilibres de charge sont maîtrisés sur l'ensemble des cas test, aussi bien avec 32 que 256 ressources. Nous constatons également que les objectifs en restriction de consommation mémoire sont atteints.

N	Méthodes	H_4	H_4P_{20}	$RCDD$	RD
32	Particules	**4.7**	1.76	1.12	1.0
	Mailles	**4.2**	**4.84**	**3.73**	**17.5**
256	Particules	**14.4**	**3.19**	1.21	1.0
	Mailles	**4.19**	**4.88**	**4.35**	**59.2**

TABLE 6.4 – Surconsommation mémoire pour le cas hétérogène par insertion latérale.

6.2 Extensibilité forte

Nous proposons de comparer les temps de restitution de l'approche dynamique avec les méthodes RD, H_R et $H_R P_X$ en extensibilité forte. Nous allons étudier l'évolution du temps d'exécution de 1 à 256 ressources de calcul avec 128 millions de particules insérées.

6.2.1 Efficacité parallèle

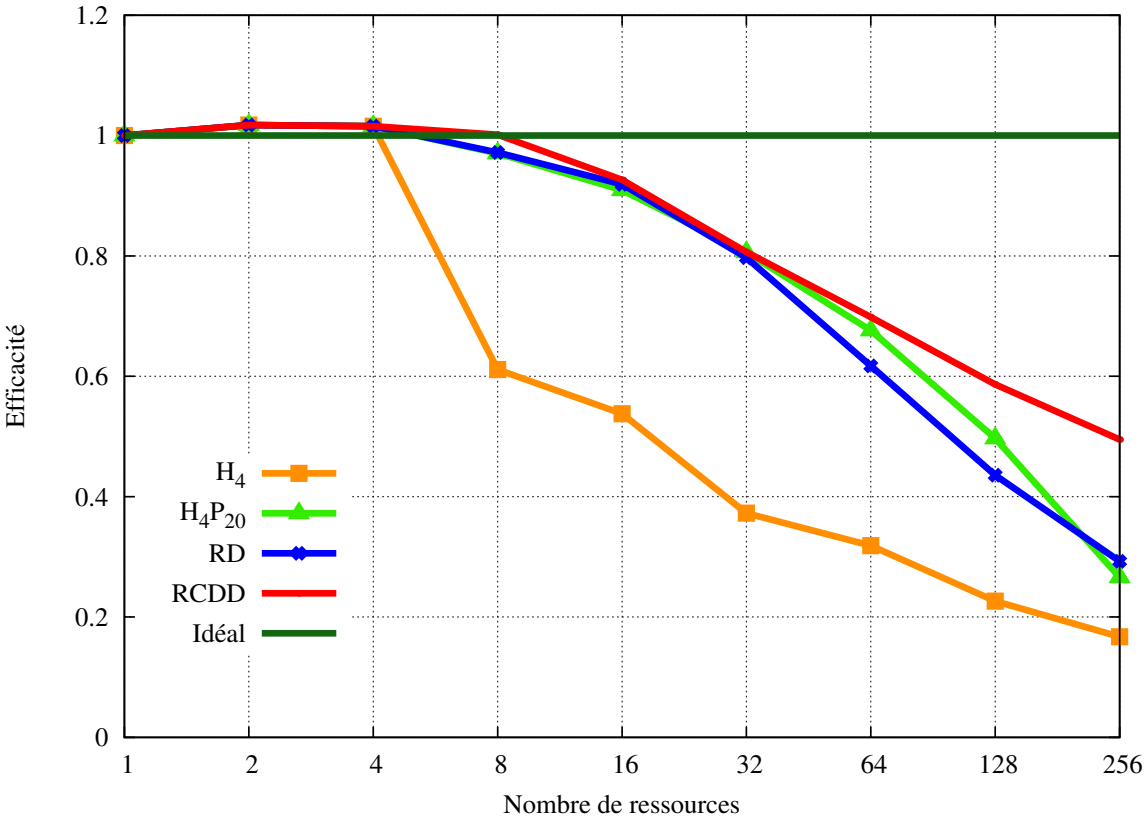


FIGURE 6.4 – Extensibilité forte pour le cas test homogène (meilleur en haut).

La figure 6.4 présente les résultats obtenus en extensibilité forte sur le cas test homogène. Nous constatons que la régulation dynamique obtient de loin les meilleurs résultats, avec une efficacité parallèle près de 2 fois supérieure à celle de la méthode de réplication de domaine avec 256 ressources de calcul. Toutefois, la régulation dynamique perd en efficacité parallèle à partir de 8 ressources. Cette perte de l'efficacité est en partie due à une perte d'efficacité de l'application elle-même.

La figure 6.5 présente les résultats obtenus en extensibilité forte sur le cas test hétérogène par insertion centralisée. Nous constatons que, bien que les écarts soient moins importants, la régulation dynamique obtient, pour ce cas test également, les meilleurs résultats. Nous observons que la courbe suit la même tendance que celle du cas test précédent.

La figure 6.6 présente les résultats obtenus en extensibilité forte sur le cas test hétérogène par insertion latérale. La tendance observée sur les deux cas test précédents se vérifie encore. Les résultats obtenus par la régulation dynamique sur ce cas test sont meilleurs que sur le cas test hétérogène précédent, mais moins bons que sur le cas test homogène.

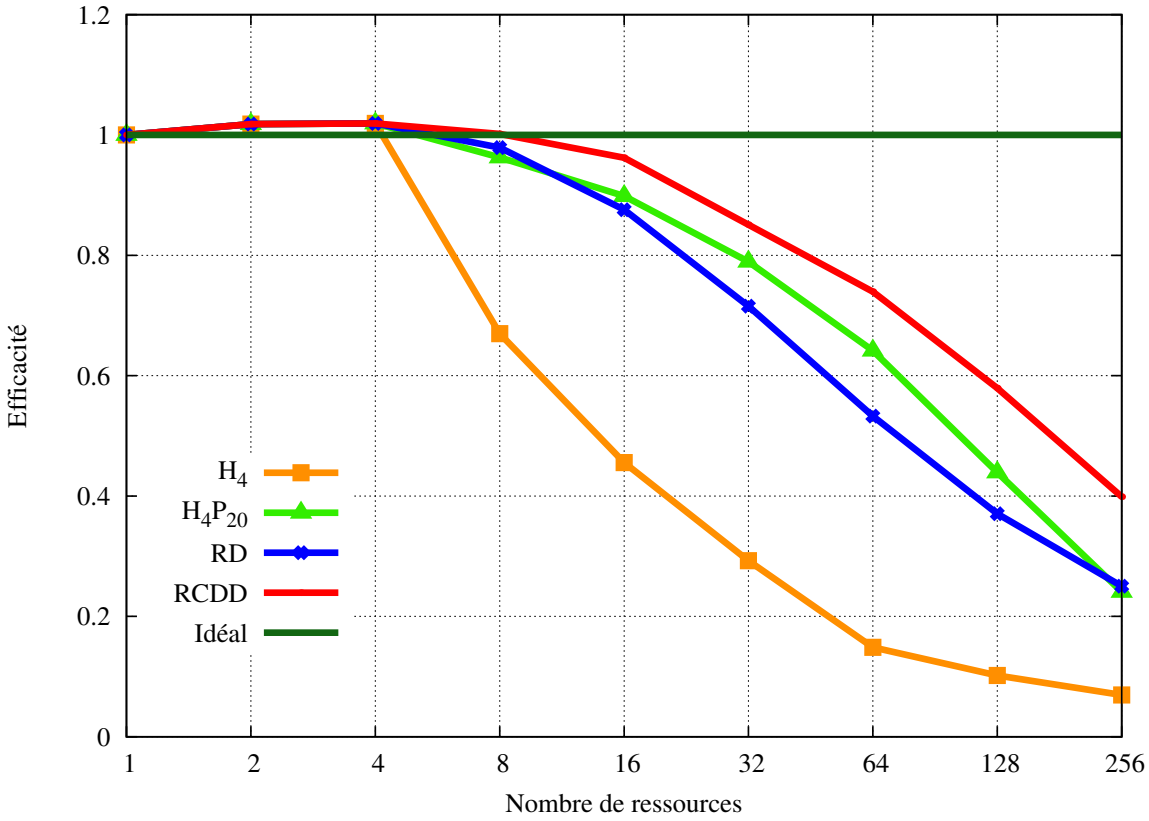


FIGURE 6.5 – Extensibilité forte pour le cas test hétérogène par insertion centralisée (meilleur en haut).

La régulation dynamique a obtenu les meilleurs résultats en extensibilité forte, améliorant de près d'un facteur deux en moyenne les résultats obtenus par la méthode de réplication de domaine. Nous avons cependant constaté une perte de performance à partir de 8 ressources. Nous proposons donc maintenant de profiler l'application pour comprendre les gains de performance observés de la régulation dynamique par rapport aux autres approches, ainsi que la perte d'efficacité constatée.

6.2.2 Profilage

La table 6.5 présente le profilage de l'application pour le cas test homogène en extensibilité forte. Le temps de partitionnement observé correspond au calcul de la partition statique. Ce partitionnement n'implique pas de communication, expliquant le faible coût observé. La section régulation désigne l'ensemble des algorithmes de régulation de charge et de donnée mais aussi l'algorithme de sélection de voisins. Cette section englobe également les échanges de graphes. Il est important de préciser que ces échanges de graphes impliquent une synchronisation entre les ressources concernées. Par conséquent, en cas de déséquilibre de charge et donc de désynchronisation des ressources, les échanges de graphes induiront un temps d'attente pénalisant. Nous rappelons que les temps mesurés correspondent aux temps de la ressource ayant obtenu le temps le plus long entre deux synchronisations globales. Ceci explique que le temps observé en section algorithme diminue entre 32 et 256 ressources. En effet, ce temps est en partie proportionnel au temps de traitement des particules (multiplié par le pourcentage de déséquilibre). Nous observons d'ailleurs que

