Rapport technique TR-CMGC-15-40

Les Performances Des Coupleurs OASIS3-MCT et OpenPALM Pour Les Grilles Icosaèdres

Yamina BOUMEDIENE
Sophie VALCKE
Laure COQUART

Avril 2015

Ce travail a été financé grâce au projet européen IS-ENES2 (Grant agreement no: 312979)

Résumé

Les grilles icosaèdres ont acquis ces dernières années une certaine importance dans le domaine de modélisation du climat grâce à leurs propriétés qui permettent de résoudre de nombreux problèmes de modélisation liés à la convergence des lignes de longitude vers le pôle (singularité dans la résolution des équations, taille non uniforme des mailles, etc.).

L'objectif principal de ce travail était de tester et de comparer les performances des deux coupleurs du CERFACS, OASIS3-MCT et OpenPALM, pour des applications couplées utilisant un maillage non structuré de type icosaèdre. En effet, OpenPALM utilise la librairie d'interpolation et de communication CWIPI développée pour traiter des maillages non structurés alors qu'OASIS3-MCT utilise la librairie MCT initialement écrite pour des grilles de points structurées.

Un modèle jouet NICAM_ORCA couplant deux codes, le premier utilisant la grille atmosphérique du modèle NICAM de type icosaèdre non structuré et le deuxième utilisant une grille océanique logiquement rectangle appelée ORCA, a été développé. Des problèmes ont été rencontrés pendant l'interfaçage du code utilisant la grille icosaèdre avec les deux coupleurs à cause de sa structure complexe et de l'absence de certaines informations concernant cette grille.

Deux problèmes ont été rencontrés pendant l'interfaçage de ce code avec le coupleur OASIS3-MCT liés au fait que pour respecter le partitionnement du maillage réellement utilisé dans le modèle NICAM, il fallait pour chaque niveau de partitionnement soit adapter le codage de la déclaration des partitions dans le code, soit faire varier l'indexage global de la grille de façon à suivre séquentiellement les partitions les unes après les autres. Cette dernière solution a été choisie et il a donc fallu réorganiser pour chaque niveau de partitionnement, les vecteurs de longitudes, latitudes et masques décrivant les points de la grille dans les fichiers d'entrée du coupleur afin de respecter l'indexage propre à chaque niveau de partitionnement.

Le deuxième problème était lié au fait qu'OASIS3-MCT effectue en séquentiel la génération des fichiers de poids et d'adresses nécessaires pour l'interpolation des champs de couplage, ce qui peut prendre des heures de calcul. Plutôt que de laisser le coupleur regénérer les fichiers de poids et d'adresses pour chaque niveau de partition, la solution adoptée a été de réorganiser les poids et les adresses de ces fichiers afin de respecter l'indexage propre à chaque niveau de décompositions, à partir d'un résultat obtenu pour un certain niveau de partitionnement.

Deux difficultés particulières ont été rencontrées lors l'interfaçage du même code avec le coupleur OpenPALM. La première était liée à l'obligation de fournir au coupleur une grille sous forme de mailles plutôt que comme un ensemble de points. Pour résoudre ce problème, il a été nécessaire de créer un maillage représentant la grille à partir de l'ensemble de ses points.

La deuxième difficulté est venu du fait que la grille NICAM est associée à un masque qui sert à différencier les points continentaux des points océaniques. Hors cette information ne peut être explicitement transmise et considérée par OpenPALM. Il a donc fallu régénérer le masque des mailles à partir de celui des coins des mailles et filtrer le maillage, de façon à ne prendre en considération que les mailles non masquées.

L'analyse des performances des deux coupleurs a été faite à partir des mesures du temps utilisé pour effectuer des échanges de type ping-pong entre les deux codes (soit un aller-retour d'un champ de couplage). La première conclusion est qu'OASIS3-MCT gère bien les grilles de type icosaèdre car les résultats sont tout à fait raisonnables, le temps d'un ping-pong étant inférieur à 0.1 sec jusqu'à 1000 cœurs. Les résultats avec Open-PALM sont aussi très bons, mais cela est moins surprenant car la librairie d'interpolation CWIPI a été conçue pour des grilles non-struturées. Finalement, en analysant les résultats, on a émis l'hypothèse que la performance des échanges est liée à la correspondance géographique des partitions des codes source et cible: moins il y a de correspondance, plus on aura un schéma de communications complexe et moins les échanges seront performants. Cette hypothèse a été vérifiée au premier ordre avec un modèle jouet couplant des codes dont les partitionnements représente plus de correspondance géographique, Pour valider complètement notre hypothèse, il faudrait faire des tests à nombre plus élevé de cœurs sur d'autres super-calculateurs.

1. Introduction

Le couplage de codes numériques a acquis ces dernières années une importance de tout premier plan dans de nombreux domaines scientifiques tels la modélisation du climat, l'assimilation de données, la combustion, la mécanique des fluides et des structures. Dans ce cadre, l'équipe "Modélisation du Climat et de son Changement Global" du CERFACS, développe les logiciels OASIS et OpenPALM qui permettent de coupler des codes numériques c'est-à-dire d'échanger de façon synchronisée de l'information entre ces codes en l'interpolant spatialement à leur interface.

OASIS (http://oasis.enes.org) a été conçu pour coupler des modèles représentant les différentes composantes du système climatique (modèles de circulation générale atmosphérique, de circulation générale océanique, de glace de mer, de sol, d'hydrologie, etc.). Il est aujourd'hui utilisé par environ 35 groupes de recherche en modélisation climatique en France et en Europe mais aussi aux États-Unis, au Canada, au Japon et en Australie.

Le coupleur OpenPALM (http://www.cerfacs.fr/globc/PALM_WEB/index.html) est quant à lui utilisé dans un large spectre de projets industriels et de recherche allant de l'assimilation de donnée opérationnelle à la modélisation multi-physique avec, par exemple, des applications en aérothermique des systèmes propulsifs et des études d'interaction fluides-structures.

Pour un grand nombre d'applications numériques couplées, il est aujourd'hui indispensable d'utiliser efficacement les architectures de calcul massivement parallèles qui devraient offrir des capacités exaflopiques d'ici la fin de la décennie. C'est dans ce contexte en pleine évolution que le CERFACS développe les coupleurs OASIS3-MCT et OpenPALM.

2. Objectif

L'objectif principal de ce travail était de tester les performances des deux coupleurs du CERFACS, OASIS3-MCT et OpenPALM, pour des applications couplées utilisant un maillage non structuré de type icosaèdre. En effet. OpenPALM utilise la librairie d'interpolation et de communication CWIPI développée pour traiter des maillages non structurés alors qu'OASIS3-MCT utilise la librairie MCT initialement écrite pour des grilles de points structurées. Il s'agissait dans un premier temps de développer des modèles couplés « jouets » utilisant pour le couplage d'une part OASIS3-MCT et d'autre part OpenPALM. Les modèles couplés jouets sont des codes qui n'effectuent pas de réels calculs mais qui reproduisent des échanges de couplage réalistes basés sur les mêmes grilles et maillages que ceux mis en œuvre dans les vrais modèles. L'objectif final de ce travail était d'étudier et de comparer les performances des deux coupleurs à nombre élevé de processus.

3. Description des coupleurs du CERFACS

Échanger de l'information entre des gros codes de simulation sur des machines massivement parallèles nécessite une gestion efficace des communications entre ces derniers, d'où la nécessité d'utiliser un coupleur qui distribue les tâches sur les milliers de processeurs nécessaires au calcul et organise les communications de manière efficace entre les codes.

3.1 OASIS3-MCT [6]

La dernière version d'OASIS, OASIS3-MCT, est interfacée avec la librairie « Model Coupling Toolkit » (MCT) développée au Los Alamos National Laboratory (É-U) et permet un échange des champs et de l'interpolation totalement parallèle.
Pour coupler des modèles avec la librairie OASIS3-MCT, ces derniers doivent être interfacés avec cette dernière en faisant appel aux

routines spécifiques à chaque étape du couplage. La figure 1 montre une abstraction haut niveau du couplage entre 2 modèles avec OASIS3-MCT.

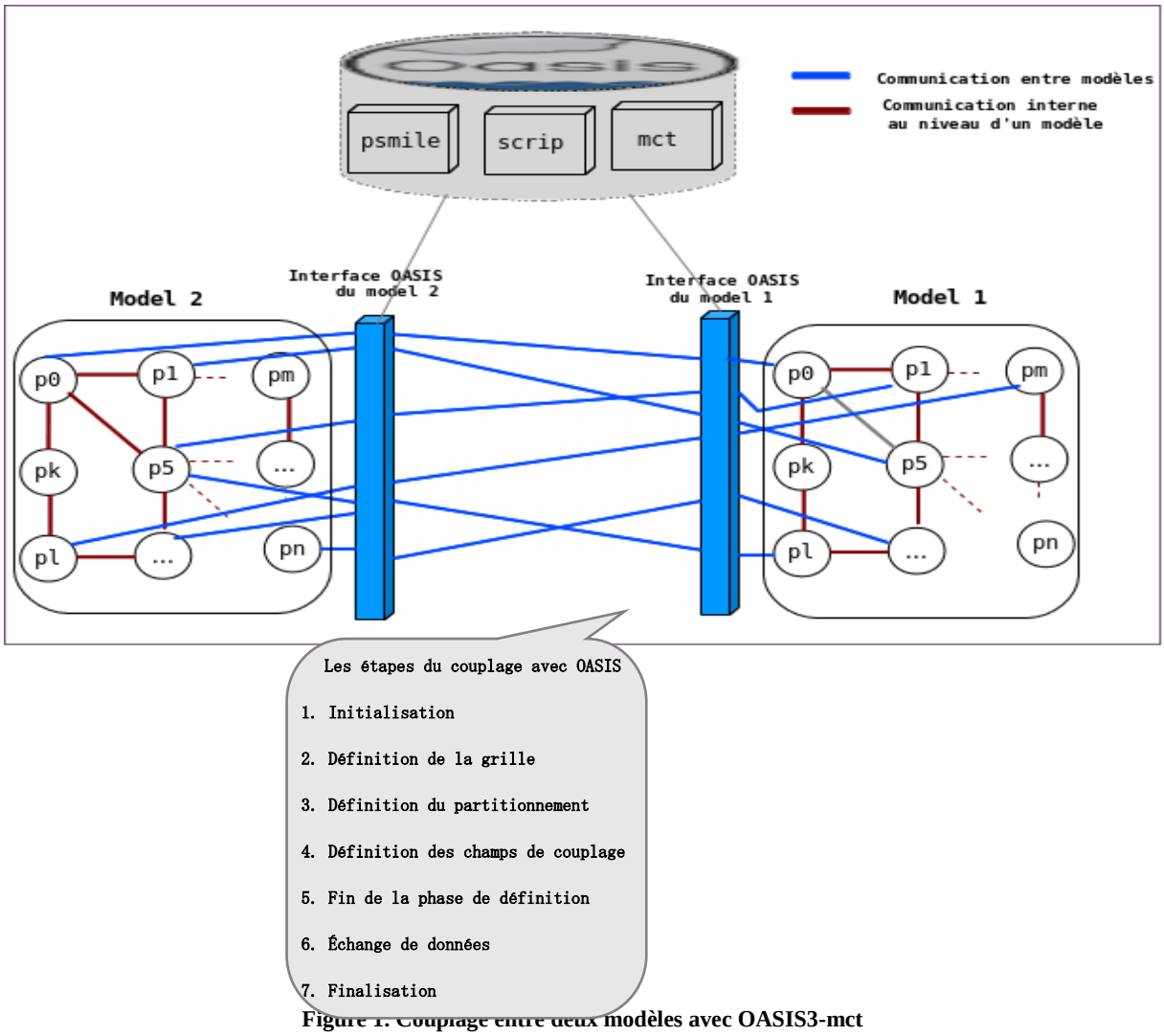


Figure 1. Couplage entre deux modèles avec OASIS3-mct

1. Initialisation : Consiste à initialiser l'environnement du couplage et à créer un communicateur local reliant les processus qui participent au couplage.
2. Définition des fichiers contenant les longitudes,latitudes et le masque de la grille : L'utilisateur a le choix entre définir ces fichiers en appelant dans le code à coupler les routines OASIS3-MCT spécifiques pour cela, ou les générer lui même au préalable.
3. Définition du partitionnement : Les champs de couplage envoyés ou reçus par un modèle sont habituellement décomposés entre les différents processus de ce dernier. Avec OASIS3-MCT, tous les processus qui échangent des données de couplage doivent définir leur partition locale dans un espace d'indexage global de la grille.
4. Définition des champs de couplage : chaque processus du modèle doit déclarer les champs envoyés ou reçus durant la simulation.
5. Fin de la phase de définition : L'établissement des schémas de communication entre les processus des modèles ainsi que la génération des fichiers de sorties nécessaires des étapes précédentes se fait à ce niveau, ainsi qu'un point de synchronisation global entre tous les processus avant de commencer l'échange de données.
6. Échange de données : Dans la boucle temporelle du modèle, chaque processus envoie et/ou reçoit sa part des champs couplés.
7. Finalisation : Tous les processus doivent appeler la routine spécifique pour assurer une fin normale du couplage.

3.2 OpenPALM [5]

Depuis janvier 2011, ce coupleur est développé en collaboration avec l'ONERA et inclut une interface vers la librairie d'interpolation

et de communication parallèle CWIPI développée par cet organisme. Un modèle interfacé par CWIPI doit passer par les étapes suivantes (Figure 2) :

1. Initialisation de la bibliothèque CWIPI et redirection des sorties CWIPI dans des fichiers spécifiques d'OpenPALM. Point de synchronisation de toutes les unités OpenPALM utilisant CWIPI.
2. Création d'un environnement de couplage (objet de couplage).
3. Définition du maillage du couplage (voir 4.6).
4. Échange de données.
5. Suppression de l'objet de couplage.
6. Finalisation.

