

Partitionnement de maillage pour solveurs numériques parallélisés et distribués en tâches

Alice Lasserre

Centre Inria de l'université de Bordeaux,
Équipe Storm - 200 Av. de la Vieille Tour
33405 Talence - France
alice.lasserre@inria.fr

Résumé

Les simulations numériques constituent un moyen précieux pour étudier la mécanique des fluides en modélisant des phénomènes complexes avec une grande précision. Nous nous intéressons aux solveurs explicites « volumes finis » sur des maillages non structurés. Les implémentations parallèles de ces solveurs reposent de plus en plus sur des modèles à base de tâches afin de maximiser l'asynchronisme à l'intérieur des nœuds de calcul et faciliter l'équilibrage de charge. Toutefois, l'exécution parfaitement équilibrée de tels codes reste un défi sur les cas industriels de production. Dans cet article, nous analysons l'origine de périodes d'inactivité observées lors de l'analyse de traces d'exécution et proposons une nouvelle stratégie de partitionnement de maillage dirigée par la classe temporelle des mailles, menant à la génération de graphes de tâches propices à une exécution équilibrée entre les noeuds. Nous validons notre solution dans le contexte d'un code de production développé et utilisé par ArianeGroup et Airbus. Une implémentation au sein d'un émulateur expose une réduction du temps d'exécution allant jusqu'à 50% pour des maillages de production allant de quelques milliers à 12 millions de composantes. Ces résultats sont confirmés directement au sein de l'application industriel avec un gain final de 20%.

Mots-clés : maillage, hpc, cfd, équilibrage, distribué

1. Introduction

Le bruit généré par les hélices d'avions, la séparation des étages de lanceurs de fusée ou la propagation d'ondes de choc lors de décollage sont des exemples de phénomènes physiques qui présentent une analyse difficile avec les seules méthodes analytiques ou expérimentales. Des simulations numériques à grande échelle ont donc été implémentées afin de modéliser ces phénomènes qui, s'appuyant sur des maillages non-structurés, requièrent l'utilisation d'architectures parallèles pour atteindre des temps d'exécution raisonnables. Classiquement, ces codes de simulation font progresser le temps par unités discrètes (i.e. les pas de temps) en actualisant les valeurs physiques, telles que la pression ou la température, mémorisées au sein de chaque cellule. Pour un solveur explicite, le pas de temps maximum admissible pour une cellule dépend principalement de son volume, qui varie fortement d'une cellule à l'autre dans des maillages non-homogènes. Afin de ne pas aligner tous les calculs sur le pas de temps admissible le plus faible, un solveur temporel *adaptatif* utilise des pas de temps différenciés en fonction de la *classe temporelle* des cellules. Chaque itération est ainsi découpée en plusieurs sous-itérations durant lesquelles seul un sous-ensemble des cellules est mis à jour. Ainsi, l'implémentation parallèle doit gérer un fort

déséquilibre de charge entre différentes zones du maillage. Bien qu'un parallélisme à base de tâches offre une plus grande souplesse dans la répartition de la charge de travail en évitant des synchronisations inutiles, il demeure difficile d'occuper toutes les unités de calcul tout au long de l'exécution. Dans cet article, nous nous intéressons à ce problème dans le contexte du code de production FLUSEPA développé et utilisé par ArianeGroup et Airbus. Nous présentons FLUSIM, un émulateur python qui reproduit fidèlement le comportement parallèle de la boucle d'itération principale du solveur aérodynamique. Dans un premier temps, nous examinons l'ordonnancement des tâches soumises par l'application afin de déterminer d'éventuelles anomalies corrélées. Nous démontrons que le problème n'est pas lié à un mauvais ordonnancement des tâches et qu'il est intrinsèque à la structure du graphe lui-même. Cette structure résultant directement de la décomposition générée par l'étape de partitionnement du maillage, nous proposons une nouvelle approche de partitionnement en vue d'obtenir un graphe de tâches doté d'une structure plus équilibrée. Cette approche s'appuie essentiellement sur la distinction des cellules du maillage en fonction de leur niveau temporel, c'est-à-dire de leur contribution dans chaque sous-itération, et repose sur des outils de partitionnement de graphe existants. Nous présentons les résultats préliminaires de nos expériences menées sur un ensemble de maillages de production représentatifs, qui laissent entrevoir une accélération du temps d'exécution total d'un facteur deux au sein d'un émulateur et ainsi qu'un gain d'environ 20% directement au sein de FLUSEPA.