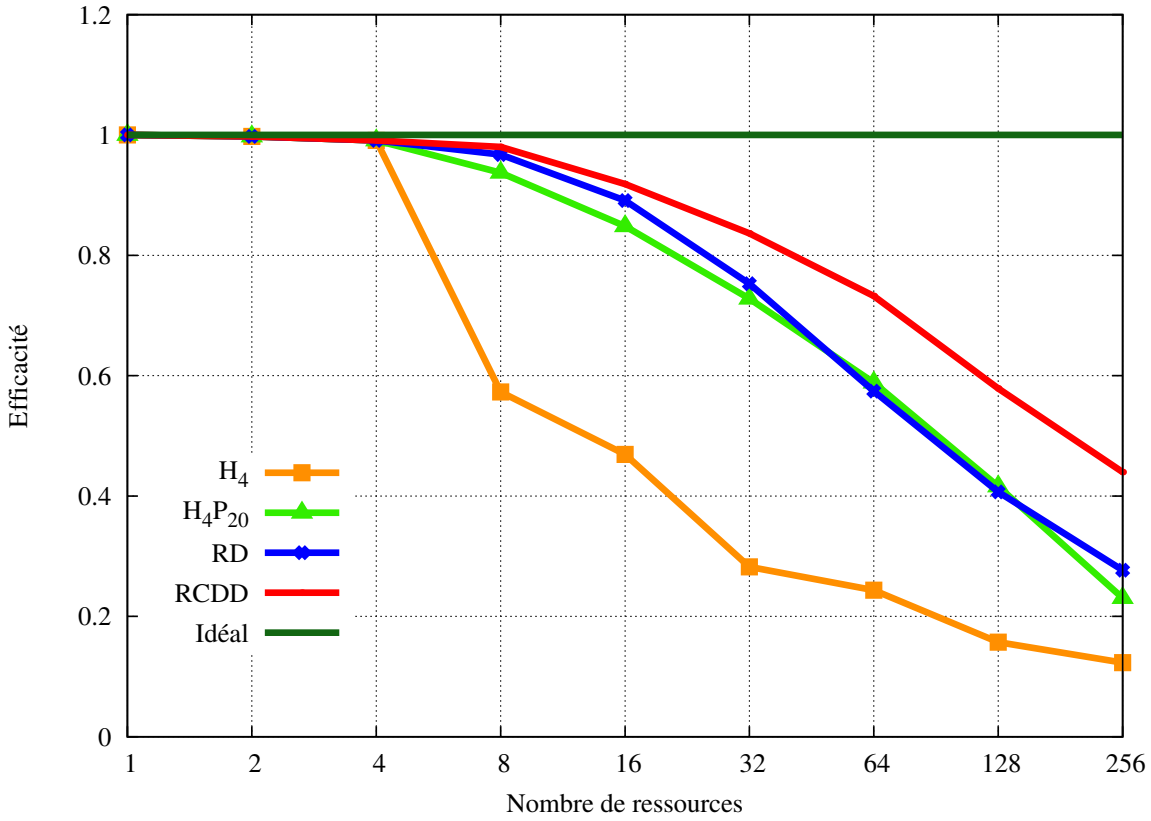


FIGURE 6.6 – Extensibilité forte pour le cas test hétérogène par insertion latérale (meilleur en haut).

ce temps représente moins du quart du temps total avec 256 ressources. Le temps de restitution de l'algorithme n'est donc pas le seul responsable de la perte d'efficacité parallèle.

Le temps de traitement des mailles explique en partie la perte d'efficacité de la régulation dynamique. En effet, outre le fait que la régulation dynamique implique la réplication de certaines mailles et donc un temps de traitement non-optimal des mailles, nous observons que ce temps ne diminue pas de manière proportionnelle au nombre de mailles par ressource. Nous obtenons 18,7 secondes avec 32 ressources et 10,3 secondes avec 256 ressources. Nous avons donc une diminution du temps d'un facteur d'environ 1,8 alors que le nombre de mailles a diminué d'environ un facteur 4. Enfin, nous constatons que les temps de traitement des particules de la méthode de régulation dynamique et ceux de la méthode de réplication de domaine sont pratiquement identiques.

La table 6.6 présente le profilage de l'application pour le cas test hétérogène par insertion cen-

Méthodes	H_4	H_4P_{20}	$RCDD$	RD
Particules + Communications	**176**	**65.6**	39.1	38.4
Mailles	9.78	10.6	10.3	**71.7**
Partitionnement(s)	0.51	**44.8**	0.23	0.0
Régulation (hors partitionnement)	0.0	0.01	**15.1**	0.0
Temps total	186	121	64.7	110

TABLE 6.5 – Profilage (secondes) en extensibilité forte pour le cas homogène.

Méthodes	H_4	H_4P_{20}	$RCDD$	RD
Particules + Communications	**313**	**49.4**	32.1	30.2
Mailles	11.3	11.8	12.2	**60.6**
Partitionnement(s)	0.51	**34.7**	0.23	0.0
Régulation (hors partitionnement)	0.0	0.01	**12.9**	0.0
Temps total	325	95.9	57.4	90.7

TABLE 6.6 – Profilage (secondes) en extensibilité forte pour le cas hétérogène par insertion centralisée.

Méthodes	H_4	H_4P_{20}	$RCDD$	RD
Particules + Communications	**213**	**60.9**	39.1	37.7
Mailles	10.8	11.2	12.0	**67.2**
Partitionnement(s)	0.52	**53.5**	0.22	0.0
Régulation (hors partitionnement)	0.0	0.01	**16.1**	0.0
Temps total	224	126	67.4	105

TABLE 6.7 – Profilage (secondes) en extensibilité forte pour le cas hétérogène par insertion latérale.

tralisée en extensibilité forte. Nous observons la même tendance. Nous observons que le temps passé dans le traitement des mailles est légèrement plus élevé sur ce cas test. Nous avions observé une consommation mémoire concernant les mailles légèrement plus élevée sur ce cas test. L'augmentation ce temps peut donc s'expliquer par le fait que le nombre de mailles par ressource est plus élevé.

La table 6.7 présente le profilage de l'application pour le cas test hétérogène par insertion latérale en extensibilité forte. Les résultats obtenus sur ce cas test confirment les observations faites sur les deux précédents cas test. Nous notons une certaine régularité dans les résultats de la régulation dynamique sur l'ensemble de ces cas test.

6.2.3 Conclusion

La régulation dynamique a obtenu les meilleurs résultats en extensibilité forte, dépassant ainsi la méthode de réplication de domaine. Les gains de performance observées résident en grande partie dans le fait que la régulation dynamique parvient à réduire le nombre de mailles par ressource sans créer de déséquilibre de charge excessifs. Nous proposons maintenant d'étudier les performances de la régulation dynamique en extensibilité faible.

6.3 Extensibilité faible

Nous proposons de comparer les temps de restitution de l'approche dynamique avec les méthodes RD, H_R et H_RP_X en extensibilité faible. Nous allons étudier l'évolution du temps d'exécution de 1 à 256 ressources de calcul avec 4 millions de particules insérées par ressource.

6.3.1 Efficacité parallèle

La figure 6.7 présente les résultats obtenus en temps en extensibilité faible sur le cas test homogène jusqu'à 256 ressources. Nous observons que les courbes de la régulation dynamique et de la méthode de réplication de domaine sont pratiquement confondues. La régulation dynamique est

en effet plus lente de seulement 3 % environ avec 256 ressources. La régulation dynamique parvient donc, sur ce cas test, a atteindre les performances de la méthode de réplication de domaine.

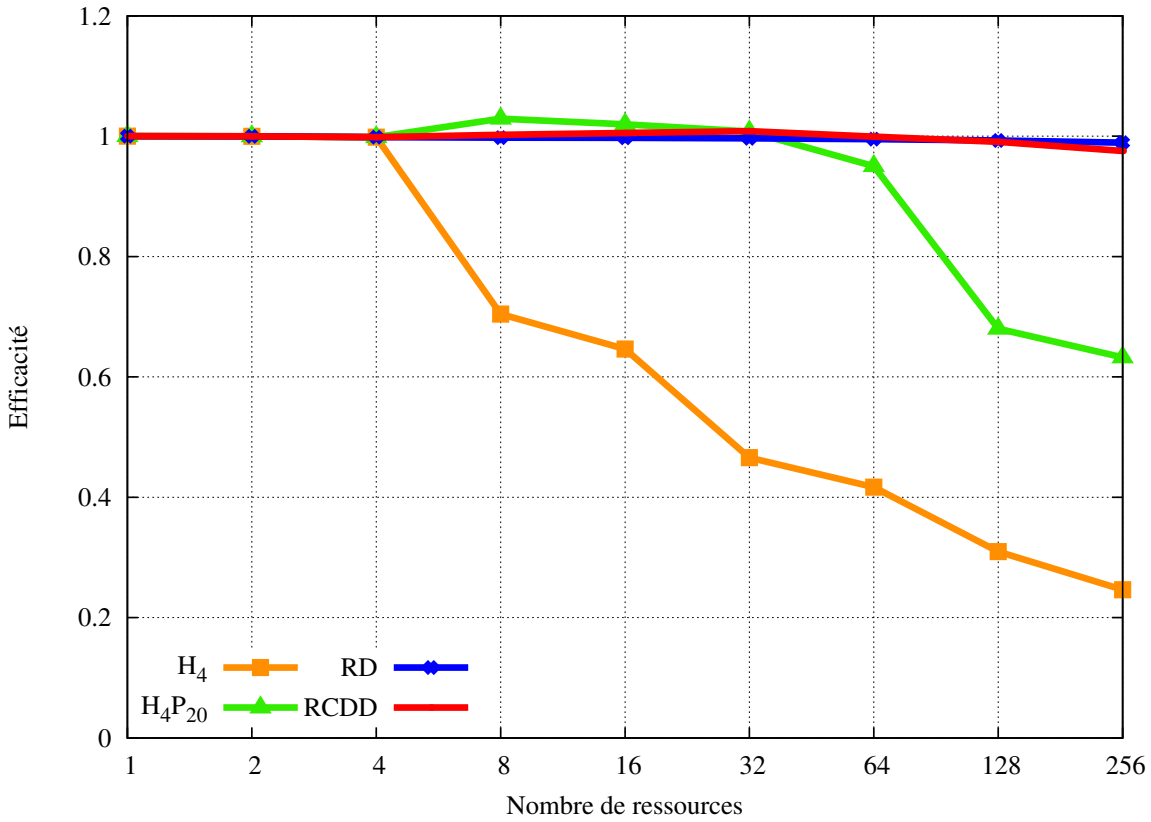


FIGURE 6.7 – Extensibilité faible pour le cas test homogène (meilleur en haut).

La figure 6.8 présente les résultats obtenus en temps en extensibilité faible sur le cas test hétérogène par insertion centralisée jusqu'à 256 ressources. Nous observons que la régulation dynamique est meilleure que la méthode de réplication de domaine jusqu'à 64 ressources. La régulation dynamique obtient même une efficacité supérieure à 1. Étant donné que seul le nombre de particules augmente proportionnellement au nombre de ressources, ce gain se justifie par la décomposition du domaine. En revanche, à partir de 128 ressources, les courbes s'inversent. Nous constatons que la régulation dynamique améliore les résultats de l'approche par partitionnement mais est plus lente d'environ 15 % que la méthode de réplication de domaine. Nous avions constaté plus de déséquilibre sur ce cas test (section 6.1). Il n'est donc pas étonnant d'observer un surcoût plus important sur ce cas test.

La figure 6.9 présente les résultats obtenus en temps en extensibilité faible sur le cas test hétérogène par insertion latérale jusqu'à 256 ressources. Nous constatons que la régulation dynamique obtient globalement les meilleurs temps. En effet, la régulation dynamique est meilleure que la méthode de réplication de domaine de 8 à 128 ressources. Les deux méthodes obtiennent pratiquement le même temps avec 256 ressources : autour de 320 secondes. Les déséquilibres de charge peu élevés sur ce cas test n'ont pas dégradé les performances de la régulation dynamique.