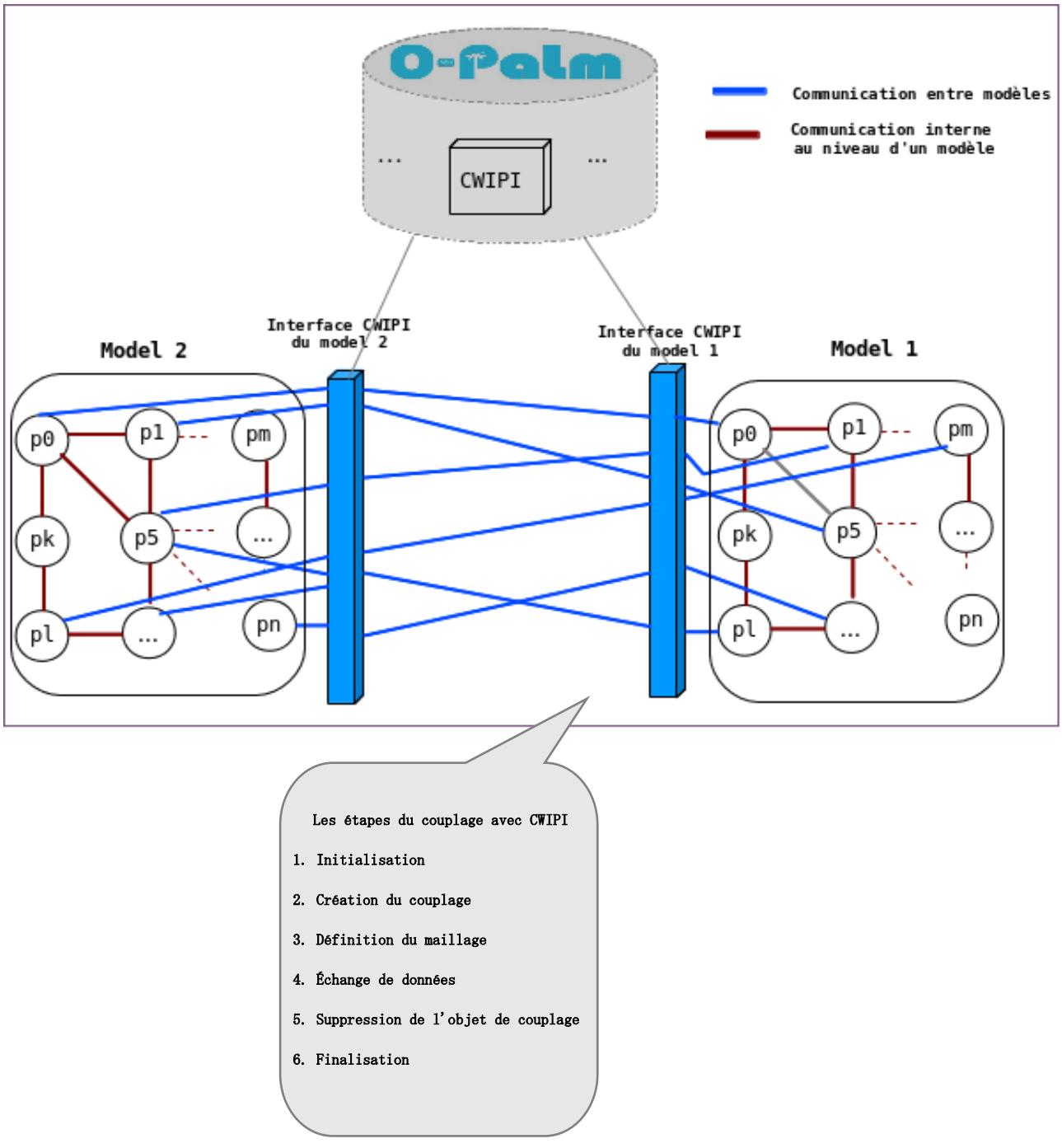


Figure 2. Couplage entre deux modèles en utilisant la librairie d'interpolation CWIPI de Open-PALM

4. La grille icosaèdre

4.1 Les spécifications d'une grille icosaèdre [7]

Au cours des dernières décennies, les météorologues et les océanographes ont développé différentes méthodes pour résoudre les équations qui décrivent l'écoulement de fluides sur une sphère. De telles équations sont utilisées dans les OGCMs et OGCMs (Ocean / Atmosphere General Circulation Models). Certains AGCMs utilisent les méthodes de différences finies, dans lesquelles les rapports des différences donnent une approximations des dérivées qui apparaissent dans les équations différentielles du problème. Ce qu'on appelle « **le problème de pôle** » se pose quand on applique la méthode de différences finies pour des grilles « longitude-latitude » (cf la figure 3), sur des valeurs de longitude ou latitude de la grille pour lesquelles les méridiens (les lignes de longitude constante) convergent vers les deux pôles (Figure 3)

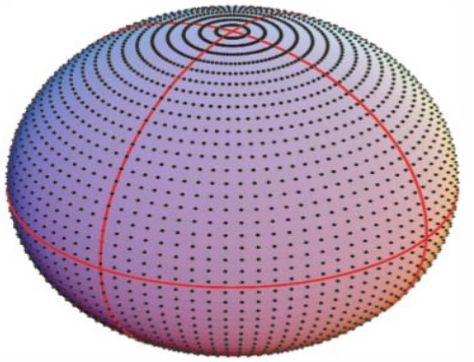


Figure 3. Un exemple de « longitude-latitude grid ». Les points noirs représentent les centres des cellules qui sont espacées en longitudes (d'ouest en est) et latitudes (du sud au nord). Les lignes rouges sont l'équateur et deux lignes de longitude constante.

Des petits pas de temps pourraient être utilisés, pour maintenir la stabilité des calculs près des pôles, mais ceci entrainerait un gaspillage de ressources, car le modèle utilise une résolution inférieure ailleurs.
La meilleure méthode pour résoudre le problème de pôle est d'utiliser des grilles sphériques quasi-uniformes générées à partir d'icosaèdres ou autres Solides de Platon[1].
Une particularité de ce type de grille est que toutes les cellules ont la même taille,ce qui permet de respecter uniformément la condition de « CFL (Courant–Friedrichs–Lewy)[2] »

La génération de la grille icosaèdre [1]

Pour construire la grille icosaèdre, on commence par un icosaèdre (Figure 4), qui se constitue de 20 faces triangulaires. Le théorème d'Euler dit que :
le nombre de segments est : E=F+V-2 => E=30 ,(E,F,V signifient respectivement, edges,faces,vertices).

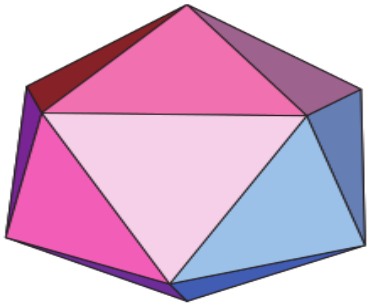


Figure 4. Un icosaèdre avec 20 faces triangulaires,12 vertex, et 30 segments.

[1]. En géométrie euclidienne , un **solide de Platon** est un polyèdre régulier et convexe. Il existent uniquement 5 solides de Platon http://fr.wikipedia.org/wiki/Solide_de_Platon

[2]. CFL est une condition de stabilité de calcul pour la résolution des équations aux dérivées partielles numériquement en utilisant la méthode des différences finies. http://en.wikipedia.org/wiki/Courant%E2%80%93Friedrichs%E2%80%93Lewy_condition

La prochaine étape est de construire les cellules de Voronoï centrées aux 12 vertex. Une cellule de Voronoï est l'ensemble de touts les points qui sont les plus proches d'un sommet donné que de toute autre sommet. Les cellules de Voronoï basées sur l'icosaèdre, donnent une discrétisation de la sphère qui se composent de 12 faces pentagonales. Cependant,on a besoin d'une grille plus fine car 12 faces sont insuffisantes pour donner une discrétisation qui représente des processus complexes du système Terre.
Comme le montre la figure 5, une première méthode pour partitionner la grille est de diviser en deux les dimensions de chaque cellule triangulaire de l'icosaèdre et produire une grille plus fine. Cela divise chaque face triangulaire en 4 faces triangulaires plus petites, et augmente le nombre de faces d'un facteur de quatre.

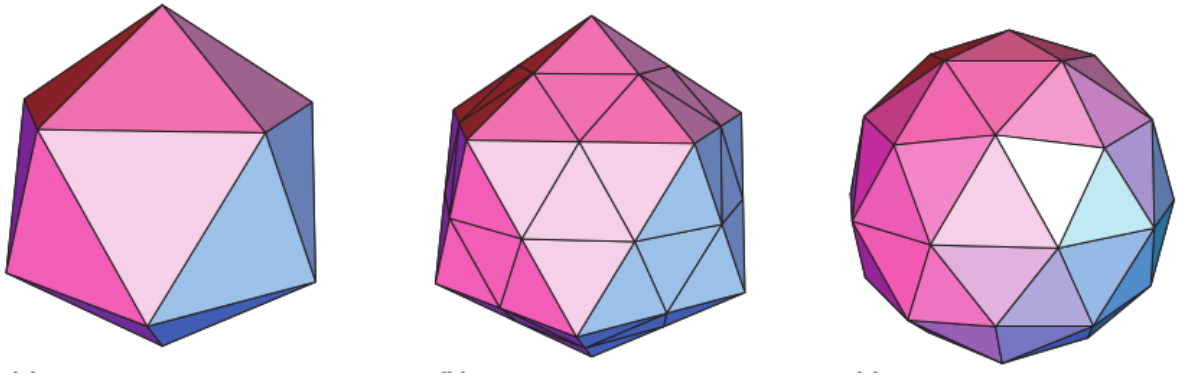


Figure 5. La 1ère étape de division de l'icosaèdre.

Les points de bissection deviennent des nouveaux vertex, et le processus de bissection est répété jusqu'au niveau de résolution souhaité.
Une autre méthode pour décomposer la grille icosaèdre est de la diviser en 10 panneaux rectangulaires, chacun correspond à une paire de faces triangulaires de l'icosaèdre original. La figure 6 montre les 10 panneaux rangés deux à deux pour former logiquement 5 panneaux rectangulaires composés de colonnes et lignes formant des tableaux rectangulaires de mailles.

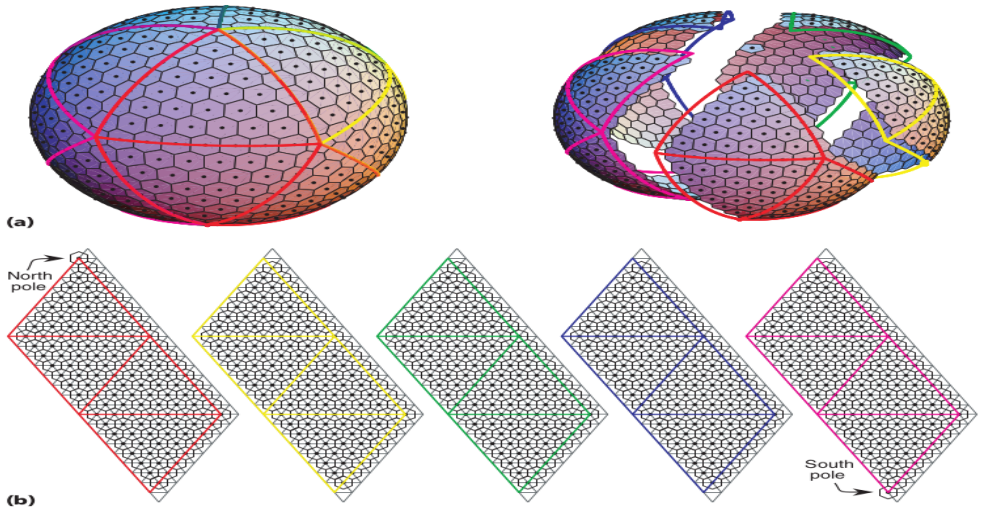


Figure 6. On peut diviser (a) une grille sphérique icosaèdre en (b) panneaux rectangulaires, qui offrent une manière pratique de stocker les données en mémoire de l'ordinateur.

Les deux mailles pentagonales restantes correspondent aux points où les panneaux se rejoignent aux deux pôles. Les panneaux rectangulaires paraissant dans la figure correspondent à l'organisation rectangulaire des données de la grille dans la mémoire de l'ordinateur.

4.2 La grille icosaèdre japonaise « NICAM »

La grille NICAM créée en 2007, par H.Tomita et M.Satoh au JAMSTEC (Japan Agency for Marine-Earth Science and Technology), est une grille icosaèdre de 2621440 points, chaque point est caractérisé par sa longitude et sa latitude. La grille est générée en utilisant la méthode des 10 panneaux rectangulaires qu'on appellera aussi ici « diamants » (voir 4.1, deuxième méthode).

Pour faciliter l'exploitation des données de la grille en parallèle, NICAM doit être partitionné selon différents niveaux de parallélisme pré définis comme le montrent les figures 5, 6, et 7.
Table 1 montre des détails concernant des niveaux de partitionnement possibles pour NICAM.

Nombre total de partitions	Nombre de partitions par diamant	Nombre de points par partition	Nombre de proc
10	1	262144	10
40	4	65536	40
160	16	16384	160
640	64	4096	640
2560	256	1024	2560
10240	1024	256	10240

Table 1. Les niveaux de partitionnement de la grille NICAM.

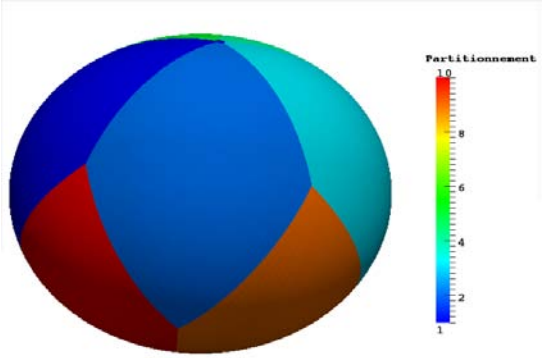


Figure 7. La grille NICAM avec 10 partitions

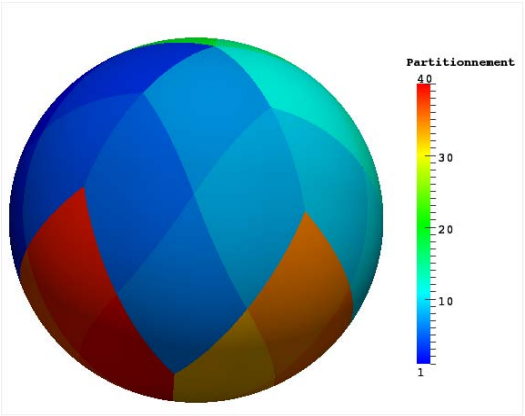


Figure 8. La grille NICAM avec 40 partitions. Nombre total de partitions=nombre de diamants*nombre de dans chaque diamant=10*4

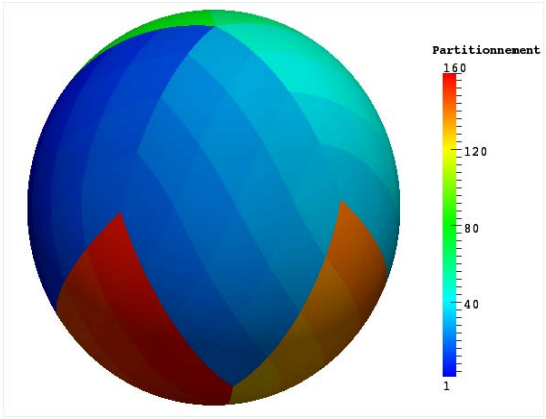


Figure 9. La grille NICAM avec 160 partitions, nombre de partitions=10*16

5. L'implémentation et le couplage des modèles jouets couplés

Les modèles couplés jouets utilisés

Quatre modèles couplés jouets ont été développés ou utilisés au cours de ce travail. Rappelons ici qu'un modèle couplé jouet intègre deux codes qui n'effectuent pas de réels calculs mais qui définissent et échangent des champs de couplage sur les mêmes grilles et maillages que ceux utilisés dans les vrais modèles. Dans notre cas, des échanges de couplage de type « ping-pong » ont été implémentés entre les deux codes grâce aux librairies d'OASIS3-MCT et d'OpenPALM : le premier code envoie un champ au deuxième code qui le reçoit et qui renvoie un autre champ en retour. Un échange de type ping-pong implique donc, en plus de la communication des champs de couplage dans un sens et puis dans l'autre, l'interpolation du premier champ de la grille du premier code vers la grille du deuxième code et l'interpolation du deuxième champ de la grille du deuxième code vers la grille du premier code. En mesurant le temps moyen requis pour un échange ping-pong, on aura donc une information sur la performance générale du coupleur tant pour la communication des données de couplage que pour leur interpolation.