2. Simulation appliquée à la mécanique des fluides

Les codes industriels de simulation reproduisant des phénomènes fluides complexes travaillent sur des maillages de très grandes tailles et doivent nécessairement être parallélisés afin de fournir des résultats en un temps acceptable. Dans cet article, nous nous focalisons sur le code de simulation FLUSEPA [9] [8], utilisé chez Airbus, qui est un solveur de Navier-Stokes parallèle dont le fonctionnement est schématisé en figure 3. Le code utilise une méthode explicite sur des volumes finis aboutissant à la génération d'un maillage 3D, qui est ensuite partitionné en un ensemble de domaines afin que les calculs associés puissent être répartis entre les différents noeuds de la machine. L'implémentation actuelle de FLUSEPA repose sur un modèle d'exécution à base de tâches [7] dont l'ordonnancement est délégué au support d'exécution StarPU [2] (essentiellement à destination de CPUs) et les communications entre noeuds à la bibliothèque MPI [13]. La granularité du parallélisme est contrôlée en ajustant le nombre de domaines d'entrée du partitionnement, puisque chacun d'entre eux engendre quelques tâches spécifiques afin d'effectuer les calculs de ces cellules et faces.

L'instabilité des phénomènes étudiés nécessite l'utilisation de maillages dont la résolution n'est pas uniforme. Le maillage d'un pas de tir Ariane 5 (illustré en figure 1) se caractérise par une densité élevée de mailles très fines au niveau du lanceur. La dimension des mailles augmente graduellement à mesure qu'on s'éloigne de ce dernier et leur densité s'amenuise. Le pas de temps maximal admissible des mailles fines est bien plus faible que celui des mailles grossières en bordure de phénomène. Afin de ne pas pénaliser la simulation en calculant l'ensemble des mailles à la fréquence maximale requise par les petites mailles, le solveur implémente un schéma temporel adaptatif. Les mailles se voient attribuer un niveau temporel τ qui reflète leur pas de temps maximal admissible. L'échelle des pas de temps est exponentielle : la durée double d'un niveau à l'autre. Les itéra-

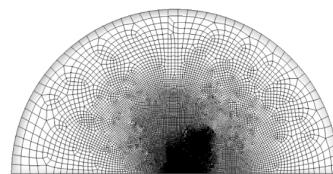


FIGURE 1 – Maillage hétérogène.

tions sont divisées en plusieurs sous-itérations, durant lesquelles seuls certains niveaux temporels sont impliqués.

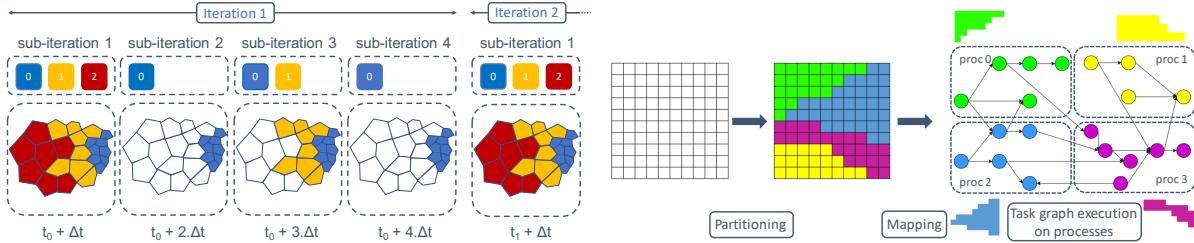


FIGURE 2 – Méthode d'intégration temporelle explicite appliquée à un maillage contenant 3 niveaux temporels. Seules certaines catégories de cellules sont calculées à chaque sous-itération.