Les résultats obtenus par la régulation dynamique sont très prometteurs. En effet, nous atteignons, voire même dépassons, les résultats de la méthode de réplication de domaine. Nous proposons maintenant de profiler l'application afin de détailler et d'interpréter plus finement ces résultats.

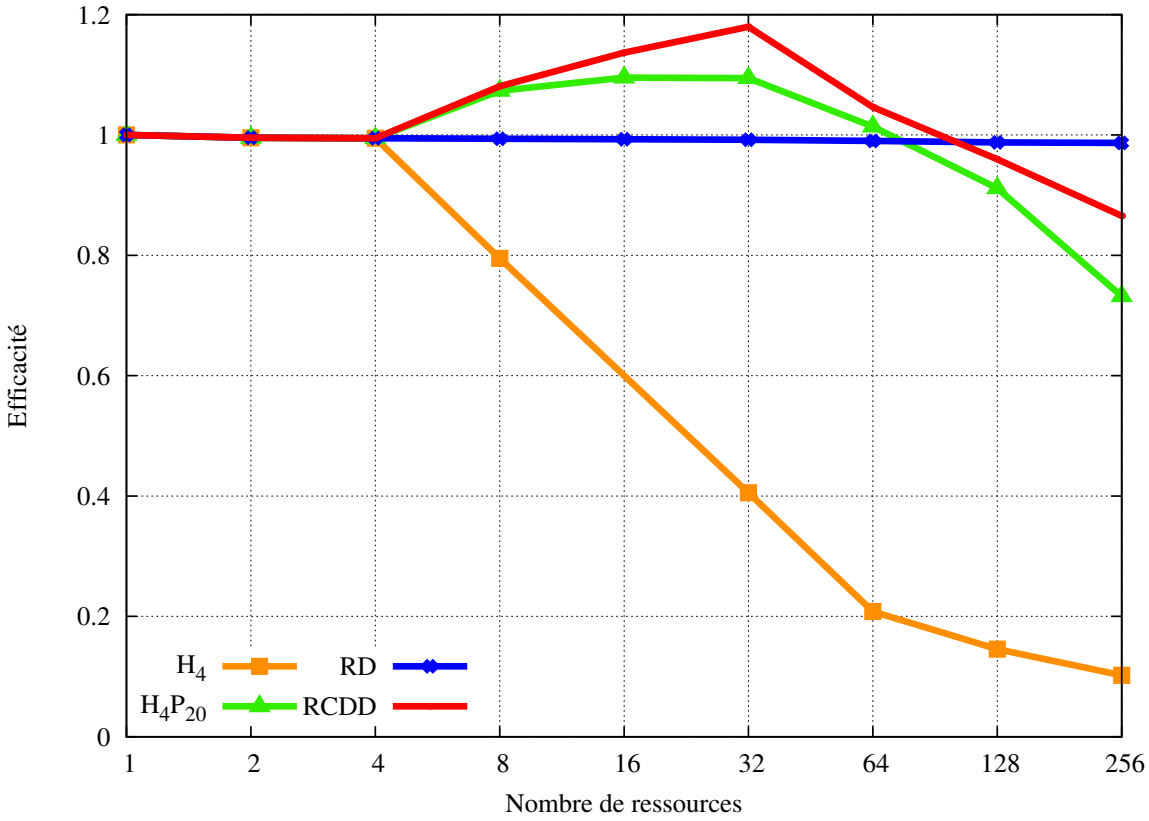


FIGURE 6.8 – Extensibilité faible pour le cas test hétérogène par insertion centralisée (meilleur en haut).

6.3.2 Profilage

La table 6.8 présente le profilage de l'application pour le cas test homogène en extensibilité faible. Nous remarquons que la réduction du temps passé dans le traitement des mailles permet à la régulation dynamique d'approcher les résultats de la méthode de réplication de domaine. Le fait que la régulation dynamique ne réussisse pas à dépasser la méthode de réplication de domaine à 256 ressources est la conséquence du temps passé dans la partie régulation, avec près de 61 secondes. Nous rappelons que ce temps est grandement influencé par les déséquilibres de charge. L'algorithme de sélection des voisins, impliquant une communication collective, est peut-être une cause du temps passé en section algorithme. Néanmoins, les résultats en extensibilité forte nous amènent à penser que le coût de la communication collective n'est pas la cause principale. En effet, nous avions obtenu 15 secondes pour la section algorithme avec 256 ressources. La communication collective, liée à la sélection des voisins, étant indépendante du nombre de particules, a donc un coût strictement inférieur à 15 secondes.

La table 6.9 présente le profilage de l'application pour le cas test hétérogène par insertion centralisée en extensibilité faible. Nous observons que, par rapport au cas test précédent, le temps passé dans les sections particules+communications et régulation a augmenté. Cette augmentation explique le surcoût légèrement supérieur sur ce cas test par rapport à la méthode de réplication de domaine.

La table 6.10 présente le profilage de l'application pour le cas test hétérogène par insertion latérale en extensibilité faible. Nous observons des résultats sensiblement proches de ceux du cas

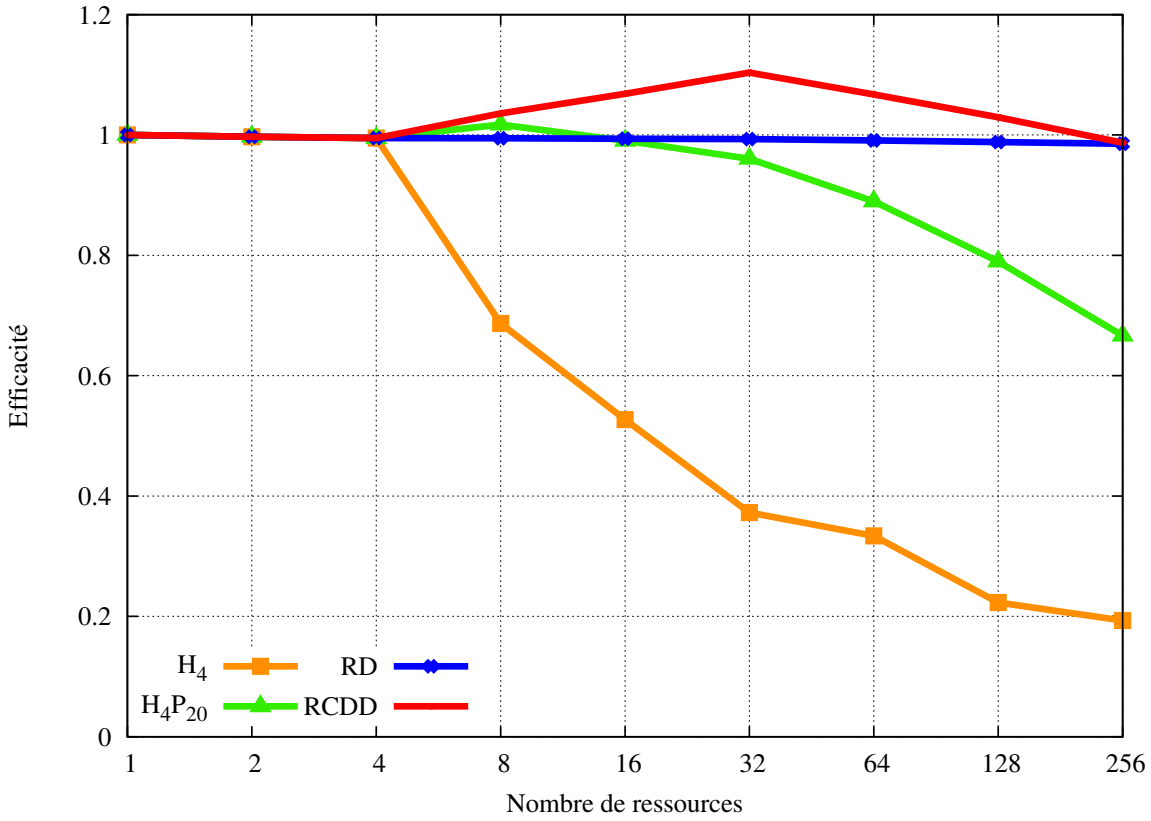


FIGURE 6.9 – Extensibilité faible pour le cas test hétérogène par insertion latérale (meilleur en haut).

test homogène. La régulation dynamique parvient à atteindre les performances de la méthode de réplication de domaine. Ce résultat provient du fait que le temps de restitution du traitement des mailles est proportionnellement assez conséquent sur ce cas test. Le gain obtenu par le fait de répartir les mailles permet de compenser le surcoût lié aux algorithmes.

6.3.3 Conclusion

Les résultats obtenus par la régulation dynamique en extensibilité faible sont très satisfaisants. En effet, la régulation dynamique obtient pratiquement les mêmes résultats en temps que la méthode très équilibrée de réplication de domaine. La répartition des mailles de la régulation dynamique permet de réduire considérablement le temps de traitement des mailles. Les déséquilibres

Méthodes	H_4	H_4P_{20}	$RCDD$	RD
Particules + Communications	**1326**	**423**	268	263
Mailles	9.86	10.6	10.4	**67.7**
Partitionnement(s)	0.51	**75.0**	0.23	0.0
Régulation (hors partitionnement)	0.0	0.01	**60.9**	0.0
Temps total	1336	508	340	331

TABLE 6.8 – Profilage (secondes) en extensibilité faible pour le cas homogène.

Méthodes	H_4	H_4P_{20}	$RCDD$	RD
Particules + Communications	**2530**	**268**	217	184
Mailles	11.7	12.0	12.7	**77.0**
Partitionnement(s)	0.51	**61.5**	0.24	0.0
Régulation (hors partitionnement)	0.0	0.01	**70.7**	0.0
Temps total	2542	341	300	261

TABLE 6.9 – Profilage (secondes) en extensibilité faible pour le cas hétérogène par insertion centralisée.

Méthodes	H_4	H_4P_{20}	$RCDD$	RD
Particules + Communications	**1606**	**351**	252	239
Mailles	10.6	11.1	12.3	**76.1**
Partitionnement(s)	0.53	**89.0**	0.24	0.0
Régulation (hors partitionnement)	0.0	0.01	**50.1**	0.0
Temps total	1617	451	315	315

TABLE 6.10 – Profilage (secondes) en extensibilité faible pour le cas hétérogène par insertion latérale.

de charge observés, peu nombreux, ne dégradent que faiblement les performances de la méthode.

6.4 Conclusion

La régulation dynamique consistant à réaliser des opérations de régulation de charge et de donnée localement entre les ressources a obtenu globalement les meilleurs résultats sur cette étude expérimentale. En effet, cette méthode garantit une limitation de la consommation mémoire tout en maintenant une répartition de la charge a un niveau satisfaisant, le déséquilibre moyen étant aux alentours de 10-15%.