Le modèle couplé jouet NICAM_ORCA avec OASIS3-MCT

Il s'agissait dans un premier temps d'implémenter un modèle jouet qu'on appellera ici NICAM_ORCA couplant deux codes interfacés par OASIS3-MCT. Ces deux derniers définissent leur champ de couplage en chaque point de leur grille grâce à une fonction en cosinus présentant une période sur le globe, soit un minimum et un maximum (voir la figure 10). Le premier utilise la grille atmosphérique de NICAM (présentée en section 4.2) suivant les étapes d'implémentation détaillées en section 5.2 et le deuxième est un code (qui était déjà implémenté avant le début de ce travail) utilisant une grille océanique logiquement rectangle appelée ORCA, avec 1442x1021 points. Chaque point de la grille ORCA peut être repéré par sa longitude et sa latitude exprimées par des tableaux 2D (e.g. lon(i,j), lat(i,j)) mais les lignes de la grille convergent vers 3 pôles situés sur les continents (un sur l'Antarctique, le 2e sur l'Amérique, le 3e sur l'Asie) afin de ne pas avoir à résoudre de singularité dans le domaine océanique, tel que montré à la figure 11.

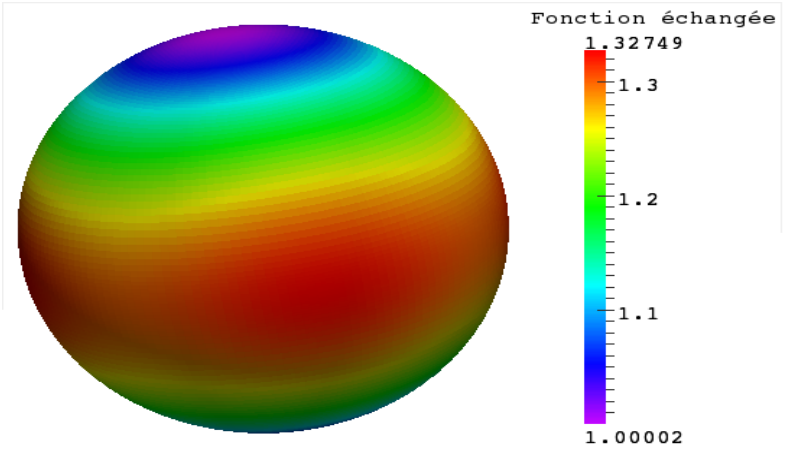


Figure 10. La fonction utilisée dans les modèles jouets pour définir les champs de couplage.

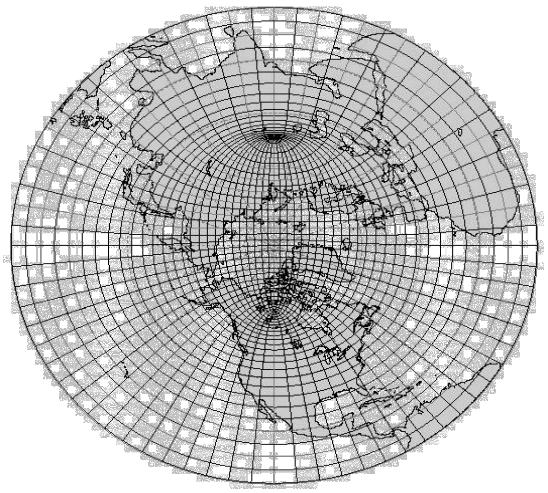
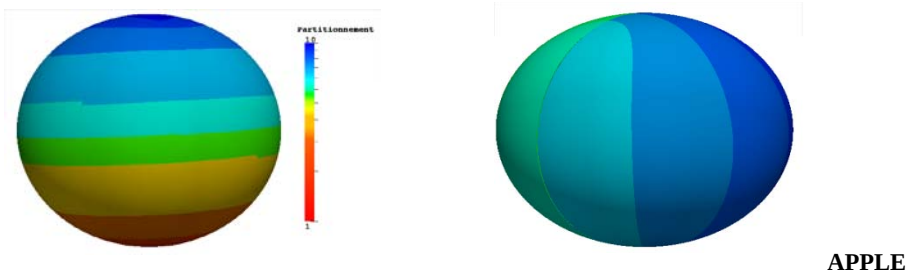


Figure 11. Vision en projection polaire de la grille ORCA avec les deux pôles de convergence sur les continents.

Concernant le partitionnement des champs de couplage, le modèle NICAM utilise le type de partitionnement présenté en section 4.2, et le modèle ORCA quant à lui, utilise trois types de partitionnement montrés par la figure 12.



BOXi **BOXij** **APPLE**

Figure 12. Les partitionnements utilisés par le modèle ORCA

Le modèle couplé jouet NICAM_ORCA avec OpenPALM

Le modèle jouet NICAM_ORCA couplant deux codes utilisant les mêmes grilles que celles décrites ci-dessus, soit les grilles NICAM et ORCA, et implémentant la même fonction pour les champs de couplage a aussi été interfacé avecOpenPALM. Aucun de ces deux derniers codes jouets n'existait. et les détails d'implémentation sont abordés respectivement en sections 5.3.1 et 5.3.2.

Le modèle couplé jouet T799_ORCA avec OASIS3-MCT

Afin de vérifier certaines hypothèses émises pendant les comparaisons de performances des deux coupleurs (voir section 6), un autre modèle jouet couplé par OASIS3-MCT , qu'on appellera ici T799_ORCA, a été utilisé. Ce modèle couple grâce à OASIS3-MCT un code jouet qui utilise une grille gaussienne réduite appelée T799 avec 843490 points, et le code jouet utilisant la grille ORCA décrite ci-dessus. Une grille gaussienne réduite est une grille dont les points sont équidistants le long d'une latitude donnée (donc sur un *parallèle*) mais dont le nombre de points à latitude donnée se réduit lorsqu'on se rapproche d'un pôle (voir la figure 13).

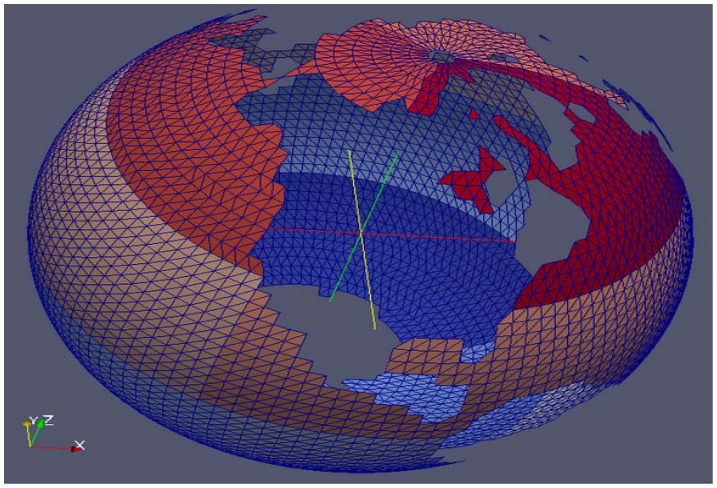


Figure 13. La grille gaussienne reduite

Dans le cas de T799_ORCA, le modèle T799 utilise deux type de partitionnements, on appelera le premier « ORANGE » ; c'est un partitionnement qui ressemble a celui de BOXij (voir figure 14) mais dont les boites ne sont pas toute a fait rectangulaires comme la grille est réduite, le deuxième est APPLE (figure 12)

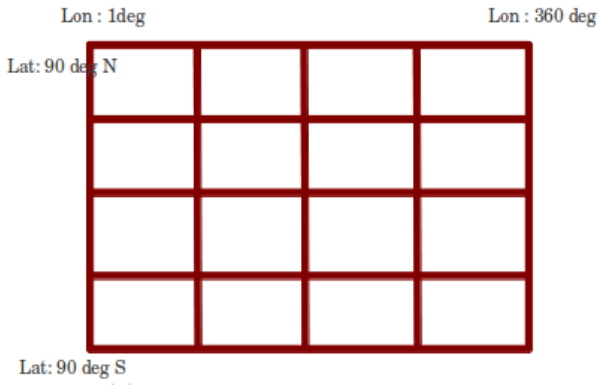


Figure 14. Le partitionnement Orange utilisés par le modèle T799_ORCA ; chaque rectangle correspond à une partition.

Le modèle couplé jouet T799_ORCA avec OpenPALM

Le modèle jouet T799_ORCA couplant deux codes utilisant les mêmes grilles que celles décrites ci-dessus, soit les grilles T799 et ORCA, et implémentant la même fonction pour les champs de couplage a aussi été interfacé avec OpenPALM. Les détails d'implémentation du modèle T799 sont abordés en sections 5.4.

5.1 Environnement de développement

Les développements ont été faits en Fortran90 , dans un environnement Linux .

La manipulation des fichiers de grilles et d'interpolation d'OASIS3-MCT au format NetCDF (voir les sections 5.2.1 et 5.2.2) a été réalisée grâce au langage NCL qui est un langage de script comportant des fonctionnalités spécifiques pour la manipulation des données de ce format.

Pour la visualisation des différents résultats obtenus, les outils suivants ont été exploités :

Ferret, Paraview, Ncview, Gnuplot, et GIMP.

Les tests de performances ont été effectués sur deux supers ordinateurs massivement parallèles. Le premier est l'IBM MareNostrum du Barcelona Supercomputing Center qui comporte 48,896 processeurs Intel Sandy Bridge réparti sur 3056 nœuds et 84 Xeon Phi 5110P sur 42 nœuds, pour une puissance crête de 1.1 Pflop/s, classée 29e dans le TOP500 de juin 2013. Sur cette machine, les tests ont été réalisés avec le compilateur Intel ifort et la librairie impi MPI 4.1.1.036.

La deuxième machine est la BULL Beaufix de Météo-France comportant 1980 processeurs Intel R Ivy Bridge EP répartis sur 990 nœuds de calcul,. Les tests on été réalisés avec le même compilateur et la même librairie MPI que sur MareNostrum, soit la version Intel MPI 4.1.1.036.

Pour les deux coupleurs, les versions suivantes sont utilisées :

OASIS3-MCT : version 3.0, révision 1153

OpenPALM : version 4.2.0, cwipi-0.8.1

5.2 Implémentation du modèle jouet couplé NICAM_ORCA pour OASIS3-MCT

5.2.1 Génération des fichiers de grilles du coupleur OASIS3-MCT pour les différentes décompositions de NICAM

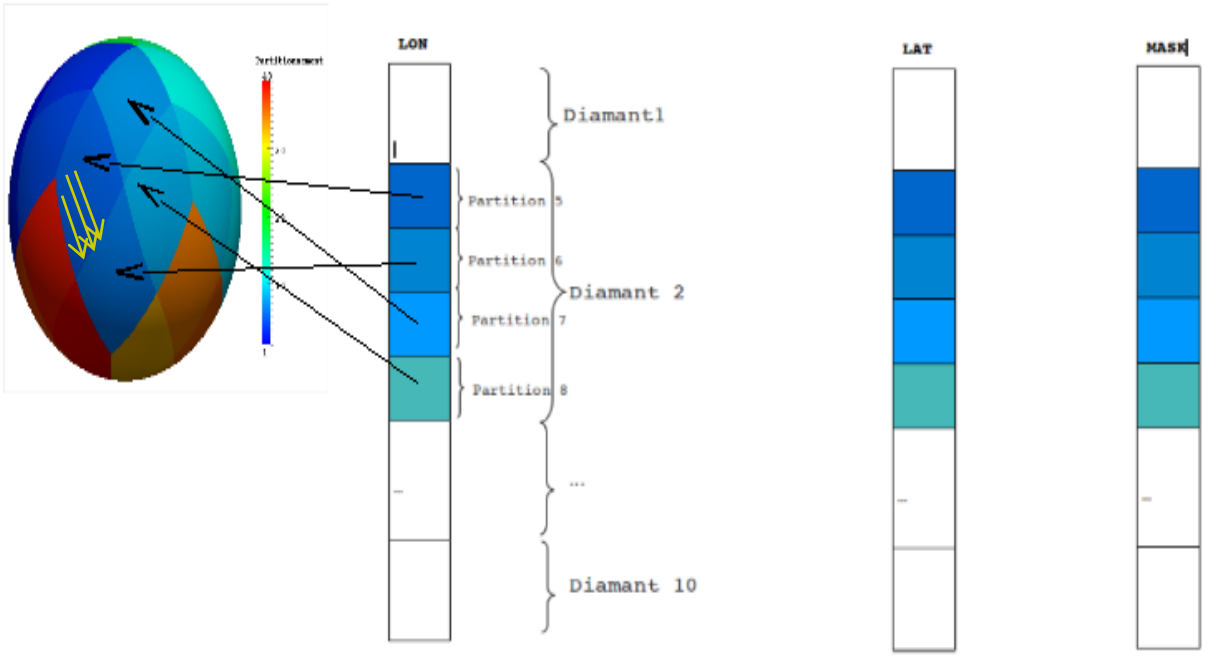


Figure 15. L'organisation des données de NICAM pour une décomposition de 40 partitions

Pour coupler un modèle en utilisant la grille NICAM, le coupleur OASIS3-MCT a besoin de la position (longitude et latitude) et du masque de chaque point de la grille stockés dans des fichiers au format NetCDF[3] dans des vecteurs suivant un indexage global dépendant du partitionnement. La figure 15 montre le contenu d'un fichier qui correspond au découpage en 40 partitions; les deux tableaux « LON » et « LAT » représentent respectivement les longitudes et les latitudes des points, et le tableau « MASK» permet de différencier les continents des océans, en associant la valeur mask(i)=0 aux points qui représentent les continents, ou la valeur mask(i)= 1 sinon.