FIGURE 3 – Schématisation du processus de simulation de FLUSEPA.

Partitioning

Mapping

Task graph execution on processes

Proc 0 Proc 1 Proc 2 Proc 3

La figure 2 illustre le fonctionnement de ce schéma d'intégration temporelle sur un maillage ayant trois niveaux temporels distincts, résultant en une division de chaque itération en 2^2 sous-itérations. Une maille de niveau $\tau = 0$ génère des calculs à chaque sous-itération (pas de temps Δ_t) tandis que les mailles $\tau = 1$ sont calculées une itération sur deux. Enfin, les mailles $\tau = 2$ ne sont calculées qu'une fois, lors de la première sous-itération. Ceci induit donc une hétérogénéité du coût des mailles en fonction de leur niveau temporel : chaque maille est caractérisée par son coût opératoire ($2^{\max-\tau}$), où \max représente le niveau temporel maximum au sein du maillage. Ainsi, chaque sous-itération injecte un volume de calcul différent durant l'exécution de la simulation. Ce dernier dépend de la quantité de maille de chaque classe de niveau temporel. Afin d'équilibrer la charge de calcul entre les domaines affectés aux processus au cours d'une itération, FLUSEPA effectue un partitionnement du maillage en s'appuyant sur une pondération des mailles en fonction de leur coût opératoire. Les partitions ainsi obtenues garantissent une charge de travail totale équilibrée entre elles. Ainsi, une partition peut contenir seulement quelques cellules de niveau temporel faible (donc coûteuses), ou au contraire contenir énormément de mailles de niveau temporel élevé (donc peu coûteuses), pourvu que les quantités de travail totales soient équivalentes. Le schéma général de l'application est fourni en figure 3. Partant du maillage initial, le partitionneur est invoqué pour former des domaines qui seront distribués sur les différents processus impliqués. L'algorithme du solveur parcourt ensuite les domaines produits afin de générer le graphe de tâche exprimant les calculs et leurs dépendances.

3. À la recherche d'un meilleur ordonnancement des tâches

Une exécution de FLUSEPA peut s'étendre sur plusieurs jours lorsque les maillages de production passés en paramètre atteignent quelques millions d'éléments, complexifiant la possibilité de tester un vaste ensemble de paramètres et nouvelles implémentations. C'est pourquoi Airbus a développé un émulateur FLUSIM qui reproduit le fonctionnement général du simulateur FLUSEPA. Le comportement de l'ordonnanceur de tâches de StarPU est reproduit sur une machine distribuée purement virtuelle. L'émulateur utilise les données de profilage produites par une exécution de calibration FLUSEPA afin d'estimer la durée de chaque tâche. De plus,

seule la bibliothèque Metis [12] liée au partitionnement est directement appelée, offrant ainsi des exécutions de quelques heures à peine sur une architecture paramétrée par l'utilisateur. La pertinence du simulateur FLUSIM est illustrée figure 6 où une comparaison de deux exécutions paramétrées de manière identique utilisant le même maillage d'entrée IND12M est effectuée, l'une provenant du solveur FLUSEPA (diagramme supérieur) et l'autre de l'émulateur FLUSIM (diagramme inférieur). IND12M est partitionné en 12 domaines en utilisant la stratégie des coûts opératoires et le graphe en résultant est exécuté sur 6 processus MPI de 4 cœurs chacun. Nous pouvons observer que FLUSIM reproduit une exécution similaire à celle de FLUSEPA en matière d'ordonnancement des tâches (avec une erreur de 30% sur la durée de l'itération). La figure 5 présente une trace d'exécution de FLUSIM, résultant du maillage CYLINDER où chaque processus MPI se voit attribuer un seul domaine dont les tâches sont exécutées sur un nombre infini de cœurs de calcul. Au sein de la trace d'exécution, les ressources actives d'un même processus sont agrégées sur une seule ressource composite. Ainsi, toute période d'inactivité émane d'un problème d'attente inhérent aux questions de distribution de données ainsi que de partitionnement. En effet, puisque chaque processus dispose d'un nombre de ressources illimité, la stratégie d'ordonnancement interne aux processus n'a pas d'influence ici.