Le surcoût de l'algorithme est modéré celui-ci étant davantage dû aux déséquilibres de charge qu'au coût des algorithmes eux-mêmes. Les échanges de graphes impliquent une synchronisation entre les ressources dont leurs sous-graphes sont voisins. Les déséquilibres de charge pénalisent donc les performances des échanges de graphes. Nous avons observé que le coût des algorithmes était faible, celui-ci n'excédant pas quelques secondes en extensibilité forte.

Alors que la régulation dynamique approche les performances de la méthode de réplication de domaine en extensibilité faible avec un surcoût n'excédant pas 15%, elle dépasse les résultats de la méthode de réplication de domaine en extensibilité forte. La régulation dynamique que nous avons proposée permet donc de résoudre la problématique que nous avions énoncée en introduction de cette thèse. Nous proposons maintenant de comparer notre approche avec d'autres études portant sur des problématiques similaires.

Chapitre 7

État de l'art

Les études menées durant cette thèse se sont intéressés aux problématiques de régulation de charge et de donnée pour les applications de transport de particules Monte-Carlo. Nous avons décrit des objectifs de régulation (voir chapitre 1, section 1.1). Ces objectifs de régulation consistent à garantir une bonne répartition de la charge de calcul et des données avec un surcoût de régulation modéré, ce dernier devant être compensé par une meilleure régulation de charge et donc une meilleure performance.

Nous avons sélectionné comme cadre d'étude les architectures massivement parallèles et homogènes. Nous avons utilisé le paradigme de programmation parallèle *MPI+OpenMP*, celui-ci étant l'une des options les mieux adaptées et optimisées à ce cadre d'étude. De plus, l'application, sur laquelle se sont basés nos travaux, est issue d'un code de simulation développé avec *MPI+OpenMP* (voir chapitre 2).

Les problématiques de régulation de charge et de donnée ont déjà été largement traitées par la communauté scientifique. Nous proposons de dresser dans ce chapitre une évaluation des approches proposées et de fournir une analyse comparative avec notre approche. Nous effectuerons une comparaison selon plusieurs critères, détaillés ci-dessous.

Environnements logiciels de parallélisation : les choix d'environnements logiciels de parallélisation permettent d'orienter la problématique selon un axe spécifique. Ils imposent également un cadre logiciel impactant l'intégration des méthodes de régulations dans des codes de simulation existants. Nous comparerons les choix d'environnements logiciels de parallélisation. Nous justifierons également notre choix.

Type de régulation : la régulation de charge et de donnée constitue l'objectif de nos travaux. Nous proposons donc d'analyser et de comparer nos choix avec ceux effectués par les approches citées.

Analyse de performance : les mécanismes de régulation mis en œuvre ont pour objectif de résoudre la problématique de régulation de charge et de donnée. Nous évaluerons dans quelle mesure cet objectif est atteint ou non. De plus, ces mécanismes ont un coût. Nous ferons alors une analyse du coût des méthodes.

Dans la communauté scientifique, nous trouvons des études susceptibles de répondre aux problématiques de régulation de charge et de donnée pour le transport de particules Monte-Carlo. Nous proposons de commencer par discuter des environnements et interfaces de régulation dynamique de charge et leurs applications au transport de particules Monte-Carlo.

7.1 Approches génériques de régulation dynamique de charge

La régulation dynamique de charge n'est pas une problématique spécifique au transport de particules Monte-Carlo. Des études se sont intéressées à cette problématique dans un contexte assez générique. Nous allons discuter en particulier d'un outil appelé *DLB (Dynamic Load Balancing)* [39], d'un environnement de programmation parallèle nommé *Charm++* [40][41] et d'une implantation de la norme *MPI*, adaptée pour la régulation de charge, appelée *Adaptive MPI* [42][43].

7.1.1 Attribution dynamique de ressources de calcul

Rééquilibrer la répartition de la charge de calcul dynamiquement entre des ressources de calcul revient à attribuer les mêmes charges à toutes les ressources. Chaque nœud de calcul étant composé de plusieurs cœurs de calcul, nous pouvons alors rééquilibrer la charge de calcul en attribuant, de manière proportionnelle à la charge, les cœurs de calcul. Cette approche a été proposée et implantée dans un outil nommé *DLB (Dynamic Load Balancing)* [39]. Cet outil permet à des processus ou des *threads* d'utiliser des cœurs de calcul sur lesquels ils n'étaient pas initialement ordonnancés.

Discussion

Environnements logiciels de parallélisation : l'outil *DLB* peut-être utilisé pour les applications de type *MPI+OpenMP*. Cela correspond au paradigme de programmation parallèle que nous avons choisi. En revanche, les techniques de régulation de charge imposent, pour être applicables, qu'il y ait au moins deux processus *MPI* par nœud de calcul. Or, nous avons opté pour une parallélisation *OpenMP* seule en intra-nœud. La minimisation du nombre de processus *MPI* permet de réduire la consommation mémoire par nœud de calcul. Nous évitons notamment de dupliquer certaines données sur un nœud de calcul. Nous pouvons prendre comme exemple les mailles fantômes. De plus, réduire le nombre de processus *MPI* permet de réduire le nombre de processus concernés par la régulation de charge et donc de diminuer la complexité de la régulation de charge. L'utilisation d'*OpenMP* seule en intra-nœud permet de minimiser le nombre de processus *MPI*.

Type de régulation : la régulation de charge se base sur une analyse de l'utilisation réelle des cœurs de calcul. Par conséquent, la régulation de charge s'applique à un grain très fin. Cette technique est donc plus précise que notre approche de régulation de charge. En revanche, la régulation de charge est limitée à l'intra-nœud. En intra-nœud, nous avons opté pour l'ordonnancement dynamique *OpenMP*. Cette technique a l'avantage d'être facile à implanter.

Analyse de performance : alors que cette approche permet une régulation de charge précise, elle ne permet pas de résoudre la problématique de régulation de charge et de donnée en inter-nœuds. En effet, avec l'outil *DLB*, un nœud de calcul demeurera surchargé en cas de déséquilibre, alors que nous sommes parvenus à réaliser de la régulation de charge en inter-nœuds.

7.1.2 Régulation dynamique par méthode haut niveau

La régulation de charge nécessite au préalable de quantifier et représenter la charge de calcul. L'abstraction des charges de calcul par des objets manipulables permet de réaliser plus facilement des transferts de charge de calcul entre les ressources. *Charm++* [40][41] est un environnement de programmation parallèle permettant de définir des objets, représentant des tâches à réaliser, nommés *chares*. Ces objets sont suffisamment abstraits pour permettre la définition la charge de calcul et les volumes de donnée d'une application. Cet objets sont migrables entre les nœuds et

entre les cœurs de calcul. La régulation de charge et de donnée est donc réalisée par une technique de migration de ces objets.

Discussion

Environnements logiciels de parallélisation : *Charm++* définit une interface de programmation parallèle reposant sur le langage C++. L'utilisation de *Charm++* pour réaliser de la régulation de charge pour des codes de simulation écrit initialement en *MPI* imposerait une réécriture complète du code. Or, nous avons réalisé notre étude sur un prototype issu d'un code de simulation développé en *MPI+OpenMP*.

Type de régulation : *Charm++* définit un ensemble d'algorithmes dynamiques de régulation de charge [44]. Ces algorithmes reposent sur des données statistiques précises obtenues pendant l'exécution. Cependant, le caractère très déséquilibré et dynamique du transport de particules Monte-Carlo nous amène à définir des algorithmes plus spécifiques. *Charm++* permet également l'intégration d'algorithmes de régulation définis par l'utilisateur. Il serait donc intéressant d'utiliser *Charm++* en y intégrant nos algorithmes de régulation de charge et de donnée.

7.1.3 Régulation dynamique par migration de processus virtuels *MPI*

Adaptive MPI [42][43] est une implantation de la norme *MPI* comportant quelques spécificités. *Adaptive MPI* repose sur le principe de virtualisation de processus *MPI* [45]. Les processus *MPI* sont représentés par des *threads* utilisateurs. *Adaptive MPI* met en œuvre ses fonctionnalités de la régulation en utilisant *Charm++*. Les processus virtuels sont encapsulés dans des *chares* [46]. La régulation de charge s'opère alors en migrant les *chares* entre les nœuds de calcul et entre les cœurs de calcul.

Discussion

Environnements logiciels de parallélisation : *Adaptive MPI* est une implantation de la norme *MPI*. Cependant, les particularités de cette implantation nécessitent l'adaptation des codes de simulation pour migrer d'une implantation *MPI* standard vers une implantation *Adaptive MPI*. Par exemple, les accès aux variables globales nécessitent un traitement spécifique. De plus, la méthode de régulation utilisée par *Adaptive MPI* repose sur le principe de la surcharge. Cela signifie que le nombre de processus *MPI* est supérieur au nombre de cœurs. Or, nous avons privilégié le paradigme de programmation parallèle hybride en déployant un seul processus *MPI* par nœud de calcul.

Type de régulation : La méthode de régulation de charge et de donnée d'*Adaptive MPI* consiste à migrer des processus virtuels *MPI* entre les nœuds de calcul et entre les cœurs. Ces migrations s'appuient sur une analyse statistique précise obtenue à l'aide du support *Charm++*. En revanche, pour être efficace, cette méthode nécessite plusieurs processus virtuels *MPI* par cœur. Cette propriété implique une augmentation du nombre de processus *MPI*. Appliquée à une décomposition de domaine, cela reviendrait à augmenter le nombre de sous-domaines et donc d'augmenter la complexité du problème de régulation de charge et de donnée. En ne déployant qu'un seul processus *MPI* par nœud, nous avons minimisé la complexité du problème à résoudre.

Analyse de performance : La migration de processus virtuels *MPI* nécessite le transfert des données de ces processus. L'augmentation de processus virtuels *MPI* amène des déséquilibres de charge plus importants et, par conséquent, des migrations plus nombreuses. Le coût des migrations

est alors une sérieuse limitation de cette approche. De plus, il existe des supports exécutifs *MPI* davantage optimisés. Nous avons donc opté pour l'utilisation de supports exécutifs *MPI* constructeurs très performants.

7.2 Régulation dynamique appliquée la méthode de parallélisation hybride décomposition-réplication de domaine

Nous avons observé au chapitre 3 que l'utilisation de la méthode de parallélisation hybride décomposition-réplication de domaine, sans mécanisme de redistribution des charges de calcul et des données, ne permettait pas d'atteindre des performances satisfaisantes pour des applications irrégulières comme le transport de particules Monte-Carlo. En effet, lorsque l'essentiel de la charge se situe dans le même sous-domaine, les ressources appartenant à ce sous-domaine sont surchargées par rapport aux ressources appartenant aux autres sous-domaines.

Pour minimiser les déséquilibres de charge de la méthode de parallélisation hybride décomposition-réplication de domaine, des approches de régulation de charge appliquées à la méthode de parallélisation hybride ont été proposées [47][48]. Ces approches se sont focalisées sur des techniques d'attribution de sous-domaines aux processus.