Description du problème

Tester la scalabilité des deux coupleurs avec la grille NICAM nécessite la disposition des différents partitionnements pour pouvoir faire tourner les modèles avec différents nombre de processus. Pour ne pas avoir à adapter le codage de la déclaration des partitions dans le modèle à chaque partionnement, nous avons décidé de faire varier l'indexage global de la grille de façon à suivre séquentiellement les partitions les unes après les autres pour tous les niveaux de partitionnement (comme c'était d'ailleurs le cas dans dans les fichiers dont on disposait initialement correspondant à un niveau de 2560 partitions.

Solution

La solution adoptée était d'écrire puis d'implémenter un algorithme qui permet, en ré agençant les données des vecteurs de longitudes, latitudes et masques initialement disponibles correspondant au niveau de 2560 partitions , de générer les vecteurs correspondant à d'autres niveaux de partitionnement plus élevés ou plus bas, selon les besoins.

L'algorithme de partitionnement :

L'algorithme se compose de deux parties essentielles, dont la première est de partir d'un certain niveau à un niveau plus élevé, et la deuxième est de partir d'un certain niveau à un autre plus bas.

[3]. Le format NETCDF: est une format de données « auto-documenté », indépendant de l'architecture matérielle qui permet la création, l'accès et le partage de données scientifiques stockées sous la forme de tableaux.

Partie 1. Algorithme de découpage

Étape 1. Il s'agit de découper chaque partition en quatre sous partitions. Les flèches jaunes dans la figure 15 représentent l'ordre dans lequel les points sont stockés.

La figure 16 montre le résultat de découpage d'une partition de taille 8x8 points en 4 partitions de 4x4 points:_ à gauche l'état initial de la partition en représentation géographique (a) et en stockage dans les vecteurs de données (b),_et à droite son état après le découpage. Les points (ou les pixels) sont représentés par les petits rectangles, et identifiés par les indices marqués dessus. Chaque couleur représente une partition.

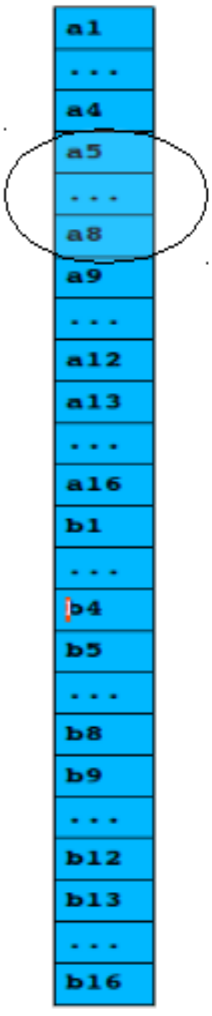
Les indices de positionnement des points dans la mémoire, avant et après le découpage sont représentés dans la figure 10-c,l'expression « x=>y » signifie que le point était à la x ème position avant, et qu'il a été déplacé à la y ème après le découpage.

a1	a9	b1	b9	c1	c9	d1	d9
a2	a10	b2	b10	c2	c10	d2	d10
a3	a11	b3	b11	c3	c11	d3	d11
a4	a12	b4	b12	c4	c12	d4	d12
a5	a13	b5	b13	c5	c13	d5	d13
a6	a14	b6	b14	c6	c14	d6	d14
a7	a15	b7	b15	c7	c15	d7	d15
a8	a16	b8	b16	c8	c16	d8	d16

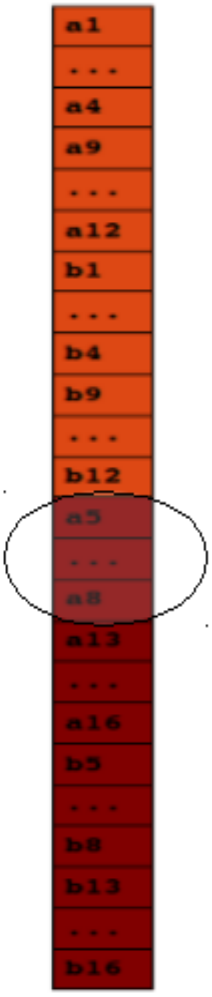
a1	a9	b1	b9	c1	c9	d1	d9
a2	a10	b2	b10	c2	c10	d2	d10
a3	a11	b3	b11	c3	c11	d3	d11
a4	a12	b4	b12	c4	c12	d4	d12
a5	a13	b5	b13	c5	c13	d5	d13
a6	a14	b6	b14	c6	c14	d6	d14
a7	a15	b7	b15	c7	c15	d7	d15
a8	a16	b8	b16	c8	c16	d8	d16

L'algorithme développé pour ce ré agencement est donné en annexe A1.

(a)



(b)



le nouveau positionnement de la 1ère moitié des points dans les vecteurs

(c) Les indices de positionnement des points avant et après le découpage

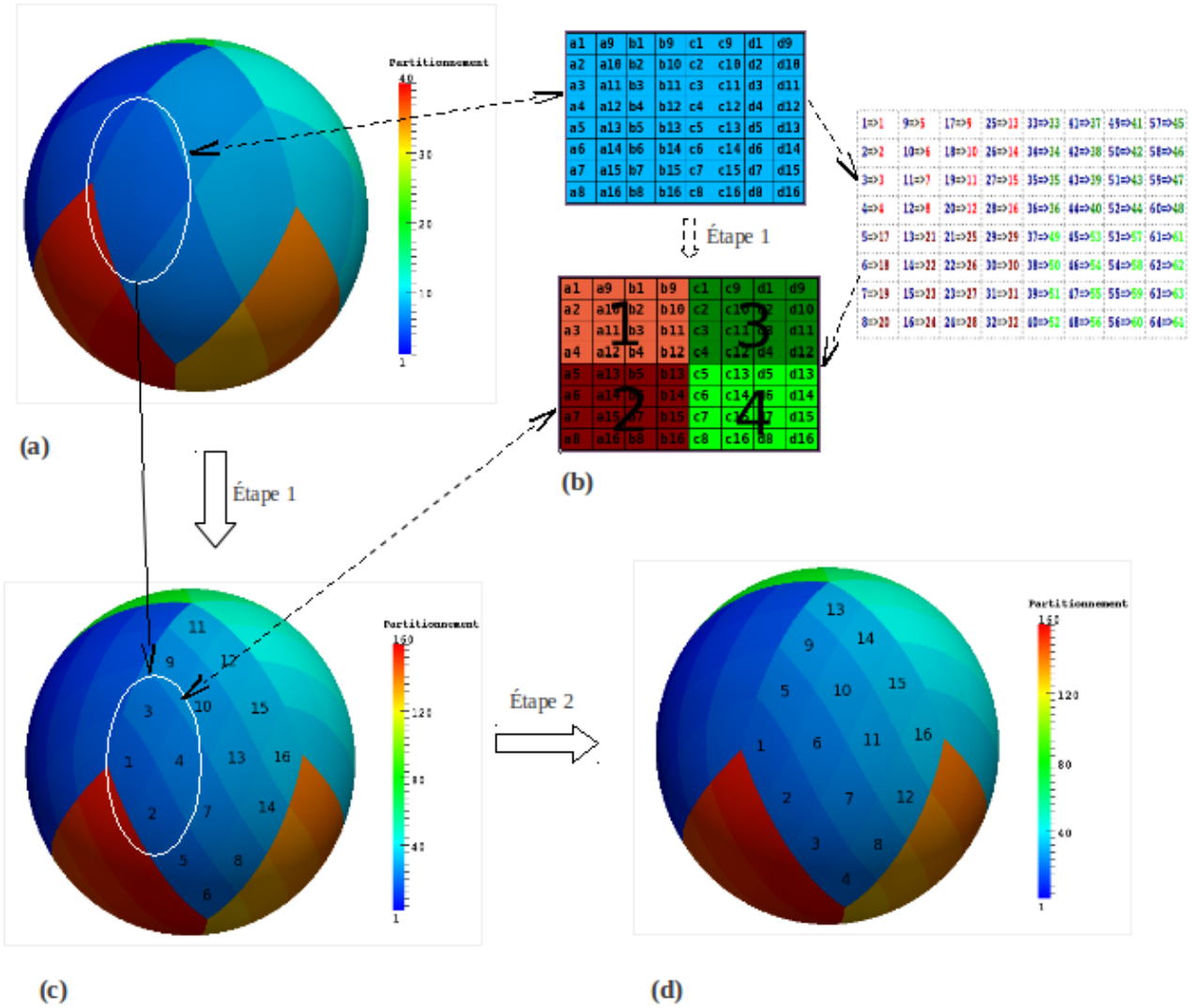
1=>1	9=>5	17=>9	25=>13	33=>33	41=>37	49=>41	57=>45
2=>2	10=>6	18=>10	26=>14	34=>34	42=>38	50=>42	58=>46
3=>3	11=>7	19=>11	27=>15	35=>35	43=>39	51=>43	59=>47
4=>4	12=>8	20=>12	28=>16	36=>36	44=>40	52=>44	60=>48
5=>17	13=>21	21=>25	29=>29	37=>49	45=>53	53=>57	61=>61
6=>18	14=>22	22=>26	30=>30	38=>50	46=>54	54=>58	62=>62
7=>19	15=>23	23=>27	31=>31	39=>51	47=>55	55=>59	63=>63
8=>20	16=>24	24=>28	32=>32	40=>52	48=>56	56=>60	64=>64

Figure 16. Le résultat de découpage d'une partition de taille 8x8 en 4 partitions de 4x4 : (a) représentation géographique; (b) disposition des données dans les vecteurs

La figure 17-c montre le positionnement des nouvelles sous-partitions ; on remarque qu'elles sont stockées en carrés de 2x2 partitions.

Étape 2 . L'étape 2 de l'algorithme consiste à ré ordonner l'ensemble de sous-partitions pour qu'elles soient rangées en colonnes. Le nouveau positionnement des partitions est montré dans la figure 17-d. L'algorithme développé pour cette 2e étape est donné en annexe A2.

Figure 17. Le positionnement des nouvelles sous-partitions après chaque étape de l'algorithme de découpage ; (a) l'état initial



correspondant à un partitionnement de 10x4, (b) le même exemple de la figure 16, (c) le positionnement des partitions après l'étape 1, (d) le positionnement final des partitions (rangement en colonnes au niveau des diamants)

Partie 2. Algorithme de regroupement

Pour avoir un niveau de partitionnement plus bas; il s'agit dans une 1ère étape de regrouper les partitions 4 par 4 pour en former d'autres plus grandes (Figure 18-b), ensuite de ré ordonner les points de chaque nouvelle grande partition pour qu'ils soient rangés en colonnes au niveau de cette dernière (figures 11-c et 11-d).

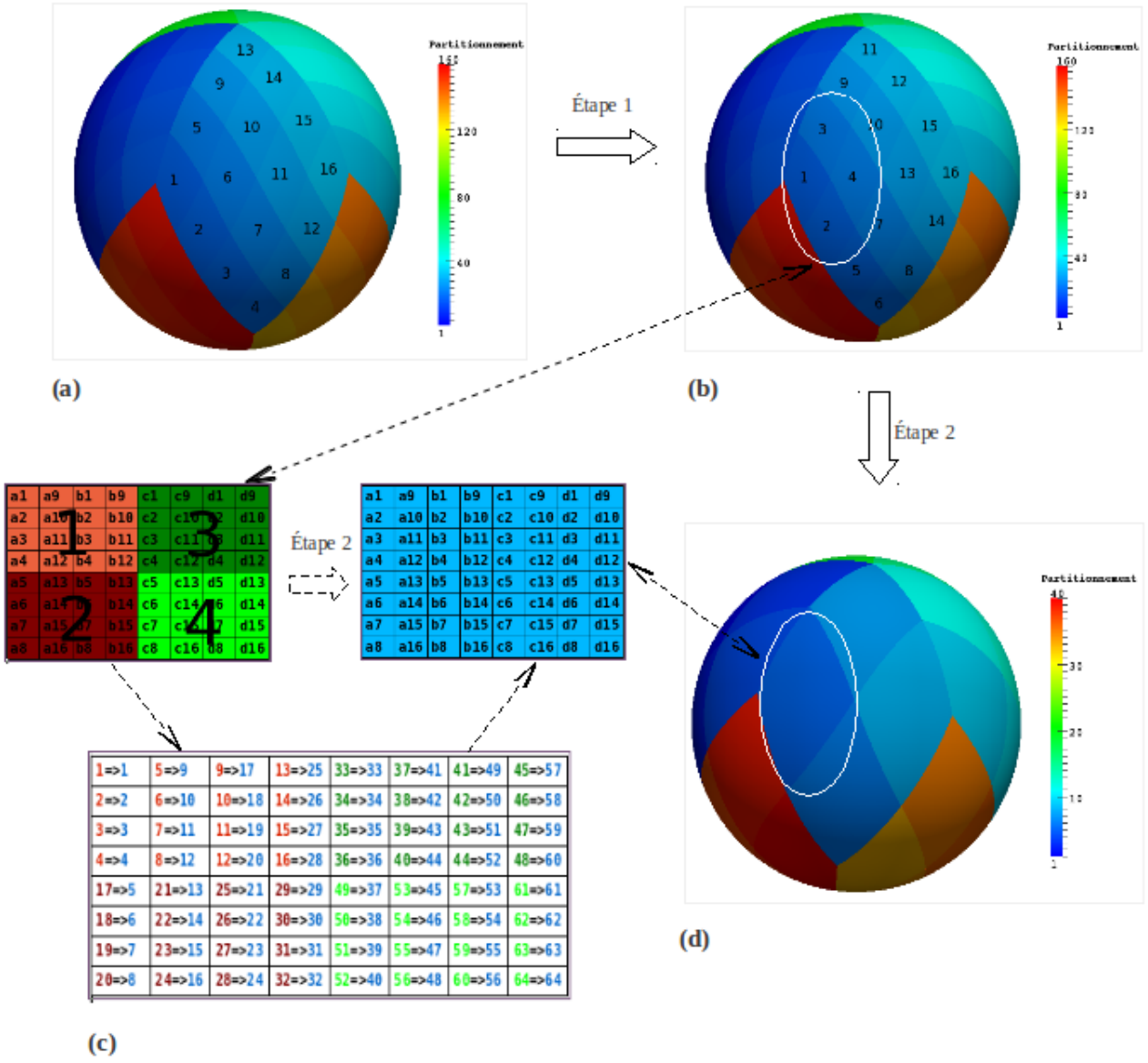


Figure 18. Le positionnement des nouvelles sous-partitions après chaque étape de l'algorithme de regroupement ; (a) l'état initial correspondant à un partitionnement de 10x16, (b) le nouveau positionnement des partitions après l'étape 1, (c) résultat de l'étape 2 appliquée sur 4 partitions de 4x4 pour former une partition de 8x8, (d) résultat final de l'algorithme de regroupement

L'algorithme développé pour les 2 dernières étapes est donné respectivement, en annexes A3 et A4.