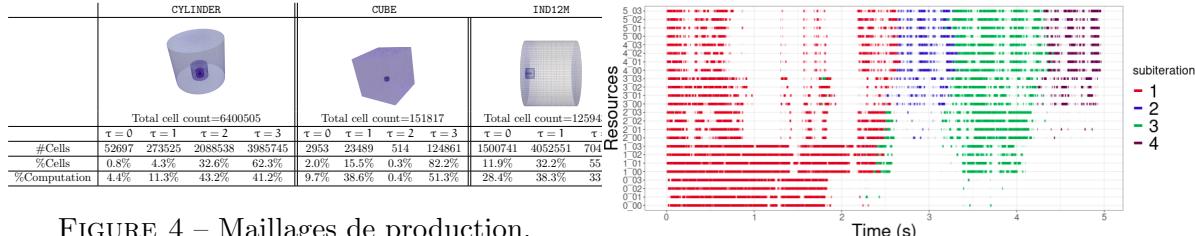


FIGURE 4 – Maillages de production.

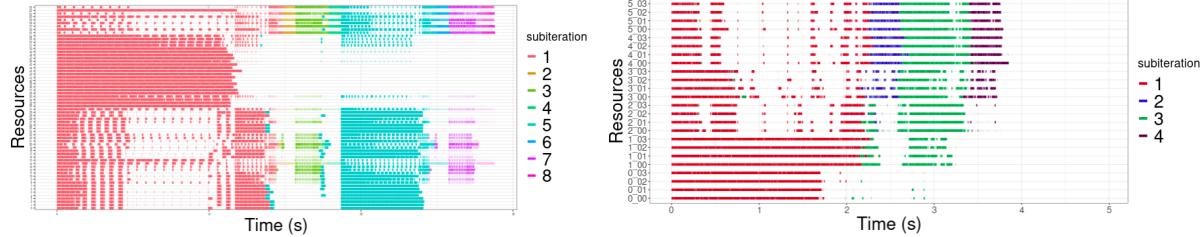


FIGURE 5 – Trace d'exécution : 64 proc. MPI (1 domaine par proc.), nombre de ressources infini par proc. MPI.

FIGURE 6 – Comparaison (paramétrage identique) des résultats de FLUSEPA (trace du haut) et de FLUSIM (trace du bas).

Bien que la stratégie de partitionnement actuelle permette d'obtenir des domaines – et donc des processus MPI – ayant une charge de travail équilibrée entre eux, l'exécution des calculs n'est pas optimale puisque des périodes d'inactivité apparaissent sur les processeurs. En effet, certains domaines génèrent du travail en quantité importante à certaines périodes d'une itération, mais en génèrent beaucoup moins à d'autres. Ces périodes correspondent aux sous-itérations de calcul définies par la méthode d'intégration temporelle adaptative. La métrique de partitionnement consistant à équilibrer les coûts opératoires ne considère pas le moment de l'exécution des tâches au sein de l'itération, c'est-à-dire la sous-itération d'insertion des calculs. La charge de travail se voit donc équilibrée pour une itération entière, et non pour chaque sous-itération. Les décompositions obtenues à partir de cette stratégie se caractérisent par des partitions pouvant contenir une quantité importante de cellules de niveau temporel élevé et d'autres contenant une quantité très faible de cellules de niveau temporel faible. En d'autres termes, les partitions sont équilibrées en ce qui concerne la charge de travail, mais inégales en termes de distribution des niveaux temporels entre les domaines. Lorsque le solveur va itérer sur les sous-itérations pour générer les tâches nécessaires, certaines partitions ne vont produire aucune insertion de tâches puisqu'elles