7.2.1 Attribution dynamique de sous-domaine par processus

En utilisant une méthode de décomposition hybride, les sous-domaines peuvent être déséquilibrés. Pour pallier ce problème, une approche d'attribution dynamique de sous-domaine aux processus a été proposée [47]. Cette approche consiste à ce que chaque processus traite un sous-domaine de telle sorte que le nombre de processus traitant un sous-domaine dépende de la charge totale de ce sous-domaine. Chaque sous-domaine doit être traité par au moins un processus *MPI*. La répartition des sous-domaines est déterminée dynamiquement. Enfin, des mécanismes permettent d'assurer l'équilibrage de charge au sein d'un sous-domaine.

Discussion

Environnements logiciels de parallélisation : Cette approche se focalise sur la problématique de parallélisation *MPI*. Nous pourrions donc utiliser ce type d'approche dans un contexte de parallélisation plus général basé sur *MPI+OpenMP*.

Type de régulation : La technique de régulation consiste à attribuer à chaque processus *MPI* un sous-domaine de telle sorte que le nombre de processus *MPI* traitant un sous-domaine soit déterminé en fonction de la charge de calcul de celui-ci. Pour déterminer la répartition des sous-domaines, la méthode nécessite une vision globale de la répartition de la charge de calcul entre les sous-domaines. Cette méthode de régulation est donc centralisée. À l'inverse, notre approche dynamique consiste à ce que chaque processus puisse faire de la régulation de charge et de donnée avec un minimum d'informations. Notre méthode de régulation est distribuée et locale.

Analyse de performance : Cette approche, parce qu'elle repose sur une décomposition du domaine, a l'avantage de garantir une limitation de la consommation mémoire en ce qui concerne les données du maillage. Notre approche n'est pas aussi robuste vis-à-vis de la répartition des mailles. En revanche, l'approche centralisée a pour inconvénient de nécessiter davantage de synchronisations entre tous les processus qu'une approche locale. De plus, le changement de sous-domaine d'un processus implique que celui-ci échange l'intégralité de ses données. Ce procédé est très

coûteux en termes de volume de communication. En comparaison, l'approche dynamique et locale que nous avons proposé limite le nombre et le volume de communication. De plus, lorsque la charge est très inégalement répartie entre les sous-domaines, la répartition en fonction de la charge des sous-domaines peut ne pas suffire à répartir correctement la charge. Par exemple, si nous avons 8 sous-domaines dont un seul possède 90% de la charge et 16 processus, nous aurions alors 9 processus traitant 90% du problème et 7 processus traitant seulement 10% du problème. Nous avons observé que notre approche dynamique était capable d'équilibrer la charge de calcul dans des cas extrêmes (voir chapitre 2, figure 2.4).

7.2.2 Attribution hybride statique-dynamique de sous-domaines par processus

Pour résoudre la problématique de répartition des sous-domaines, une approche d'attribution hybride statique-dynamique de sous-domaines aux processus a été proposée [48]. Cette approche s'appuie sur une décomposition de domaine classique. Chaque processus traite un sous-domaine attribué de manière statique à l'initialisation. Puis, en fonction de la répartition de la charge, les processus traitent un second sous-domaine choisi dynamiquement.

Discussion

Environnements logiciels de parallélisation : Cette approche se focalise également sur la problématique de parallélisation *MPI*. Nous pourrions donc utiliser ce type d'approche dans un contexte de parallélisation *MPI+OpenMP*.

Type de régulation : Pour les mêmes raisons que l'approche précédente, la méthode de régulation est elle aussi centralisée. Chaque sous-domaine est attribué statiquement à un processus. Chaque processus est chargé de garantir la répartition de la charge de calcul entre les processus partageant son sous-domaine statique. Chaque processus est également chargé de réaliser les communications avec les sous-domaines voisins. La méthode de régulation est donc centralisée au sein de chaque sous-domaine. En comparaison, nous avons fait le choix de permettre à chaque processus d'intervenir dans la régulation de charge.

Analyse de performance : Cette approche permet de réduire considérablement la consommation mémoire, chaque processus traitant exactement deux sous-domaines. En comparaison, notre approche dynamique permet également cette limitation en choisissant $R = 2$. Néanmoins, notre approche est moins robuste et cette restriction mémoire pourrait ne pas être respectée dans certaines configurations. Qui plus est, la décomposition des données géométriques et des sections efficaces n'a pas encore été traitée.

Cependant, le coût de synchronisation de l'approche précédente s'applique également à cette approche. Le volume de communication lorsqu'un processus change de sous-domaine est en revanche deux fois moins important en moyenne que l'approche précédente. En effet, le sous-domaine statique est systématiquement conservé, par conséquent les données de ce sous-domaine ne sont pas échangées. Néanmoins, le volume de communication reste élevé par rapport à notre approche dynamique.

Le processus auquel un sous-domaine a été attribué statiquement doit réaliser les communications avec les sous-domaines voisins mais aussi la régulation de charge au sein de son sous-domaine. Cette propriété est un goulot d'étranglement. Cette saturation pourrait être réduite en intégrant une méthode de communication entre les sous-domaines [49] et une méthode de régulation de charge au sein d'un sous-domaine basée sur le principe Round-Robin [50] [51]. En comparaison, notre approche reposant sur de la régulation dynamique et locale permet de ne saturer aucun processus.

7.3 Décomposition de donnée de simulation relatives au domaine et parallélisme de particules

Des déséquilibres de charge apparaissent lorsqu'une décomposition de domaine est utilisée. La réplication de domaine permet de maintenir les déséquilibres de charge entre les ressources à un niveau proche de l'optimal. En effet, le principe de la réplication de domaine permet à chaque ressource de traiter n'importe quelle particule peu importe où elle se trouve. En effet, le traitement d'une particule nécessitant l'accès aux données de la maille où elle se situe, le stockage mémoire de toutes les mailles pour chaque ressource permet le traitement de n'importe quelle particule. Le stockage mémoire implique néanmoins une surconsommation mémoire excessivement importante.

7.3.1 Approche par serveur de données

Une solution a été proposée afin de contourner le problème de la consommation mémoire [52][38][53]. L'idée est de permettre le traitement de n'importe quelle particule par n'importe quelle ressource, tout en distribuant les données relatives au domaine. Pour permettre cela, l'approche se base sur des accès distants par communications unidirectionnelles. La méthode utilise, par exemple, la fonction *MPI_Get* pour accéder aux données de simulation distantes permettant le traitement des particules et la fonction *MPI_Accumulate* pour mettre à jour les données de simulation distantes de résultats.

Discussion

Environnements logiciels de parallélisation : Cette approche s'appuie sur les communications unidirectionnelles définies par la norme *MPI*. Les fonctions utilisées sont définies dès la norme *MPI-2* [7]. L'utilisation de cette technique n'impose pas de changer d'interface de programmation parallèle lorsque l'application cible est développée en *MPI*. En revanche, le traitement des particules et le calcul de leurs contributions, lorsqu'elles sont localisées sur des mailles distantes, nécessitent des mécanismes permettant de communiquer avec le serveur possédant la donnée. Cela oblige donc d'adapter l'algorithme de calcul sur les particules. Cette méthode est donc très intrusive. À l'inverse, notre algorithme de régulation dynamique de charge s'applique en dehors de la phase de calcul des contributions des particules aux mailles.

Type de régulation : L'approche par serveur de donnée repose sur une décomposition statique du maillage et une décomposition statique des particules. Les particules sont équitablement réparties entre les processus. Les données du maillage sont distribuées entre des processus désignés comme étant des serveurs. Le nombre de serveurs est inférieur au nombre de processus. Il est proposé plusieurs degrés de décomposition. Ces données comprennent la géométrie décrivant le domaine de simulation, les sections efficaces permettant de calculer les interactions des particules et nous avons les statistiques correspondant au résultat de la simulation. Nous pouvons décomposer soit l'ensemble de toutes les données du maillage, soit seulement quelques unes. Par exemple, il est possible de dupliquer les données de géométrie et les sections efficaces de telle sorte qu'un processus n'ait pas besoin de lire au préalable les données à distance. Notre approche s'est focalisée sur la décomposition des données statistiques uniquement, les données géométriques et les sections efficaces pouvant être calculées localement. Néanmoins, notre approche permet de réaliser une décomposition de toutes les données du maillage, le nombre de mailles par processus étant dynamiquement limité. En utilisant l'approche par serveur de donnée, chaque processus peut traiter n'importe quelle particule, et les mécanismes de régulation de charge ne sont alors pas nécessaires.

Cette approche se focalise davantage sur la problématique des accès concurrents aux données du maillage.

Analyse de performance : Cette approche permet d'obtenir une répartition de la charge de calcul équivalente à celle de la réplication de domaine. Cette méthode permet également de limiter la consommation mémoire, les données du maillage étant décomposées. En revanche, le nombre de serveurs étant inférieurs au nombre de processus, les données ne sont pas parfaitement décomposées ayant pour conséquence une surconsommation mémoire. De plus, l'approche de décomposition de donnée possède un coût de communication bien plus élevé en comparaison de notre approche de régulation dynamique. En effet, le traitement des particules sur une maille distante nécessite une communication. Il faut envoyer potentiellement $Ns - 1$ messages où Ns est le nombre de serveurs. Des problèmes de contention apparaissent quand plusieurs processus veulent mettre à jour les mêmes données distantes.

7.3.2 Accessibilité directe de donnée distantes

GVR (Global View Resilience) [54][55] est une approche de programmation de type *PGAS (Partitioned Global Address Space)* [56] consistant à exprimer une vision globale de la mémoire, les données étant distribuées. Une étude, s'appuyant sur le principe de la décomposition de donnée et du parallélisme de particules [57], a proposée d'utiliser *GVR* afin de gérer les communications de donnée. Cette étude propose d'accéder aux données distantes de manière directe en utilisant des tableaux de donnée partagés entre les processus.

Discussion

Environnements logiciels de parallélisation : Les approches de types *PGAS* permettent une meilleure programmabilité. En revanche, l'utilisation de cette méthode obligerait la réécriture complète d'une application développée en *MPI*. Nous privilégions donc les approches reposant des support exécutifs *MPI* et *OpenMP*.

Type de régulation : Cette méthode s'appuie sur le même principe que celles décrites précédemment à la différence que cette fois il y a autant de serveurs de donnée que de processus. Les accès aux données ne sont plus réalisés par passage de message mais par des accès directs aux données indexées dans un tableau. Les communications sont alors implicites. Notons que cette approche ne décompose que les données statistiques, celles-ci impactant considérablement la consommation mémoire de l'application.

Analyse de performance : Cette étude propose un système de tampon permettant de mettre à jour plus efficacement les données statistiques des mailles. De plus, cette méthode permet de réduire la consommation mémoire par rapport à l'étude précédente, le nombre de serveurs de donnée étant égal au nombre de processus. La répartition de la charge de calcul et des données de cette approche est donc de meilleure qualité qu'avec notre approche dynamique. Cependant, le volume de communication et la contention sur les données demeurent des sérieuses limitations de cette méthode. L'approche dynamique que nous avons proposée a l'avantage d'avoir un coût en communication limité.