5.2.2 Génération et adaptation des fichiers d'interpolation du coupleur OASIS3-MCT pour les différentes décompositions de NICAM

L'échange des données entres 2 modèles qui utilisent des grilles différentes, nécessite l'interpolation des points cibles dans l'ensemble

des points de la grille source. Il s'agit de définir pour chaque point cible les adresses des points sources utilisés pour le calcul de la valeur du champ reçu sur ce point cible et leurs poids respectifs (ce qu'on appelle la recherche des poids et adresses d'interpolation). OASIS3-MCT effectue la recherche des poids et adresses d'interpolation au moment de l'étape « Fin de phase de définition » (voir la figure 1), où il stocke aussi les résultats de ce calcul dans un fichier sur disque, ce qui lui permet de les réutiliser pour faire tourner le même couplage avec le même type d'interpolation plusieurs fois. Il est important de pouvoir stocker ces résultats sur disque car ce calcul séquentiel est très pénalisant en temps de calcul (la recherche des poids et adresses d'interpolation pour des grilles haute résolution prend des heures de calcul). La figure 19 montre le contenu d'un fichier de poids et d'adresses généré par OASIS3-MCT.

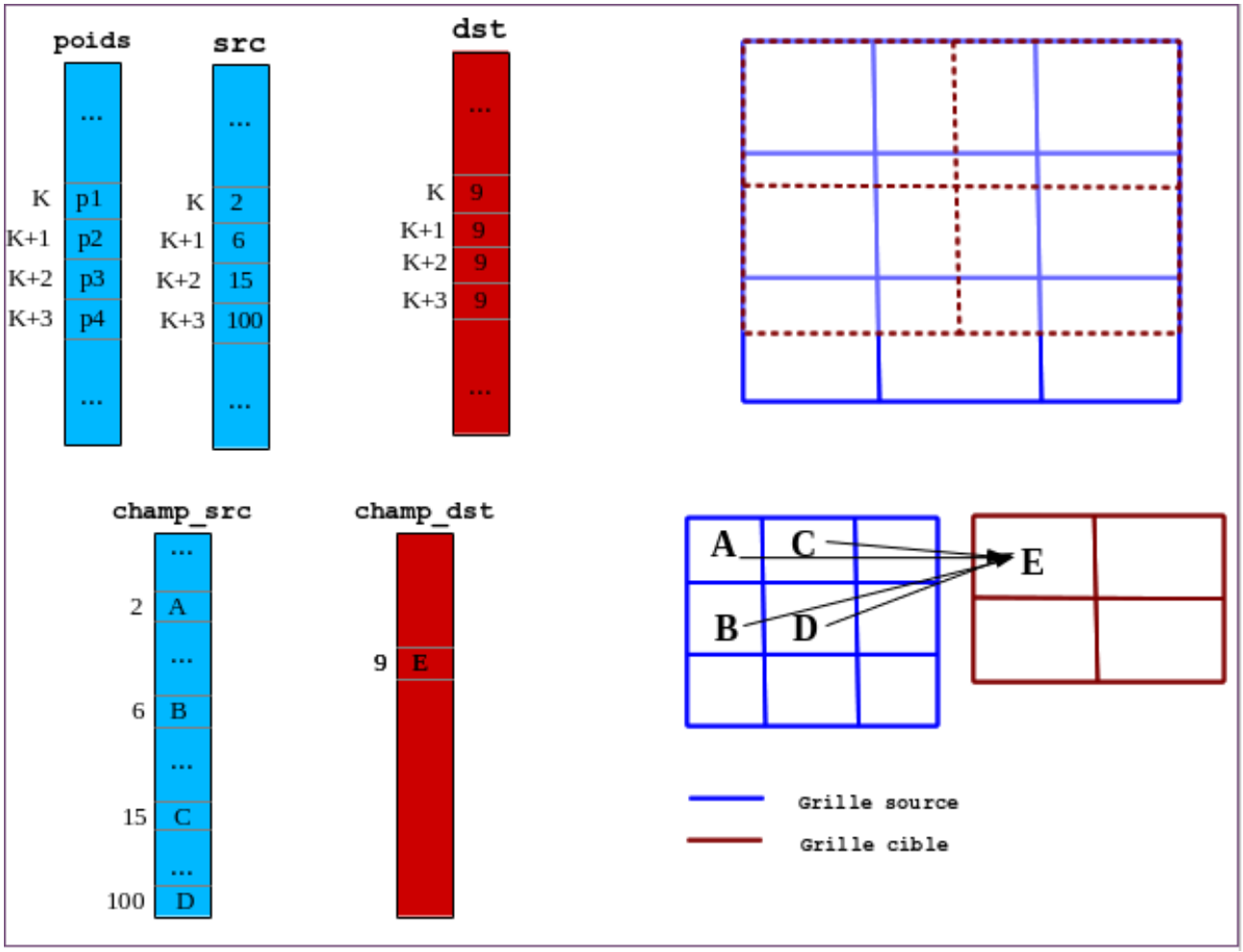


Figure 19. à gauche, le contenu d'un fichier de poids et d'adresses généré par OASIS3-MCT, a droite, un exemple de localisation de 4 points sources A, B, C, et D pour interpoler un point cible E

Les champs de couplage sur les grilles source et cible sont représentés respectivement dans les tableaux « champ_src » et « champ_dst » . Les deux tableaux « src » et « dst » contiennent respectivement les adresses des points sources et cibles associés. L'ensemble poids[k]=p1, src[k]=2 et dst[k]=9 signifie que le 2ème point de la grille source pondéré par un poids p1 contribue au calcul du 9 ème point de la grille cible .

Problème : A cause de la solution adoptée en 2.2.1 , la réutilisation du même fichier de poids et d'adresses n'est plus possible si on change le nombre de processus, puisque l'indexage des points de la grille n'est plus le même pour deux partitionnements différents. Autrement dit, si on prend le tableau « src » pour un couplage où NICAM est la grille source avec 10 partitions, les adresses contenues dans ce tableau ne pointent plus sur les mêmes points pour une grille de 40 partitions, puisque les points de cette dernière correspondent à un indexage différent.

Solution : La solution adoptée était d'écrire puis implémenter un algorithme qui permet à partir d'un certain ensemble de poids et d'adresses calculé pour un certain niveau de partitionnement, de générer d'autres résultats pour d'autres niveaux de partitionnement sans refaire la recherche des poids et des adresses avec OASIS3-MCT. Il s'agit dans un 1er temps d'appliquer l'un des deux algorithmes de regroupement ou de découpage (selon les besoins) présentés dans la section 4.3.2, tout en sauvegardant les nouveaux indices de positionnement des points, ensuite de ré agencer les adresses contenues dans l'un des tableaux « src » (dans le cas où NICAM est la grille source) ou « dst » (dans le cas contraire) selon les nouvelles adresses des points.
L'algorithme développé pour ce calcul est donné en annexe B.

5.3 Implémentation du modèle jouet couplé NICAM_ORCA avec OpenPALM

5.3.1 Adaptation du modèle jouet à la grille NICAM pour OpenPALM

5.3.1.1 Création du maillage

La bibliothèque CWIPI d'OpenPALM n'accepte que des grilles définies comme un maillage i.e. un ensemble de mailles définies par des sommets (et non pas comme un ensemble de points). L'utilisateur doit donc fournir à CWIPI un maillage ce qui correspond à l'étape 3 de l'interfaçage d'un modèle (voir la figure 2) . Dans CWIPI, il existe une association explicite entre un maillage et un couplage. L'attache d'un maillage à un couplage est réalisée via une primitive appelée « PCW_Define_mesh », dans laquelle on précise le nombre de sommets du maillage « n_vertex », son nombre de cellules « n_elements », ainsi que les tableaux de coordonnées des sommets « coords », et de connectivité « « connec_index » et « connec » », comme l'illustre la figure 20.

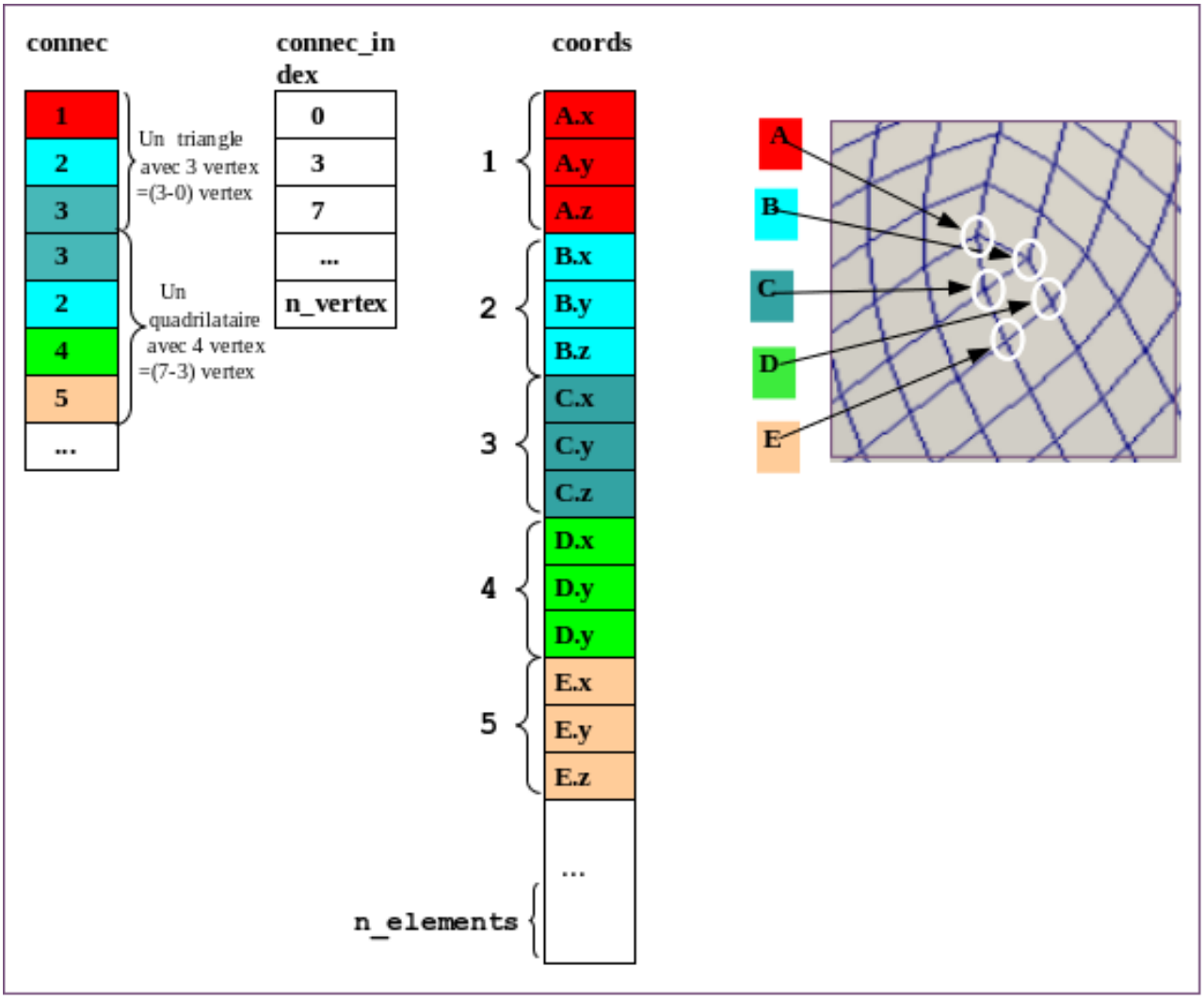


Figure 20. La définition d'un maillage avec CWIPI

Le travail effectué à ce niveau est d'utiliser l'ensemble des points de la grille NICAM pour construire un maillage non structuré

(figure 20). Comme la définition du maillage se fait en local aux niveaux des processus , CWIPI utilise cette information locale pour effectuer la localisation des points et établir les schémas de communications nécessaires au couplage.

Les démarches suivies pour créer le maillage

Pour créer le maillage, il s'agissait de créer des connectivités entre les points voisins géographiquement de manière à former des mailles qui couvrent la surface de la grille. Pour simplifier le traitement, l'ensemble des points de la grille est divisé en deux catégories : la 1ère représente les points situés dans le même diamant, et la 2ème représente les points situés dans les bords des diamants différents et qui sont voisins (voir la figure 21). Ainsi, le traitement se compose de deux parties, chacune pour connecter les points d'une même catégorie.

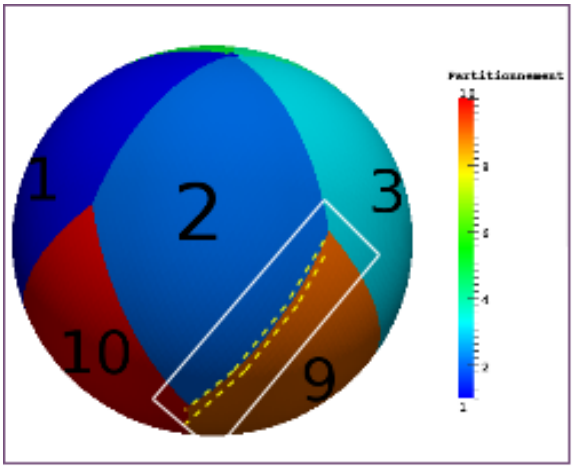


Figure 21. Un exemple des points voisins géographiquement,situés dans deux diamants différents

Pour minimiser la complexité de connecter les points de la 2ème catégorie, nous avons choisit de créer le maillage en séquentiel, à partir de la décomposition (voir 4.3.1) qui correspond au plus bas niveau de partitionnement (nbr partitions=nbr diamants=10), et distribuer ensuite l'information sur l 'ensemble des mailles aux différents processus tout en respectant la méthode de partitionnement de NICAM (voir 4.3.1).

La figure 22 montre les mailles créées en connectant les points de la 1ere catégorie 4 par 4 pour former des quadrilatères.