ne possèdent pas les niveaux temporels nécessitant d'être calculés à ce moment de la simulation. L'ordre des calculs étant strict (ceux de la sous-itération n°2 dépendent principalement de la n°1 et ainsi de suite), les processus associés à ces partitions seront inactifs durant les sous-itérations correspondantes. Ainsi, améliorer le comportement de notre application semble exiger que *la charge de travail produite à chaque sous-itération* soit répartie uniformément sur les ressources disponibles. L'idée est donc que quelque soit le niveau temporel maximal de la sous-itération, la quantité de calcul générée soit équitablement répartie entre les processus. La complexité réside ici dans la garantie que chaque processus se voit attribuer équitablement les cellules actives de la première sous-itération *et* celles de la seconde, ainsi de suite. Dans un contexte plus générale, l'objectif du partitionnement est d'équilibrer la charge de travail pour chaque sous-itération. Une approche naïve consisterait à effectuer un partitionnement distinct pour chaque sous-itération. Cela entraînerait cependant une augmentation considérable du volume de communication en raison de la redistribution nécessaire des données entre deux sous-itérations successives. Une façon plus raisonnable de résoudre le problème consiste à envisager une stratégie de partitionnement qui équilibre la charge de travail pour toutes les sous-itérations à la fois. Garantir que chaque processus traite une quantité égale de travail entre eux pour *chacune des sous-itérations* se traduit ici par une répartition uniforme des cellules *de chaque niveau temporel présent dans le maillage d'entrée* entre chaque domaine. Autrement dit, quelque soit le niveau temporel maximal d'une sous-itération donnée, il s'agit de garantir que la quantité de travail extraite du maillage soit équitablement répartie entre domaines et donc entre processus par le biais d'une modification de la métrique de partitionnement, conduisant ainsi à la génération d'une exécution efficace et bien équilibrée. En section suivante, nous présentons donc une nouvelle stratégie de partitionnement qui résulte en un ensemble de domaines contenant une quantité équivalente de chaque niveau temporel afin de garantir une distribution de charge équilibrée dans le temps.

4. Partitionnement multi-critères pour équilibrer les niveaux temporels

À l'origine, chaque cellule est caractérisée par une estimation de son coût opérationnel, et la stratégie de partitionnement initialement utilisée équilibre ce critère unique entre les différents domaines. L'équilibrage de la quantité de cellules entre les domaines de chaque niveau temporel présent au sein du maillage exige donc la mise en œuvre d'un partitionnement multi-critères. Ainsi, la nouvelle implémentation est composée des étapes suivantes. Chaque cellule est représentée par un vecteur de contraintes booléennes. La taille de ce vecteur correspond au nombre de niveaux temporels distincts dans le maillage, et la valeur est fixée à 1 à l'indice correspondant au niveau temporel de la cellule. Par exemple, le vecteur d'une cellule $\tau = 2$ au sein d'un maillage de $\max = 3$ prendra la forme suivante : [0, 0, 1, 0]. Par la suite, nous faisons

appel à l'outil *gpmetis* de la bibliothèque Metis afin de répartir équitablement les cellules selon chacun des critères, homogénéisant ainsi la répartition des niveaux temporels entre les domaines. L'exécution incluant le partitionneur de graphe multi-contraintes dévoile en figure 8 un comporte-

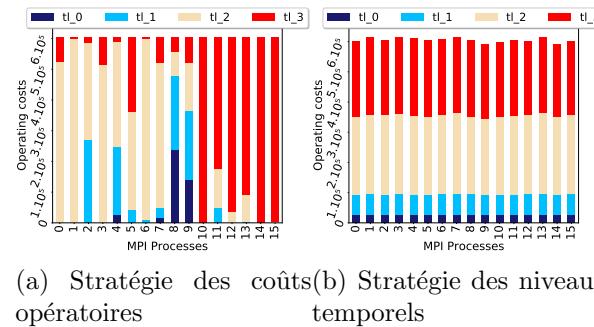


FIGURE 7 – Répartition des coûts opératoires totaux (8) générés par CYLINDER entre les processus MPI concernés.