Discussion sur la complémentarité avec notre approche

Le coût en communication des approches de décomposition de donnée que nous venons d'analyser en est une sérieuse limitation. Néanmoins, ces approches offrent une alternative intéressante. En effet, les études réalisées ont proposé des techniques permettant de minimiser le coût d'une

lecture distante et le coût d'une écriture distante. Notre approche consiste à permettre une décomposition de domaine sans déséquilibres de charge significatifs, tout en offrant un coût de communication réduit. Elle s'appuie sur le fait que n'importe quelle maille peut être créé à la volée. Cette création n'est possible que si la ressource a accès aux données de géométrie et aux sections efficaces. Ainsi, l'utilisation des mécanismes optimisés de lecture distante, proposée par les méthodes de décomposition de donnée, nous permettrait de réaliser une décomposition de domaine complète avec un surcoût minimal. Le couplage de notre approche avec l'approche de décomposition de donnée possède alors des perspectives intéressantes.

7.4 Conclusion

La problématique de régulation dynamique de charge et de donnée du transport de particules Monte-Carlo a fait l'objet de plusieurs études. L'approche que nous avons proposée cible des architectures massivement parallèles et homogènes. Nous avons également choisi le paradigme de programmation parallèle *MPI+OpenMP* parce que celui-ci est parmi les mieux adaptés pour exploiter de telles architectures, mais aussi parce que le code de simulation d'où est extrait notre prototype utilise ce paradigme. L'intégration de notre méthode dans ce code de simulation n'impliquera pas de changement structurel trop conséquent.

Dans la littérature, des études se sont intéressées aux problématiques de régulation de charge et de donnée. Alors que les outils de régulation dynamique permettent, pour un large panel d'applications, de garantir un optimisation suffisante de la répartition de la charge de calcul, les applications de transport de particules Monte-Carlo nécessitent des méthodes de régulation dynamique plus spécifiques.

L'optimisation de la méthode hybride, en permettant l'attribution dynamique des sous-domaines aux processus, permet une amélioration significative de la distribution des charges de calcul. Cette optimisation est cependant assez coûteuse en terme de communication. De plus, cette méthode nécessite une vision globale de la répartition des particules sur le domaine, impliquant donc d'importants mécanismes de synchronisation.

La décomposition de donnée permet à la fois une limitation de la consommation mémoire et une répartition de la charge de calcul proche de l'optimal. En revanche, le volume de communication nécessaire aux lectures et écritures distantes constitue une sérieuse limitation de la méthode.

L'approche dynamique que nous proposons permet de minimiser le déséquilibre de charge de calcul et de limiter la consommation mémoire. De plus, les algorithmes de régulation dynamique que nous avons mis en œuvre ne nécessitent pas une vision globale de la répartition des particules et le volume de communication nécessaire à leur fonctionnement est réduit. Pour conclure, nous pensons que le couplage de notre approche avec l'approche de décomposition de donnée pourrait faire l'objet de travaux futurs.

Chapitre 8

Conclusion

Les applications de transport de particules Monte-Carlo [1][2] consistent à suivre les histoires de particules dans un domaine de simulation. Les particules se déplacent et subissent des interactions. La répartition des particules est inégale sur le domaine de simulation. De plus, celle-ci évolue dynamiquement au cours de la simulation. La parallélisation des applications de transport de particules Monte-Carlo soulève des problématiques de répartition de charge de calcul et de donnée. Les méthodes classiques de parallélisation obtiennent des répartitions de charge et de donnée satisfaisantes pour des applications régulières ou faiblement irrégulières. Ces méthodes sont très impactées par des applications trop irrégulières, telles que le transport de particules Monte-Carlo.

Les applications de transport de particules Monte-Carlo nécessitent des mécanismes pour bien répartir la charge de calcul et les données de simulation. Ces mécanismes doivent garantir une bonne répartition de la charge et des données, avec un coût de régulation modéré, pour permettre le passage à l'échelle des applications de transport Monte-Carlo. Nous avons alors posé la problématique suivante :

Comment garantir une bonne répartition dynamique des charges de calcul et des données, sur des plateformes massivement parallèles, en minimisant le coût de communication ?

Nous avons proposé plusieurs approches permettant d'apporter des solutions à cette problématique. Nous avons ciblé les architectures homogènes massivement parallèles de type supercalculateur multiprocesseurs. Nous commencerons par résumer les principales contributions de la thèse en sous-section 8.1, puis nous discuterons des travaux futurs en sous-section 8.2 et enfin nous terminerons par des propositions de travaux sur le plus long terme en sous-section 8.3.

8.1 Contributions

L'utilisation des méthodes classiques de parallélisation ne permettent pas d'atteindre les objectifs de régulation de charge et de donnée dans toutes les configurations. Pour mettre en évidence les limitations de ces méthodes, nous avons réalisé une analyse théorique et expérimentale (sous-section 8.1.1). Pour corriger ces limitations, nous avons proposé une approche de partitionnement de graphe (sous-section 8.1.2). Enfin, en nous appuyant sur des analyses de la méthode par partitionnement, nous avons proposé une approche dynamique (sous-section 8.1.3)

8.1.1 Analyse des méthodes classiques de parallélisation appliquées au transport Monte-Carlo

Pour répondre à la problématique de régulation de charge et de donnée des applications de transport de particules Monte-Carlo, nous avons commencé par étudier les méthodes classiques de parallélisation. Cette étude s'est décomposée en deux parties, une analyse théorique et une analyse expérimentale.

Analyse théorique

La première partie a mis en évidence des faiblesses en termes de répartition de la charge et des données des méthodes classiques de parallélisation. Cette étude théorique avait pour objectif d'exprimer la répartition de la charge de calcul et des données avec la meilleure configuration et avec la pire configuration. Cette étude a permis également d'exprimer les coûts de communication. Les performances parallèles d'une méthode sont impactées par la charge de calcul et la consommation mémoire de la ressource la plus surchargée. Nous avons donc exprimé la charge de calcul maximale et la consommation mémoire maximale par ressource. Nous avons également proposé une analyse lorsque le nombre de ressources tend vers l'infini. L'objectif de cette étude était d'exprimer formellement les limitations des méthodes classiques de parallélisation lorsque celles-ci sont appliquées au transport de particules Monte-Carlo. Nous avons conclu que ces méthodes ne permettaient pas de répartir à la fois les particules et les mailles dans les configurations étudiées.

Analyse expérimentale

La seconde partie a validé les conclusions de l'analyse théorique. Cette analyse expérimentale s'est décomposée en plusieurs thématiques. Nous avons commencé par analyser la répartition des particules et la consommation mémoire. Ensuite, nous avons analysé les méthodes en extensibilité forte et en extensibilité faible. Cette analyse a mise en évidence les déséquilibres de charge induits par la méthode de décomposition de domaine et, dans une moindre mesure, de la méthode hybride. Elles ont également permis d'illustrer les inconvénients qu'impliquaient la réplication intégrale du domaine en termes de consommation mémoire et de temps de traitement des mailles. Nous avons alors conclu que l'usage d'une méthode hybride pourrait être pertinent dans toutes les configurations seulement si des mécanismes de régulation de charge étaient instaurés.

8.1.2 Régulation de charge par partitionnement de graphe

Pour résoudre efficacement la problématique de régulation de charge et de donnée, notre première approche a été d'utiliser les techniques de partitionnement de graphe. Pour cela, nous avons proposé une méthode de régulation semi-dynamique de charge et de donnée par partitionnement de graphe.

Modélisation sous forme de graphe

La problématique de répartition de la charge de calcul et des données est complexe. Nous devons distribuer des millions de particules réparties aléatoirement entre des centaines de milliers de mailles. Pour faciliter le partitionnement d'un tel maillage, nous avons proposé une modélisation sous forme de graphe capable d'exprimer fidèlement la répartition de la charge de calcul et des données. Nous avons opté pour un graphe pondéré dont chaque sommet représente un groupe de mailles et dont le poids correspond au nombre de particules présentes sur les mailles qu'il

représente. Ce modèle a permis de simplifier structurellement les éléments à partitionner et de réduire considérablement leur nombre.

Partitionnement semi-dynamique de graphe

Nous avons proposé une méthode permettant de partitionner et de repartitionner ce graphe si nécessaire, avec une fréquence maximale fixée à l'avance. Pour cela, nous avons défini des critères de partitionnement fonction de la répartition des particules et des mailles. Nous avons alors mis en œuvre des mécanismes capables d'invoquer un partitionnement du graphe lorsqu'au moins un des critères de déséquilibre était satisfait.

Résultats expérimentaux

La méthode par partitionnement a permis une amélioration significative, de la répartition de la charge de calcul et des données, par rapport aux méthodes de décomposition de domaine et hybride. Bien que l'approche par partitionnement implique une légère surconsommation mémoire relative aux particules, elle a permis une réduction significative de la consommation mémoire relative aux mailles par rapport à la méthode de réplication de domaine. La méthode par partitionnement a également obtenu des performances en extensibilité forte en temps de restitution proches de celles de la méthode de réplication de domaine (environ 10 % plus lente). En extensibilité faible, en revanche, nous avons observé un écart plus important par rapport à la méthode de réplication de domaine (environ 50 % plus lente). Nous avons alors conclu que l'absence de régulation de charge entre deux partitionnements amenait des déséquilibres de charge trop importants. Nous avons également observé que l'augmentation de la fréquence du partitionnement était trop coûteuse pour être compensée. Ce constat nous a orienté vers une approche de régulation dynamique et locale.

8.1.3 Régulation dynamique de charge et de donnée

L'approche par régulation dynamique et locale consiste à ce que chaque ressource contribue localement à la redistribution des charges de calcul et des données. Nous avons proposé un modèle de graphe et des algorithmes dynamiques de régulation.

Modélisation dynamique sous forme de graphe et voisinage

Dans le but de permettre à chaque ressource d'intervenir localement dans le processus de régulation de charge et de donnée, nous avons défini un modèle dynamique de graphe. Ce graphe est soumis à des règles de construction capables d'exprimer la répartition locale des charges de calcul et des données tout en prenant en compte l'aspect itératif. À chaque ressource correspond son propre sous-graphe. Nous avons proposé une méthode de voisinage permettant à chaque ressource de sélectionner les ressources avec lesquelles elle mettra en œuvre une régulation de charge et de donnée. Pour cela, nous avons développé un algorithme itératif de sélection reposant sur une modélisation sous forme d'un graphe quotient.

Limitation du nombre de mailles

À l'aide de mécanismes locaux et dynamiques, nous avons proposé des algorithmes de régulation de charge et de donnée basés sur un modèle de graphe local. Nous avons proposé une

méthode de restriction de la consommation mémoire relatives aux mailles. Cette restriction repose sur une limitation du nombre de sommets de chaque sous-graphe. Pour cela, chaque ressource transfère des sommets sélectionnés vers les sous-graphes voisins.