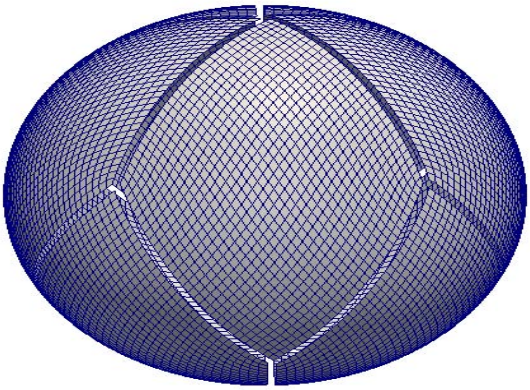


Figure 22. Résultats de la 1ère étape de création du maillage dans laquelle il s'agit de connecter entre les points situés dans le même diamant

La figure 23 montre les mailles créées en connectant les bords de chaque diamant avec ceux des diamants voisins, en formant, en plus des mailles de type quadrilatère, d'autres mailles de type triangle et pentagone afin de relier les coins des diamants .

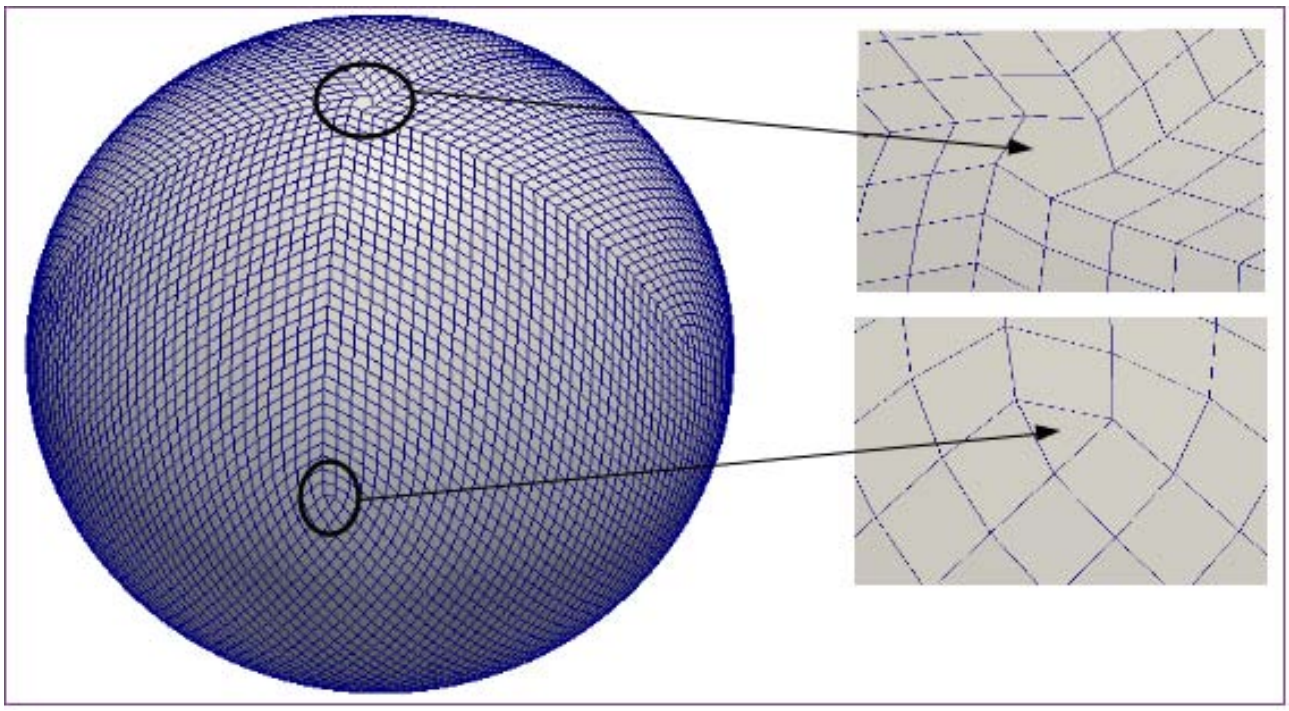


Figure 23. résultat final de la création du maillage

Les deux algorithmes développés pour ce traitement sont donnés en annexe C.

5.3.1.2 Traitement du masque

Le couplage d'un modèle atmosphérique avec un autre océanique consiste à échanger de l'information seulement sur la surface de la grille qui couvre les océans. Par conséquence, une grille qui représente l'océan ou l'atmosphère doit contenir l'information du masque qui sert à différencier les points continentaux des points océaniques. Le coupleur OASIS3-MCT permet de prendre en considération cette information qui doit lui être fournie explicitement durant la phase d'initialisation du couplage (voir 3.1), ce qui n'est pas le cas pour CWIPI de Open-PALM.

Le travail effectué à ce niveau a consisté à filtrer le maillage, de façon à ne garder que les mailles non masquées, avant la réalisation de l'échange d'information. Une autre difficulté est venue du fait que l'information du masque dont on disposait pour NICAM concernait les points et donc les coins des mailles. Il s'agissait dans un premier temps de générer l'information du masque pour chacune des mailles à partir de celle de ses coins en appliquant la règle suivante : si l'un des sommets de la maille n'est pas masqué, la maille n'est pas masquée.

La figure 24 montre la grille NICAM avant et après le traitement du masque.

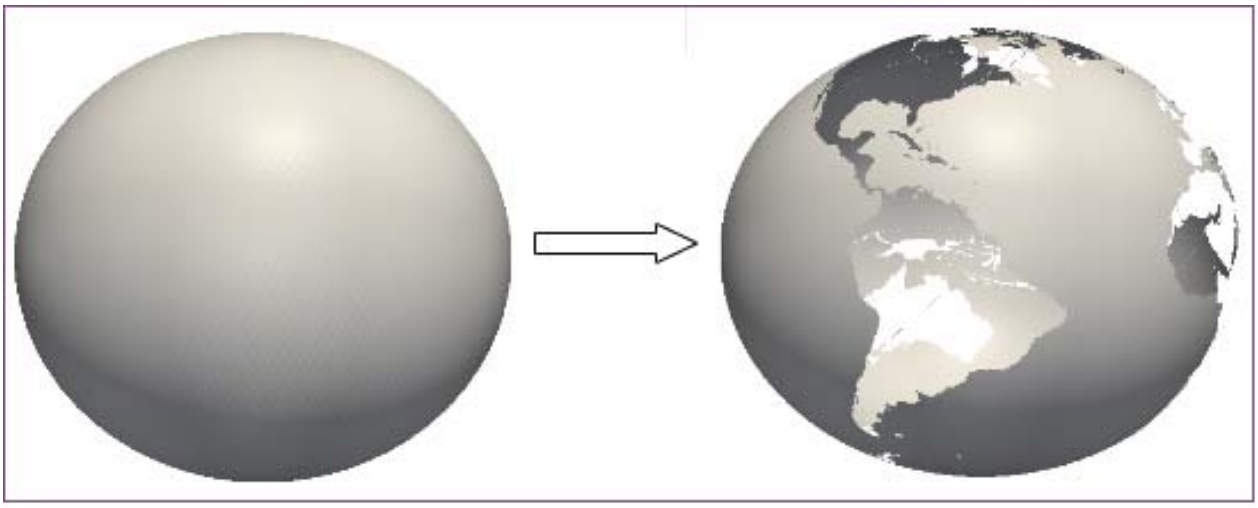


Figure 24. A gauche la grille avant l'application du masque, et à droite le résultat de ce traitement.

5.3.2 Adaptation du modèle jouet à la grille ORCA pour OpenPALM

Contrairement à ce qui s'était passé lors de l'interfaçage du code utilisant la grille NICAM avec OpenPALM, toutes informations nécessaires au couplage avec CWIPI étaient présentes dans le cas de la grille ORCA, en particulier les coins associés à chaque point et formant donc un maillage.

En effet, la grille ORCA est définie dans des fichiers au format NetCDF dans lesquels on retrouve, pour chaque point de longitude(i,j) et latitude(i,j), un ensemble de 4 points (k=1...4) de longitude(i,j,k) et latitude(i,j,k) représentant les coins de la maille associée à ce point. Il s'agissait donc à ce niveau simplement de transformer ces informations au format nécessaire pour CWIPI (voir 5.3.1.1 , la figure 20).

Concernant le masquage des continents, on a simplement associé le masque des points de la grille au masque des mailles ; ainsi si un point était masqué, on a supposé que la maille associée l'était aussi.

5.4 Implémentation du modèle jouet couplé T799_ORCA avec OpenPALM

Il s'agissait à ce niveau d'adapter le modèle jouet à la grille T799 ensuite le coupler avec le même modèle ORCA décrit dans la section précédente.

La grille gaussienne réduite est définit dans des fichiers au format NetCDF dans lesquels on retrouve, pour chaque point de longitude(i) et latitude(i) un ensemble de 4 points de longitude(i,k) et latitude(i,k), (k=1...4) représentant les coins de la maille associée à ce point. Il s'agissait donc à ce niveau de suivre les même étapes suivies dans le modèle ORCA pour transformer les données de la grille au format nécessaire pour CWIPI (section 5.3.1.1 , la figure 20),

et d'appliquer la même procédure pour le masquage des continents.

5.5 Un récapitulatif des modèles implémentés, des grilles et des partitionnements utilisés

La figure 25 montre un récapitulatif des tests effectués en listant les modèles jouets correspondants, leurs grilles et leurs partitionnement. Notons ici que les partitionnements « apple », « apple-lat-entière », et « boxj » sont de fait pratiquement semblables.

Coupleurs	Modèles	partitionnements
OASIS3-mct	NICAM_ORCA025	diamant_apple-lat-entiere
		diamant_boxij
		diamant_boxi
	T799_ORCA025	orange_boxij
		apple_boxij
		apple_apple-lat-entiere
		apple_boxi
OpenPALM	NICAM_ORCA025	diamant_apple
		diamant_apple-lat-entiere
		diamant_boxi
	T799_ORCA025	apple_apple-lat-entiere
		apple_boxi

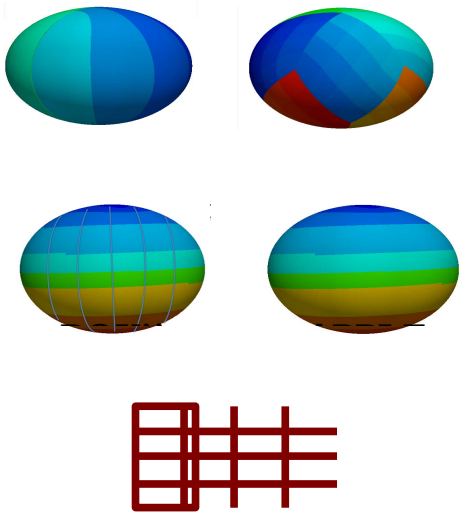


Figure 25. Un récapitulatif des modèles implémentés, des grilles et des partitionnements utilisés

6. Tests de performances

Les tests effectués correspondent à mesurer la vitesse d'un ping-pong pour les deux coupleurs, par rapport aux différents types de grilles utilisés dans les modèles jouets. La mesure du premier ping-pong a été séparée de la mesure des pings-pongs suivants car ils se comportent pas de la même façon. Pour ces derniers, on rapporte ci-dessous la moyenne des 99 échanges suivant le premier, ce qu'on appelle la mesure d'un ping-pong « quelconque ». Ainsi, afin d'obtenir plus de précision pour l'évaluation, les tests de performances sont répétés cinq fois, et chaque mesure est obtenue en calculant la moyenne de 5 mesures correspondantes à cinq tests.

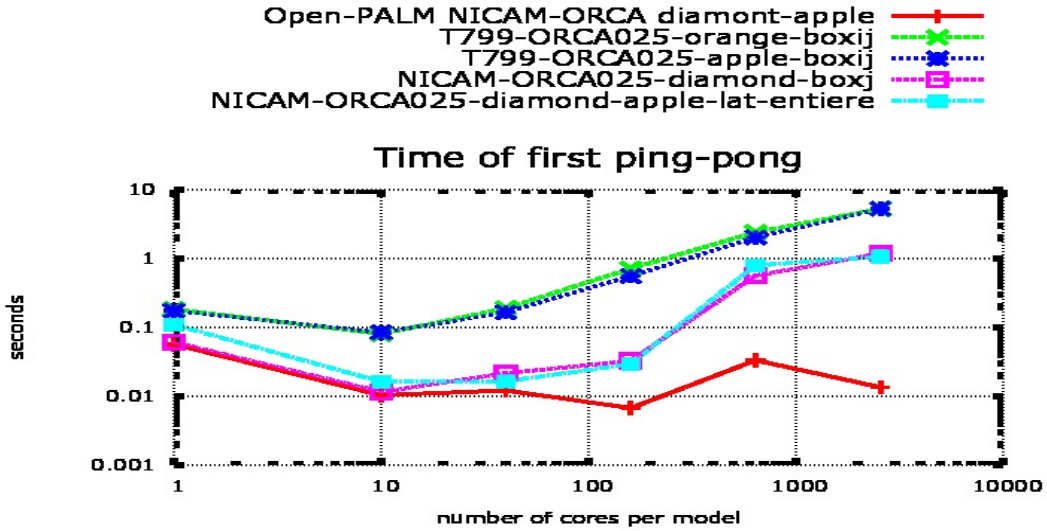


Figure 26. La comparaison du temps du 1er ping-pong par rapport aux différents modèles

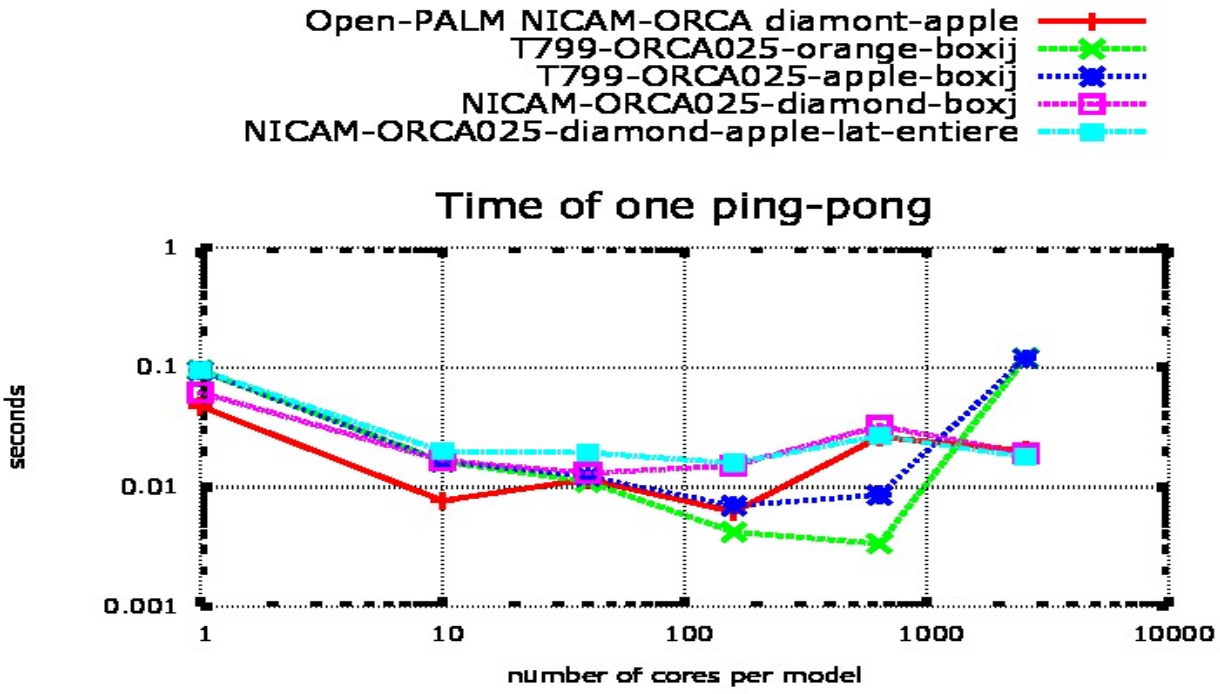


Figure 27. La comparaison du temps moyen d'un ping-pong quelconque par rapport aux différents modèles

La première conclusion est qu'OASIS3-MCT gère bien les grilles non-structurées de type icosaèdres car les résultats sont tout à fait raisonnables : le temps d'un ping-pong quelconque est inférieur à 0.1 sec jusqu'à 2560 cœurs par modèle, ce qui est négligeable par rapport au temps totale passé dans un vrai modèle. Les résultats avec OpenPALM sont aussi très bons, mais cela est moins surprenant car CWIPI a été conçu pour des grilles non-struturées (ce qui n'est pas le cas pour MCT).