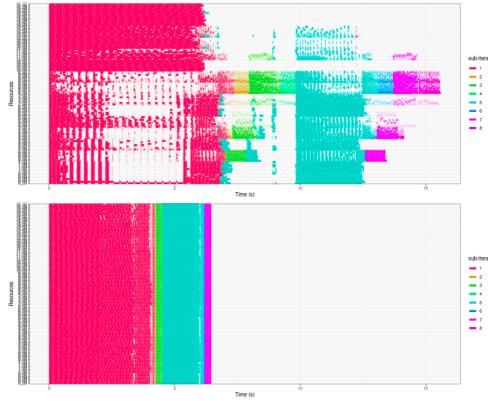


FIGURE 8 – Exécutions obtenues en appliquant la stratégie des coûts opératoires (trace de haut) et celle des niveaux temporels (trace du bas). CYLINDER est partitionné en 128 domaines dont les calculs sont émulés avec FLUSIM sur 16 proc. MPI virtuels (32 cœurs par proc.).

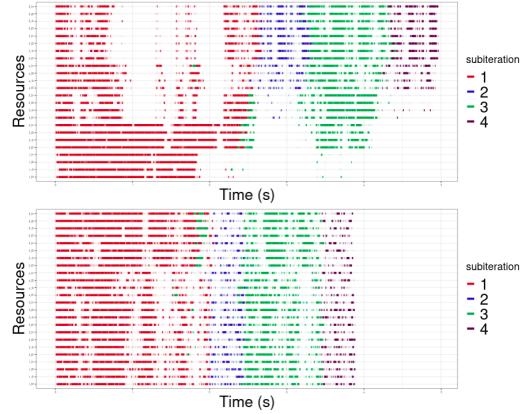


FIGURE 9 – Les résultats extraits de FLUSEPA démontrent un gain de 20% dans le code de production, utilisant la même configuration et CYLINDER est partitionné en 128 domaines dont le même maillage que ceux illustrés dans la figure 6.

ment encourageant qui induit un facteur 2 d'accélération. Il est manifeste que cette amélioration significative est fortement corrélée avec la disparition presque totale (diagramme temporel du bas) des périodes d'inactivité. Les figures 7a et 7b révèlent la distribution des coûts opératoires générés par le maillage, pour une itération, entre les processus impliqués, et ce, suivant le niveau temporel des cellules qui leur sont assignées. Dans le contexte de la stratégie mono-critère (figure 7a) d'équilibrage des coûts opératoires seuls, chaque processus possède effectivement une charge équivalente. En revanche, leur composition en niveaux temporels est très disparate : certains se voient attribuer une majorité de niveau temporel 2 et 3 tandis que d'autres détiennent la presque totalité des mailles de niveau temporel 0, 1. Le contraste est donc marquant par rapport à une stratégie multi-critères (figure 7b) où une *quantité équivalente de tous les niveaux temporels* est présente entre chaque processus (les coûts opératoires sont implicitement répartis). Par rapport à la stratégie initiale, l'équilibrage au niveau temporel induit, pour la même charge de travail produite par le maillage, la génération d'un plus grand nombre de tâches. Appliqué à un solveur numérique parallèle avec des calculs distribués, cela se traduit par une augmentation potentielle des communications entre processus MPI. D'après une première estimation, les communications augmentent de manière significative et croissent avec le nombre de partitions et donc le nombre de tâches.