Redistribution dynamique de particules

Pour garantir une bonne répartition de la charge de calcul, nous avons proposé une méthode de redistribution dynamique des particules. L'objectif de cette méthode est de permettre à chaque ressource de transférer à d'autres ressources des particules en vue de rééquilibrer la répartition de la charge. Pour cela, nous avons instauré un système de jeton dynamique et des mécanismes de transfert de particules.

Mécanismes de contrôle

Enfin, un ensemble de mécanismes de contrôle a également été proposé afin de corriger d'éventuelles erreurs. Nous avons défini des critères de validité de la solution proposée par l'approche dynamique. Nous avons développé deux algorithmes de correction. Le premier consiste à mettre à jour le voisinage des sous-graphes. Le second consiste à réaliser un partitionnement du graphe global.

Résultats expérimentaux

L'approche dynamique a obtenu des résultats probants en permettant une limitation de la consommation mémoire sans déséquilibre de charge significatif. Les performances en temps de restitution en extensibilité forte de l'approche dynamique sont meilleures que celles des autres méthodes, y compris la méthode de réplication de domaine. L'approche dynamique a également obtenu des performances en extensibilité faible pratiquement équivalentes à celles de la méthode de réplication de domaine, avec moins de 10 % d'écart en moyenne.

8.2 Travaux futurs

Dans le cadre de cette thèse, nous avons proposé des approches semi-dynamique et dynamique. Ces approches ont obtenu des résultats encourageants. Cependant, des pistes d'amélioration à court et moyen terme sont envisagées.

Repartitionnement

Nous avons fait le choix d'utiliser l'outil de partitionnement *MeTiS* plutôt que *ParMeTiS* pour réaliser les partitionnements. Nous n'avons donc pas utilisé l'algorithme de repartitionnement permettant de minimiser les migrations de donnée. Une évaluation de la méthode par partitionnement en se basant sur une technique de repartitionnement est donc envisagée. Nous pensons que l'utilisation de cette technique pourrait réduire le coût global des partitionnements. Nous émettons en revanche quelques doutes quant à la qualité des partitions.

Accès aux données géométriques et sections efficaces

L'approche dynamique que nous avons proposée permet une répartition dynamique des données de simulation. Cette approche s'appuie sur le fait que les mailles puissent être créées à

la volée. Pour des simulations complexes, les données géométriques et les sections efficaces ne peuvent pas être calculées dynamiquement. Ces données doivent alors être calculées et stockées en mémoire dès l'initialisation de la phase de transport. L'implantation actuelle de notre approche ne permet pas d'accéder dynamiquement à des données géométriques et à des sections efficaces qui auraient été distribuées en mémoire. La répartition des données géométriques entre les ressources impliquerait des communications permettant de récupérer les informations décrivant les mailles. Les travaux sur la décomposition de donnée [38][52][53][58] décrits dans le chapitre 7 donnent des pistes pour réaliser cette décomposition des données géométriques et des sections efficaces. En effet, l'utilisation de communication unidirectionnelles permettant de lire à distance les informations concernant les mailles est probablement une méthode bien adaptée à notre cas. De plus, la limitation du nombre de mailles que nous avons mise en œuvre nous garantit un coût en communications réduit. Nous pourrons également utiliser le voisinage des sous-graphes pour déterminer la localisation des données. L'utilisation du voisinage nous permettra de réduire la contention sur les données.

Interactions avec les voisins

Nous avons observé que les échanges de graphes, bien que relativement peu coûteux en termes de volume de communication, impliquaient une synchronisation. Il pourrait être envisageable de ne pas systématiquement échanger les graphes ou alors de recouvrir ces échanges par du calcul. Nous pourrions, par exemple, permettre la lecture distante des graphes par des communications unidirectionnelles non systématiques. Il faudrait également gérer, en utilisant le même principe, la lecture distante du nombre de particules des ressources voisines. Une mise à jour du nombre de particules d'une ressource distante, pour prendre en compte les transferts de particules liés à la limitation du nombre de sommets, pourrait également être étudiée.

L'algorithme de redistribution des particules se base sur un algorithme de prise de jeton dynamique. Cet algorithme présente l'avantage de s'adapter à l'évolution de la répartition de la charge de calcul. En revanche, il comporte certaines limitations. Une ressource ne peut recevoir que d'une seule autre ressource. Dans certains cas, il serait pourtant pertinent de permettre à plusieurs ressources d'envoyer des particules à une seule et même ressource. Ensuite, cet algorithme se base sur le principe du ≪premier arrivé, premier servi≫. Cette méthode permet une certaine équité entre les ressources. Cependant, lorsqu'une ressource est très surchargée, nous devons garantir que cette ressource puisse prendre suffisamment de jetons. Nous suggérons pour cela plusieurs pistes. Premièrement, un système de donation de jeton est envisagé. En effet, plutôt que les ressources cibles soient inactives nous proposons qu'elles puissent choisir quelles ressources pourront lui envoyer des particules. Deuxièmement, nous envisageons une approche de jeton multiple en lieu et place de l'approche du jeton unique. L'idée consiste à ce que chaque ressource permette le transfert de particules provenant de plusieurs sources.

8.3 Perspectives

Nous avons proposé des méthodes appliquées à un cadre spécifique. Nous nous sommes focalisés sur la phase de transport des applications de transport de particules Monte-Carlo. Nous avons choisi des approches reposant sur une modélisation des données de simulation. Nous proposons maintenant de discuter comment notre approche pourrait être étendue.

Raffinement de régulation

La régulation dynamique de charge de calcul pourrait être étendue par une technique de régulation dynamique intra-itération. Le principe serait alors de permettre du vol de tâches lorsqu'une ressource a terminé le traitement de ses particules avant ses ressources voisines. Cette technique permettrait de corriger le déséquilibre que l'algorithme de régulation de charge n'a pas réussi à éliminer. De plus, cette technique permettrait de corriger le déséquilibre de charge réel et pas seulement le déséquilibre en nombre de particules. En effet, deux ressources, ayant le même nombre de particules, peuvent avoir des temps de traitement des particules différents. Par exemple, les performances de l'ordonnancement dynamique *OpenMP* peuvent varier selon la répartition des particules sur les mailles. De plus, le temps de traitement d'une particule n'est pas le même lorsqu'une interaction se produit. Concrètement, une régulation dynamique intra-itération permettrait de redistribuer plus précisément la charge de calcul.

Adaptivité de voisinage

L'approche dynamique que nous avons proposée minimise les communications collectives et la duplication de donnée décrivant la répartition de la charge de calcul et des données entre les ressources. Alors que les algorithmes de limitation du nombre de sommets et de redistribution du nombre de particules fonctionnent localement et dynamiquement, l'algorithme de mise à jour des voisins implique une communication collective et n'est pas complètement dynamique. Une piste d'amélioration serait de rendre cet algorithme plus dynamique et d'éliminer la communication collective correspondant à l'échange des graphe quotients. Une approche dynamique de sélection de voisins amènerait une potentielle amélioration des algorithmes de régulation de charge et de donnée.

Extension à d'autres applications

Notre approche de régulation dynamique permet de résoudre la problématique de régulation de charge et de donnée du transport de particules Monte-Carlo. Cette approche pourrait également être utilisée pour d'autres applications impactées par une problématique similaire. Typiquement, il pourrait être intéressant d'adapter notre approche aux applications dont la charge de calcul par maille dépend des données physiques contenues dans chaque maille. Nous pourrions alors adapter notre méthode pour qu'elle puisse traiter des problèmes différents.

Apprentissage

La phase de transport de particules Monte-Carlo sur laquelle nous nous sommes focalisés s'inscrit dans un processus plus large. La simulation complète consiste à alterner des phases d'hydrodynamique et de transport de particules. La simulation est décomposée en pas de temps. À chaque pas de temps, la phase de transport de particules utilise le domaine de simulation qui a été mis à jour en fonction des pas de temps précédents. À chaque pas de temps, nous devons donc effectuer la régulation de charge et de donnée pendant la phase de transport de particules. Notre algorithme repose sur des paramètres statiques. L'idée serait d'adapter la valeurs des paramètres selon le cas test. Une approche par apprentissage permettrait de prendre en compte le déroulé des pas de temps précédents, dans le but d'optimiser le comportement de l'algorithme de régulation. Ce principe pourrait même être étendu d'une simulation sur l'autre.

Bibliographie

[1] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247) :335–341, 1949.

[2] I. Lux and L. Koblinger. *Monte Carlo particle transport methods : neutron and photon calculations*. CRC Press, 1991.

[3] Paul C. Martin and Julian Schwinger. Theory of many-particle systems. i. *Physical Review*, 115(6) :1342, 1959.

[4] InfiniBand Trade Association et al. Infinibandtm architecture specification volume 1 release 1.3 (general specifications), 2015.

[5] Message Passing Interface Forum. Mpi : A message passing interface standard. http ://mpi-forum.org/.

[6] Leonardo Dagum and Ramesh Menon. Openmp : an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1) :46–55, 1998.

[7] Marc Snir. *MPI : the complete reference. The MPI-2 extensions*, volume 2. Mit Press, 1998.

[8] Jack Dongarra et al. Mpi : A message-passing interface standard, version 3.0. *High Performance Computing Center Stuttgart (HLRS)*, 2013.

[9] ARB OpenMP. Openmp application program interface version 4.0, 2013.

[10] Ian Foster. Designing and building parallel programs. http ://www.mcs.anl.gov/ itf/dbpp/text/node30.html.

[11] Andrea Toselli and Olof B. Widlund. *Domain decomposition methods : algorithms and theory*, volume 34. Springer, 2005.

[12] David Dureau and Gaël Poëtte. Hybrid parallel programming models for amr neutron monte-carlo transport. In *SNA+ MC 2013-Joint International Conference on Supercomputing in Nuclear Applications+ Monte Carlo*, page 04202. EDP Sciences, 2014.

[13] Thomas A. Brunner and Patrick S. Brantley. An efficient, robust, domain-decomposition algorithm for particle monte carlo. *Journal of Computational Physics*, 228(10) :3882–3890, 2009.

[14] John C. Wagner, Scott W. Mosher, Thomas M. Evans, Douglas E. Peplow, and John A. Turner. Hybrid and parallel domain-decomposition methods development to enable monte carlo for reactor analyses. *Progress in nuclear science and technology*, 2(1) :815–820, 2011.

[15] Patrick S. Brantley and M. McKinley. Site internet du projet mercury. https ://wci.llnl.gov/simulation/computer-codes/mercury.

[16] Paul K. Romano and Benoit Forget. The openmc monte carlo particle transport code. *Annals of Nuclear Energy*, 51 :274–281, 2013.

[17] George Karypis and Vipin Kumar. A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. *University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN*, 1998.

[18] François Pellegrini and Jean Roman. Scotch : A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *International Conference on High-Performance Computing and Networking*, pages 493–498. Springer, 1996.

[19] Cédric Lachat, François Pellegrini, and Cecile Dobrzynski. PaMPA : Parallel Mesh Partitioning and Adaptation. In *DD22 22nd International Conference on Domain Decomposition Methods*, Lugano, Switzerland, September 2013. Institute of Computational Science (ICS), Università della Svizzera italiana (USI).