On remarque que le premier ping-pong avec OASIS3-MCT est sensiblement plus couteux que les suivants (voir figure 26), ce qui n'est pas le cas pour OpenPALM. Cela peut être expliqué comme suit :

-Dans les modèles interfacés par OASIS3-MCT, les premières communications MPI globales s'effectuent au niveau du premier ping-pong, et comme MPI prend du temps pour initialiser de nouvelles communications, cela ralentit le premier échange. Ceci a été confirmé par une

étude de performance détaillée effectuée par Chris Goodyer de NAG dans le cadre du projet EXA2CT (C. Goodyer, 2015, comm. pers.)

-Concernant les modèles interfacés par OpenPALM, les premières communications MPI globales sont faites avant le premier échange, don le premier ping-pong se comporter comme les suivants.

Avant de passer à l'explication des performances obtenues pour un ping-pong quelconque, on propose une relation entre l'augmentation des communications et le type de partitionnement : moins les partitions source et cible ne présentent de correspondance géographique, plus on aura un schéma de communications complexe et donc couteux,

Les figures 26 et 27 montrent les comportements des schémas de communications selon les types de partitionnements utilisés pour les partitions sources et cibles.

Dans le cas de la figure 28, on a peu de correspondance géographique entre la partition de NICAM en diamants et sous-diamants et la partition APPLE de la grille ORCA.

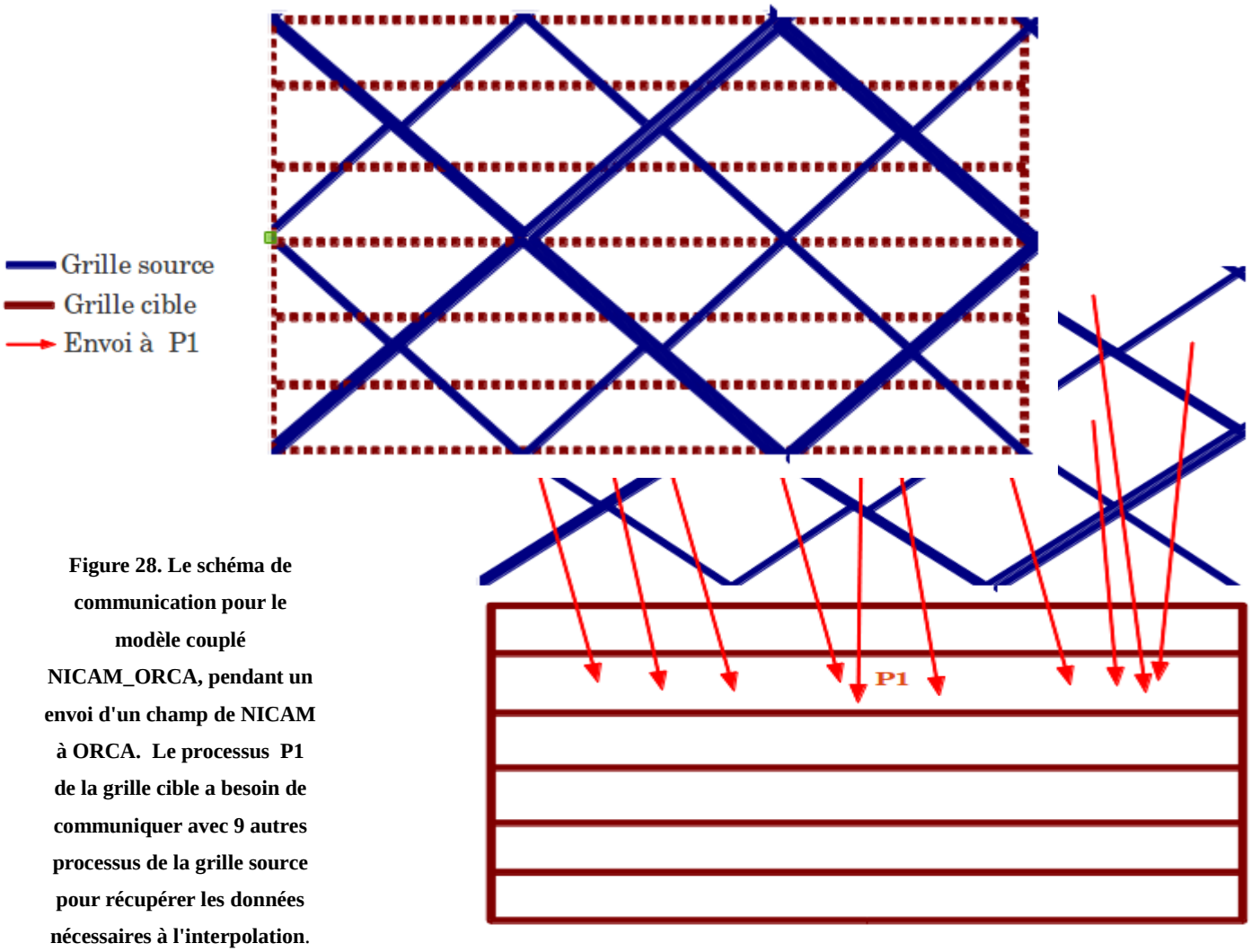


Figure 28. Le schéma de communication pour le modèle couplé NICAM_ORCA, pendant un envoi d'un champ de NICAM à ORCA. Le processus P1 de la grille cible a besoin de communiquer avec 9 autres processus de la grille source pour récupérer les données nécessaires à l'interpolation.

Dans le cas de la figure 29 qui montre le schéma de communication du modèle jouet T799_ORCA, le partitionnement utilisé pour la grille T799, sous forme de rectangles, correspond mieux au partitionnement de la grille ORCA, aussi sous forme de rectangles, que celui de la grille NICAM.

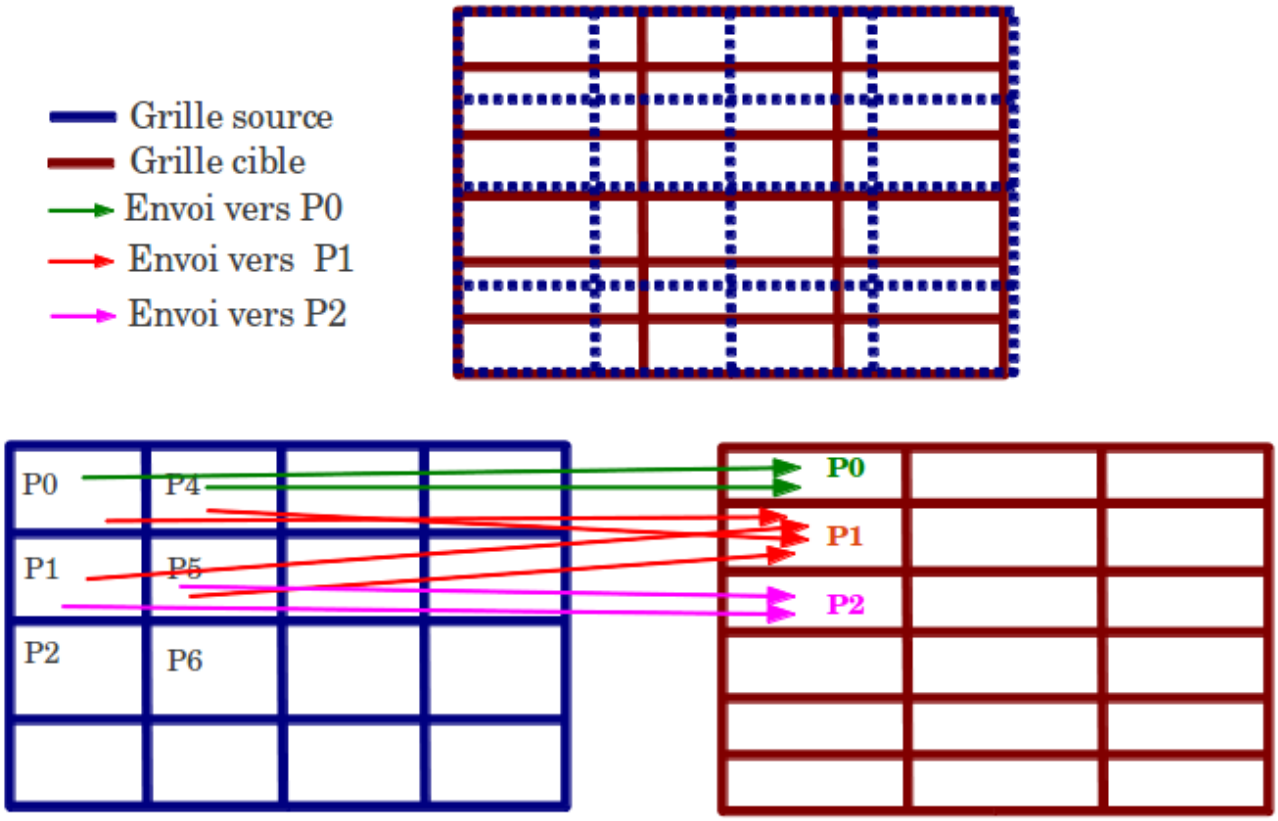
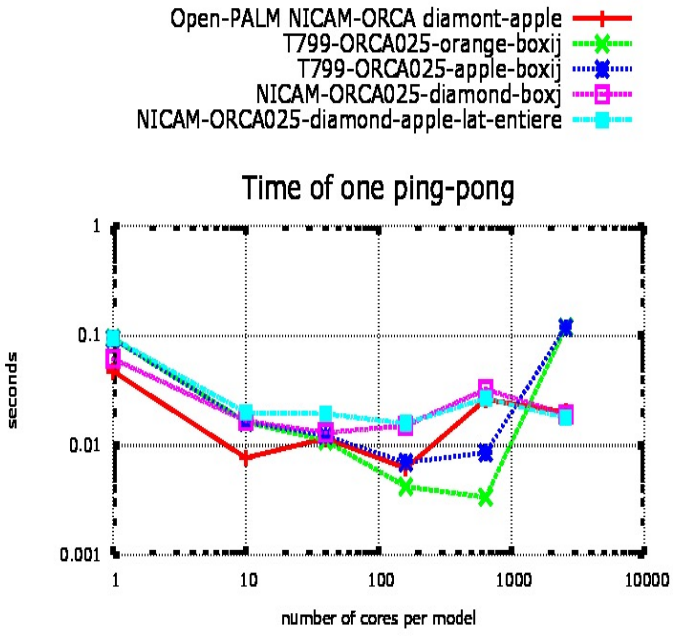


Figure 29. Le schémas de communications pour le modèle couplé T799_ORCA , pendant un envoi d'un champ de T799 à ORCA . Un processus de la grille cible a besoin de communiquer avec au plus 4 processus de la grille source afin de récupérer les données nécessaires pour l'interpolation.

Revenant aux performances d'un ping-pong quelconque, la figure 30 montre à gauche les mêmes courbes présentées ci-dessus avec à droite les correspondances géographiques qui peuvent y avoir pour les différents modèles.



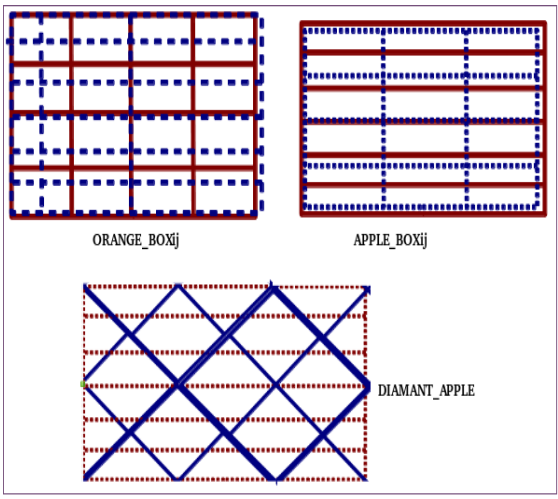


Figure 30. La relation entre les performances du ping-pong et la correspondance géographique des partitions

On remarque que le temps moyen d'un ping-pong quelconque pour le modèle NICAM-ORCA est moins performant que celui de T799-ORCA

sauf pour le dernier point (2560 proc par modèle). Ainsi, jusqu'à 640 processus par modèle, notre hypothèse reliant les performances aux correspondances géographiques des partitions se trouve vérifiée; le modèle T799-ORCA correspondant aux partitionnements ORANGE pour la T799 et BOXij pour ORCA (ORANGE_BOXij) présente la meilleure correspondance géographique, et les meilleures performances, suivi par le même modèle avec les partitionnements Apple-BOXij, suivi par le modèle NICAM_ORCA avec les deux coupleurs. Ce dernier modèle jouet présentant le moins de correspondance géographique entre les partitions source et cible.

Nous n'expliquons pas pour l'instant la remontée des deux courbes du modèle T799-ORCA avec OASIS à 2560 processus. Pour répondre à cette question, il faudrait faire des tests à nombre plus élevé de cœurs et sur d'autres super-calculateurs pour voir si ce comportement est robuste ou pas.