Afin de valider l'approche décrite dans cet article, les niveaux temporels sont équilibrés entre les processus MPI dans le cas d'une *exécution réelle du solveur FLUSEPA*. En d'autres termes, cela se réfère au code de production lui-même, avec tous les surcoûts et les communications associés. Dans cette situation, le maillage particulier IND12M, qui est composé de plus de 12 millions de cellules et est affiché dans la colonne de droite du tableau 4, est utilisé en tant que paramètre. Ainsi, la figure 9 illustre clairement le fait que cette approche permet de réduire le temps d'exécution du solveur industriel FLUSEPA d'environ 20%.

5. Travaux connexes

Dans cet article, nous formulons une approche multi-contraintes et avons choisi pour cette étude le partitionneur de graphe Metis, ce dernier permettant de définir un vecteur de contraintes influant directement sur le partitionnement initial de l'algorithme multi-niveaux [11, 15]. Une approche multi-critère peut être aussi directement obtenue à l'aide de partitionneur qui propose des approches géométriques tel que Zoltan [10], s'appuyant directement sur les coordonnées physiques des données du maillage sans prendre en compte la connectivité des éléments. Étendre les partitionneurs mono-critère de manière à intégrer des contraintes supplémentaires constitue une autre approche. Certains travaux proposent de nouvelles implémentations des différentes étapes de l'algorithme multi-niveaux (contraction, partitionnement initial et raffinement) intégré au sein de la bibliothèque de partitionnement de graphe Scotch [14]. Ceux d'A. Casadei [6] se traduisent par l'obtention d'un partitionnement hiérarchique équilibrant la charge de travail entre les processus puis au sein des cœurs de calcul tout en minimisant les coupes d'arêtes. Les travaux de R. Barat [4, 5, 3] se concentrent sur l'obtention de partitions respectant strictement les tolérances d'équilibre afin d'obtenir la solution la plus efficace, et ce, en minimisant les communications.

Enfin, le travail le plus proche de l'approche proposée dans cet article est présenté dans [16]. Les auteurs présentent des stratégies de partitionnement de maillage directement appliquées à des simulations numériques multi-phases dont le comportement décrit est similaire à celui présenté dans l'article. En effet, ces simulations présentent plusieurs phases, séparées par des étapes de synchronisation, qui insèrent une quantité de calcul fortement disparate entre elles. Deux approches sont proposées afin d'équilibrer les phases individuellement. La première consiste à effectuer des partitionnements indépendants pour chacune des phases et de redistribuer les données entre chaque phase, ce qui induit d'importants transferts de données. Une approche plus probante proposée dans ce même article consiste à équilibrer le travail de ces phases simultanément en un seul partitionnement. Un exemple présenté distribue une quantité égale d'éléments actifs de chaque phase entre domaines.

6. Conclusion

Les solveurs explicites utilisés dans l'aéronautique œuvrent sur des maillages non structurés de très grande taille. Les schémas de résolution s'appuient sur une intégration temporelle adaptative distinguant plusieurs classes de cellules, ayant chacune son propre pas de temps maximal admissible. L'implémentation parallèle et distribuée de ces solveurs pose des problèmes d'équilibrage de charge épineux, car la simple distribution d'un même volume de calcul à chaque processus MPI est insuffisante. Dans cet article, nous mettons en évidence que le problème est inhérent à la (mauvaise) répartition des classes de niveaux temporels entre les domaines, et nous proposons une nouvelle stratégie de partitionnement directement guidée par une distribution homogène de chaque classe de niveau temporel au sein de chaque domaine. Nous avons implanté cette stratégie sous la forme d'un partitionnement multicritère avec l'outil Metis, et nous avons évalué l'amélioration de l'exécution à l'aide de l'émulateur FLUSIM. Les résultats obtenus de multiple maillages de production révèlent que le gain en temps d'exécution peut atteindre 50%. Enfin, cette stratégie est validée par son intégration directe dans le solveur numérique FLUSEPA utilisé en production, entraînant une réduction de 20% du temps d'exécution. Nos travaux futurs s'orienteront vers des post-traitements minimisant les artefacts produits par les partitionneurs lorsqu'ils sont contraints par de nombreux critères [1], ainsi que sur l'implémentation de la chaîne de traitement globale au sein du simulateur FLUSEPA.