[20] Cédric Lachat, François Pellegrini, and Cecile Dobrzynski. Parallel mesh adaptation using parallel graph partitioning. In Eugenio Oñate, Xavier Oliver, and Antonio Huerta, editors, *5th European Conference on Computational Mechanics (ECCM V)*, volume 3 of *Minisymposia in the frame of ECCM V*, pages 2612–2623, Barcelone, Spain, July 2014. IACM & ECCOMAS, CIMNE International Center for Numerical Methods in Engineering. ISBN 978-84-942844-7-2.

[21] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory with applications*, volume 290. Macmillan London, 1976.

[22] Thang Nguyen Bui and Curt Jones. Finding good approximate vertex and edge partitions is np-hard. *Information Processing Letters*, 42(3) :153–159, 1992.

[23] George Karypis and Vipin Kumar. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 48(1) :96–129, 1998.

[24] George Karypis and Vipin Kumar. A coarse-grain parallel formulation of multilevel k-way graph partitioning algorithm. In *PPSC*, 1997.

[25] Peter Sanders and Christian Schulz. Engineering multilevel graph partitioning algorithms. In *ESA*, volume 6942, pages 469–480. Springer, 2011.

[26] George Karypis and Vipin Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48(1) :71–95, 1998.

[27] George Karypis and Vipin Kumar. Parmetis–parallel graph partitioning and field–reducing matrix ordering. *URL http ://glaros. dtc. umn. edu/gkhome/metis/parmetis/overview*, 2013.

[28] Cédric Chevalier and François Pellegrini. Pt-scotch : A tool for efficient parallel graph ordering. *Parallel computing*, 34(6) :318–331, 2008.

[29] George Karypis and Vipin Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, pages 1–13. IEEE Computer Society, 1998.

[30] George Karypis. Multilevel algorithms for multi-constraint hypergraph partitioning. Technical report, DTIC Document, 1999.

[31] Kirk Schloegel, George Karypis, and Vipin Kumar. Parallel multilevel algorithms for multi-constraint graph partitioning. In *Euro-Par 2000 Parallel Processing*, pages 296–310. Springer, 2000.

[32] Kirk Schloegel, George Karypis, and Vipin Kumar. Parallel static and dynamic multi-constraint graph partitioning. *Concurrency and Computation : Practice and Experience*, 14(3) :219–240, 2002.

[33] Roy D. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency : Practice and experience*, 3(5) :457–481, 1991.

[34] Chris Walshaw, Mark Cross, and Martin G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 47(2) :102–108, 1997.

[35] Kirk Schloegel, George Karypis, and Vipin Kumar. Multilevel diffusion schemes for repartitioning of adaptive meshes. *Journal of Parallel and Distributed Computing*, 47(2) :109–124, 1997.

[36] Leonid Oliker and Rupak Biswas. Plum : Parallel load balancing for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 52(2) :150–177, 1998.

[37] Umit V. Catalyurek, Erik G. Boman, Karen D. Devine, Doruk Bozdağ, Robert T Heaphy, and Lee Ann Riesen. A repartitioning hypergraph model for dynamic load balancing. *Journal of Parallel and Distributed Computing*, 69(8) :711–724, 2009.

[38] Paul K. Romano, Andrew R. Siegel, Benoît Forget, and Kord Smith. Data decomposition of monte carlo particle transport simulations via tally servers. *Journal of Computational Physics*, 252 :20–36, 2013.

[39] BSC (Barcelona Supercomputing Center). Dlb (dynamic load balancing). https ://pm.bsc.es/dlb.

[40] Laxmikant V. Kalé and Sanjeev Krishnan. Charm++ : A portable concurrent object oriented system based on c++. In *Proceedings of the Eighth Annual Conference on Object-oriented Programming Systems, Languages, and Applications*, OOPSLA '93, pages 91–108, New York, NY, USA, 1993. ACM.

[41] Bilge Acun, Abhishek Gupta, Nikhil Jain, Akhil Langer, Harshitha Menon, Eric Mikida, Xiang Ni, Michael Robson, Yanhua Sun, Ehsan Totoni, et al. Parallel programming with migratable objects : Charm++ in practice. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 647–658. IEEE Press, 2014.

[42] Chao Huang, Orion Lawlor, and Laxmikant V. Kalé. Adaptive mpi. In *International workshop on languages and compilers for parallel computing*, pages 306–322. Springer, 2003.

[43] Milind Bhandarkar, L. Kalé, Eric de Sturler, and Jay Hoeflinger. Adaptive load balancing for mpi programs. *Computational Science-ICCS 2001*, pages 108–117, 2001.

[44] Gengbin Zheng, Esteban Meneses, Abhinav Bhatele, and Laxmikant V. Kalé. Hierarchical load balancing for charm++ applications on large supercomputers. In *Parallel Processing Workshops (ICPPW)*, pages 436–444. IEEE, 2010.

[45] Laxmikant V. Kalé. The virtualization approach to parallel programming : Runtime optimizations and the state of the art. In *Los Alamos Computer Science Institute Symposium-LACSI*, 2002.

[46] Laxmikant V. Kalé and Gengbin Zheng. Charm++ and ampi : Adaptive runtime strategies via migratable objects. *Advanced Computational Infrastructures for Parallel and Distributed Applications*, pages 265–282, 2009.

[47] R. Procassini, M. O'Brien, and J. Taylor. Load balancing of parallel monte carlo transport calculations. *Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications, Palais des Papes, Avignon, Fra*, 2005.

[48] Hiroshi Nakashima, Yohei Miyake, Hideyuki Usui, and Yoshiharu Omura. Ohhelp : a scalable domain-decomposing dynamic load balancing for particle-in-cell simulations. In *Proceedings of the 23rd international conference on Supercomputing*, pages 90–99. ACM, 2009.

[49] M. J. O'Brien and P. S. Brantley. *Particle Communication and Domain Neighbor Coupling : Scalable Domain Decomposed Algorithms for Monte Carlo Particle Transport*. Jan 2015.

[50] Matthew J. O'Brien, Patrick S. Brantley, and Ken I. Joy. Scalable load balancing for massively parallel distributed monte carlo particle transport. In *Proceedings of International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C 2013), Sun Valley, Idaho*, 2013.

[51] Henry E. Daniels. Round-robin tournament scores. *Biometrika*, 56(2) :295–299, 1969.

[52] P. K. Romano, B. Forget, and F. Brown. Towards scalable parallelism in monte carlo transport codes using remote memory access. *Prog.Nucl.Sci.Technol.*, 2 :670–675, 2011.

[53] Paul K. Romano, Benoit Forget, Kord Smith, and Andrew Siegel. On the use of tally servers in monte carlo simulations of light-water reactors. In *SNA+ MC 2013-Joint International Conference on Supercomputing in Nuclear Applications+ Monte Carlo*, page 04301. EDP Sciences, 2014.

[54] Global view resilience project. http ://gvr.cs.uchicago.edu.

[55] A. Chien, Pavan Balaji, P. Beckman, Nan Dun, Aiman Fang, Hajime Fujita, Kamil Iskra, Z. Rubenstein, Ziming Zheng, Rob Schreiber, et al. Versioned distributed arrays for resilience in scientific applications : Global view resilience. *Procedia Computer Science*, 51 :29–38, 2015.

[56] Katherine Yelick, Dan Bonachea, Wei-Yu Chen, Phillip Colella, Kaushik Datta, Jason Duell, Susan L. Graham, Paul Hargrove, Paul Hilfinger, Parry Husbands, et al. Productivity and performance using partitioned global address space languages. In *Proceedings of the 2007 international workshop on Parallel symbolic computation*, pages 24–32. ACM, 2007.

[57] Nan Dun, Hajime Fujita, John R. Tramm, Andrew A. Chien, and Andrew R. Siegel. Data decomposition in monte carlo neutron transport simulations using global view arrays. *International Journal of High Performance Computing Applications*, 29(3) :348–365, 2015.

[58] Paul Kollath Romano. *Parallel algorithms for Monte Carlo particle transport simulation on exascale computing architectures*. PhD thesis, Massachusetts Institute of Technology, 2013.

[59] Judith F. Briesmeister et al. Mcnptm-a general monte carlo n-particle transport code. *Version 4C, LA-13709-M, Los Alamos National Laboratory*, page 2, 2000.

[60] Umit V. Catalyurek, Erik G. Boman, Karen D. Devine, Doruk Bozdag, Robert Heaphy, and Lee Ann Riesen. Hypergraph-based dynamic load balancing for adaptive scientific computations. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–11. IEEE, 2007.

[61] Karen D. Devine, Erik G. Boman, Robert T. Heaphy, Bruce A. Hendrickson, James D. Teresco, Jamal Faik, Joseph E. Flaherty, and Luis G. Gervasio. New challenges in dynamic load balancing. *Applied Numerical Mathematics*, 52(2-3) :133–152, 2005.

[62] Matthew Joseph O'Brien. *Scalable Domain Decomposed Monte Carlo Particle Transport*. PhD thesis, 2014.

[63] Bongki Moon and Joel Saltz. Adaptive runtime support for direct simulation monte carlo methods on distributed memory architectures. In *Proceedings of the scalable High-Performance Computing Conference, 1994*, pages 176–183. IEEE, 1994.

Résumé

Les applications de transport de particules Monte-Carlo consistent à étudier le comportement de particules se déplaçant dans un domaine de simulation. La répartition des particules sur le domaine de simulation n'est pas uniforme et évolue dynamiquement au cours de la simulation. La parallélisation de ce type d'applications sur des architectures massivement parallèles amène à résoudre une problématique complexe de répartition de charge de calcul et de donnée sur un grand nombre de cœurs de calcul.

Nous avons d'abord identifié les difficultés de parallélisation des applications de transport de particules Monte-Carlo à l'aide d'analyses théoriques et expérimentales des méthodes de parallélisation de référence. Une approche semi-dynamique reposant sur des techniques de partitionnement a ensuite été proposée. Enfin, nous avons défini une approche dynamique capable de redistribuer la charge de calcul et les données tout en maintenant un faible volume de communication. L'approche dynamique a obtenu des accélérations en extensibilité forte et une forte réduction de la consommation mémoire par rapport à une méthode de réplication de domaine parfaitement équilibrée.

Abstract

Monte Carlo particle transport applications consist in studying the behaviour of particles moving about a simulation domain. Particles distribution among simulation domains is not uniform and change dynamically during simulation. The parallelization of this kind of applications on massively parallel architectures leads to solve a complex issue of workloads and data balancing among numerous compute cores.

We started by identifying parallelization pitfalls of Monte Carlo particle transport applications using theoretical and experimental analysis of reference parallelization methods. A semi-dynamic based on partitioning techniques has been proposed then. Finally, we defined a dynamic approach able to redistribute workloads and data keeping a low communication volume. The dynamic approach obtains speedups using strong scaling and a memory footprint reduction compared to the perfectly balanced domain replication method.