Bibliographie

1. Aftosmis (M.), Berger (M.) et Murman (S.). – *Applications of Space-Filling-Curves to Cartesian Methods for CFD*.
2. Augonnet (C.), Thibault (S.), Namyst (R.) et Wacrenier (P.-A.). – StarPU : a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation : Practice and Experience*, vol. 23, n2, 2011, pp. 187–198.
3. Barat (R.). – *Load Balancing of Multi-physics Simulation by Multi-criteria Graph Partitioning*. – Theses, Université de Bordeaux, décembre 2017.
4. Barat (R.), Chevalier (C.) et Pellegrini (F.). – Partitionnement multi-critères de graphes pour l'équilibrage de charge de simulations multi-physiques. – In *Conférence d'informatique en Parallélisme, Architecture et Système (COMPAS)*, Lorient, France, juillet 2016.
5. Barat (R.), Chevalier (C.) et Pellegrini (F.). – Multi-criteria Graph Partitioning with Scotch. – In Manne (F.), Sanders (P.) et Toledo (S.) (édité par), *SIAM Workshop on Combinatorial Scientific Computing, Proceedings of the Seventh SIAM Workshop on CSC*, Proceedings of the Seventh SIAM Workshop on CSC, pp. 66–75, Bergen, Norway, juin 2018. Society for Industrial and Applied Mathematics and University of Bergen, Society for Industrial and Applied Mathematics.
6. Casadei (A.), Ramet (P.) et Roman (J.). – An improved recursive graph bipartitioning algorithm for well balanced domain decomposition. – In *IEEE International Conference on High Performance Computing (HiPC 2014)*, pp. 1–10, Goa, India, décembre 2014.
7. Couteyen Carpaye (J. M.). – *Contribution à la parallelisation et au passage à l'échelle du code FLUSEPA*. – Theses, Université de Bordeaux, septembre 2016.
8. Couteyen Carpaye (J. M.), Roman (J.) et Brenner (P.). – Towards an efficient Task-based Parallelization over a Runtime System of an Explicit Finite-Volume CFD Code with Adaptive Time Stepping. – In *International Parallel and Distributed Processing Symposium, PDSEC'2016 workshop of IPDPS*, PDSEC'2016 workshop of IPDPS, p. 10, Chicago, IL, United States, mai 2016.
9. Couteyen Carpaye (J. M.), Roman (J.) et Brenner (P.). – Design and Analysis of a Task-based Parallelization over a Runtime System of an Explicit Finite-Volume CFD Code with Adaptive Time Stepping. *International Journal of Computational Science and Engineering*, 2017, pp. 1 – 22.
10. Devine, Karen and Boman, Erik and Riesen, Lee and Catalyurek, Umit and Chevalier, Cédric. – *Zoltan : Parallel toolkit for combinatorial scientific computing : dynamic partitioning, graph coloring and ordering*, 2022.
11. Karypis (G.) et Kumar (V.). – Multilevel algorithms for multi-constraint graph partitioning. – pp. 28– 28, 12 1998.
12. Karypis, George and Kumar, Vipin. – *METIS : A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices*, 2022.
13. Message Passing Interface Forum. – *MPI : A Message-Passing Interface Standard Version 4.0*, juin 2021.
14. Pellegrini, François. – *SCOTCH : Static mapping, graph partitioning, and sparse matrix block ordering package*, décembre 2021.
15. Schloegel (K.), Karypis (G.) et Kumar (V.). – Parallel multilevel algorithms for multi-constraint graph partitioning (distinguished paper). – pp. 296–310, 01 2000.
16. Schloegel (K.), Karypis (G.), Kumar (V.), Dongarra (J.), Foster (I.), Fox (G.), Kennedy (K.), White (A.) et Kaufmann (M.). – Graph partitioning for high performance scientific

simulations. 06 2000.