7. Conclusion

Un modèle jouet NICAM_ORCA couplant deux codes, le premier utilisant la grille atmosphérique du modèle NICAM de type icosaèdre non structuré et le deuxième utilisant une grille océanique logiquement rectangle appelée ORCA, a été développé avec les coupleurs OASIS3-MCT et OpenPALM. Des problèmes ont été rencontrés pendant l'interfaçage du code utilisant la grille icosaèdre avec les deux coupleurs à cause de sa structure complexe et de l'absence de certaines informations concernant cette grille.

Deux problèmes ont été rencontrés pendant l'interfaçage de ce code avec le coupleur OASIS3-MCT liés au fait que pour respecter le partitionnement du maillage réellement utilisé dans le modèle NICAM, il fallait pour chaque niveau de partitionnement soit adapter le codage de la déclaration des partitions dans le code, soit faire varier l'indexage global de la grille de façon à suivre séquentiellement les partitions les unes après les autres. Cette dernière solution a été choisie et il a donc fallu réorganiser pour chaque niveau de partitionnement, les vecteurs de longitudes, latitudes et masques décrivant les points de la grille dans les fichiers d'entrée du coupleur afin de respecter l'indexage propre à chaque niveau de partitionnement.

Le deuxième problème était lié au fait qu'OASIS3-MCT effectue en séquentiel la génération des fichiers de poids et d'adresses nécessaires pour l'interpolation des champs de couplage, ce qui peut prendre des heures de calcul. Plutôt que de laisser le coupleur regénérer les fichiers de

poids et d'adresses pour chaque niveau de partition, la solution adoptée a été de réorganiser les poids et les adresses de ces fichiers afin de respecter l'indexage propre à chaque niveau de décompositions, à partir d'un résultat obtenu pour un certain niveau de partitionnement.

Deux difficultés particulières ont été rencontrées lors l'interfaçage du même code avec le coupleur OpenPALM. La première était liée à l'obligation de fournir au coupleur une grille sous forme de mailles plutôt que comme un ensemble de points. Pour résoudre ce problème, il a été nécessaire de créer un maillage représentant la grille à partir de l'ensemble de ses points.
La deuxième difficulté est venue du fait que la grille NICAM est associée à un masque qui sert à différencier les points continentaux des points océaniques. Hors cette information ne peut être explicitement transmise et considérée par OpenPALM. Il a donc fallu régénérer le masque des mailles à partir de celui des coins des mailles et filtrer le maillage, de façon à ne prendre en considération que les mailles non masquées.
L'analyse des performances des deux coupleurs a été faite à partir des mesures du temps utilisé pour effectuer des échanges de type ping-pong entre les deux codes (soit un aller-retour d'un champ de couplage). La première conclusion est qu'OASIS3-MCT gère bien les grilles de type icosaèdre car les résultats sont tout à fait raisonnables, le temps d'un ping-pong étant inférieur à 0.1 sec jusqu'à 1000 cœurs. Les résultats avec Open-PALM sont aussi très bons, mais cela est moins surprenant car la librairie d'interpolation CWIPI a été conçue pour des grilles non-struturées. Finalement, en analysant les résultats, on a émis l'hypothèse que la performance des échanges est liée à la correspondance géographique des partitions des codes source et cible: moins il y a de correspondance, plus on aura un schéma de communications complexe et moins les échanges seront performants. Pour valider complètement notre hypothèse, il faudrait faire des tests à nombre plus élevé de cœurs sur d'autres super-calculateurs.

8. Annexes

Annexe A1 : Algorithme de Découpage-étape 1

Entrées :

width : largeur de la partition avant le découpage.
width_new : largeur d'une sous partition obtenu après le découpage.
nb_part : nombre total de partition.
shift : décalage de width*width/2 .
shift2 : décalage de k*width*width, où $0 \leq k < nb\_part$ est l'indice de la partition.
lon, lat, et mask : 3 tableaux 1d qui représentent respectivement les latitudes, les longitudes, et les masques.
lon2, lat2, et mask2 : 3 tableaux intermédiaires.

Sorties : lon, lat, mask (modifiés)

On remarque que les points d'une même sous partition se déplacent de la même manière (Figure 16-c), ce qui permet de les traiter dans la même boucle.

le traitement se compose de 4 boucles, une boucle pour chaque 1/4 de la partition, comme illustré dans la figure 31.

```
lon2=lon
lat2=lat
mask2=mask
```

```
do i=0,width_new-1
  do j=width_new*i+shift2,width_new*(i+1)-1+shift2
    lon(j)=lon2(j+(width_new*i))
    lat(j)=lat2(j+(width_new*i))
    mask(j)=mask2(j+(width_new*i))
  end do
end do
```

```
do i=0,width_new-1
  do j=shift+width_new*i+shift2,shift+width_new*(i+1)-1+shift2
    lon(j)=lon2(j+(width_new*i))
    lat(j)=lat2(j+(width_new*i))
    mask(j)=mask2(j+(width_new*i))
  end do
end do
```

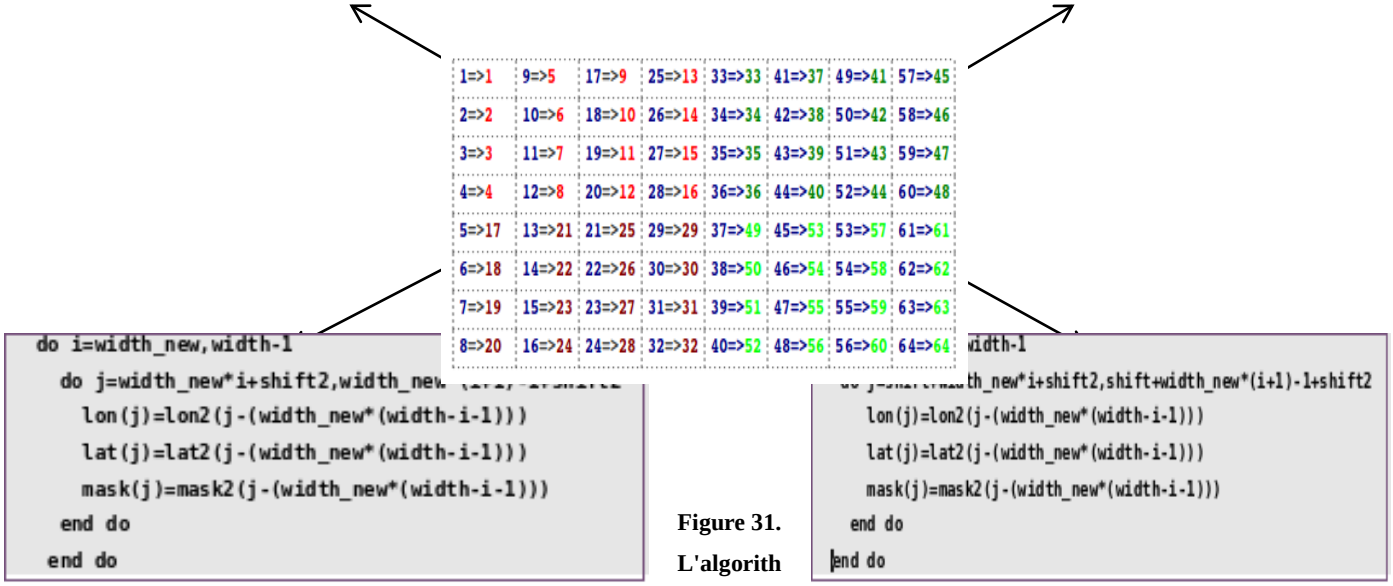


Figure 31. L'algorithme de découpage d'une partition en 4 sous partitions

Annexe A2 . Algorithme De Découpage-étape 2

Partant des sorties obtenues à étape 1,

Entrés :

les sorties de l'étape 1 : lon, lat, et mask

width_diamant : largeur du diamant en nombre de partition (après le découpage)

size_diamant=width_diamant*width_diamant : la taille du diamant en nombre de partition

width : largeur de la partition (sous partition obtenu après étape 1)

size_part=width*width : la taille d'une partition

Sorties : lon, lat, et mask (modifiés)

```
lon2(:)=lon(:)
lat2(:)=lat(:)
mask2(:)=mask(:)
!pour chaque diamant
    do p=0,9
      M=0
      do L=p*size_diamant,width_diamant*(width_diamant-1)+size_diamant*p,width_diamant*2
        M=L+width_diamant
        k=L
      !La 1ère boucle pour traiter les partitions situées dans les colonnes paires
       do j=L,L+width_diamant-2,2
        ! La k ème partition devient la j ème
        lon(j*size_part:(j+1)*size_part-1)=lon2(k*size_part:(k+1)*size_part-1)
        lon((j+1)*size_part:(j+2)*size_part-1)=lon2((k+1)*size_part:(k+2)*size_part-1)

        lat(j*size_part:(j+1)*size_part-1)=lat2(k*size_part:(k+1)*size_part-1)
```

```
    lat((j+1)*size_part:(j+2)*size_part-1)=lat2((k+1)*size_part:(k+2)*size_part-1)

    mask(j*size_part:(j+1)*size_part-1)=mask2(k*size_part:(k+1)*size_part-1)
    mask((j+1)*size_part:(j+2)*size_part-1)=mask2((k+1)*size_part:(k+2)*size_part-1)

    k=k+4
  end do

  k=2+L
 ! La 2ème boucle pour traiter les partitions situées dans les colonnes impaires
  do j=M,M+width_diamant-2,2
   ! La k ème partition devient la j ème
   lon(j*size_part:(j+1)*size_part-1)=lon2(k*size_part:(k+1)*size_part-1)
   lon((j+1)*size_part:(j+2)*size_part-1)=lon2((k+1)*size_part:(k+2)*size_part-1)

   lat(j*size_part:(j+1)*size_part-1)=lat2(k*size_part:(k+1)*size_part-1)
   lat((j+1)*size_part:(j+2)*size_part-1)=lat2((k+1)*size_part:(k+2)*size_part-1)

   mask(j*size_part:(j+1)*size_part-1)=mask2(k*size_part:(k+1)*size_part-1)
   mask((j+1)*size_part:(j+2)*size_part-1)=mask2((k+1)*size_part:(k+2)*size_part-1)

  k=k+4
  end do

 end do
end do
```

Annexe A3 : Algorithme De Regroupement-étape 1

Entrés :

lon, lat, et mask : respectivement longitude, latitude, et mask

width_diamant : largeur du diamant en nombre de partition

size_diamant=width_diamant*width_diamant : la taille du diamant en nombre de partition

width : largeur de la partition

size_part=width*width : la taille d'une partition

Sorties : lon, lat, et mask (modifiés)

```
lon2(:)=lon(:)
lat2(:)=lat(:)
mask2(:)=mask(:)
!Pour chaque diamant
do p=0,9
 size_part=width*width
  M=0
 do L=p*size_diamant,width_diamant*(width_diamant-1)+size_diamant*p,width_diamant*2
   M=L+width_diamant
   k=L
  !traiter les partitions situées dans les colonnes paires
```

```
  do j=L,L+width_diamant-2,2

   lon(k*size_part:(k+1)*size_part-1)=lon2(j*size_part:(j+1)*size_part-1)
   lon((k+1)*size_part:(k+2)*size_part-1)=lon2((j+1)*size_part:(j+2)*size_part-1)

   lat(k*size_part:(k+1)*size_part-1)=lat2(j*size_part:(j+1)*size_part-1)
   lat((k+1)*size_part:(k+2)*size_part-1)=lat2((j+1)*size_part:(j+2)*size_part-1)


   mask(k*size_part:(k+1)*size_part-1)=mask2(j*size_part:(j+1)*size_part-1)
   mask((k+1)*size_part:(k+2)*size_part-1)=mask2((j+1)*size_part:(j+2)*size_part-1)
  k=k+4

   end do

   k=2+L
  ! traiter les partitions situées dans les colonnes impaires
  do j=M,M+width_diamant-2,2

   lon(k*size_part:(k+1)*size_part-1)=lon2(j*size_part:(j+1)*size_part-1)
   lon((k+1)*size_part:(k+2)*size_part-1)=lon2((j+1)*size_part:(j+2)*size_part-1)

   lat(k*size_part:(k+1)*size_part-1)=lat2(j*size_part:(j+1)*size_part-1)
   lat((k+1)*size_part:(k+2)*size_part-1)=lat2((j+1)*size_part:(j+2)*size_part-1)


    mask(k*size_part:(k+1)*size_part-1)=mask2(j*size_part:(j+1)*size_part-1)
    mask((k+1)*size_part:(k+2)*size_part-1)=mask2((j+1)*size_part:(j+2)*size_part-1)

  k=k+4
   end do

  end do
 end do
```

Annexe A4 : Algorithme De Regroupement-étape 2

Entrées :

lon, lat, et mask obtenues à l'étape 1 de regroupement

width : largeur de la partition (grandes partitions obtenues) après l'étape 1

nb_part : nombre de partition de la grille (après étape 1)

width_new=width/2

shift=width*width/2

Sorties : lon, lat, et mask (modifiés)

```
lon2(:)=lon(:)
lat2(:)=lat(:)
mask2(:)=mask(:)

!Pour chaque partition
do k=0,nb_part-1

    shift2=k*width*width
 !Traitement du 1 er ¼ de la partition
do i=0,width_new-1
        do j=width_new*i+shift2,width_new*(i+1)-1+shift2
         lon(j+(width_new*i))=lon2(j)
         lat(j+(width_new*i))=lat2(j)
         mask(j+(width_new*i))=mask2(j)
        end do
       end do
   !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
    !Traitement du 2ème ¼ de la partition
      do i=width_new,width-1
        do j=width_new*i+shift2,width_new*(i+1)-1+shift2
         lon(j-(width_new*(width-i-1)))=lon2(j)
         lat(j-(width_new*(width-i-1)))=lat2(j)
                  mask(j-(width_new*(width-i-1)))=mask2(j)
        end do
       end do
   !-------------------------------!
    !Traitement du 3 ème ¼ de la partition
     do i=0,width_new-1
       do j=shift+width_new*i+shift2,shift+width_new*(i+1)-1+shift2
        lon(j+(width_new*i))=lon2(j)
        lat(j+(width_new*i))=lat2(j)
        mask(j+(width_new*i))=mask2(j)
        end do
       end do
   !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
  !Traitement du 4 ème ¼ de la partition
  do i=width_new,width-1
       do j=shift+width_new*i+shift2,shift+width_new*(i+1)-1+shift2
         lon(j-(width_new*(width-i-1)))=lon2(j)
         lat(j-(width_new*(width-i-1)))=lat2(j)
                  mask(j-(width_new*(width-i-1)))=mask2(j)
        end do
       end do
   end do
```

Annexe B : Algorithme de génération des fichiers de poids et d'adresses pour les différentes décompositions :

On prend le cas où NICAM est la grille source, et qu'on veut générer les résultats de localisation pour un niveau de partitionnement

plus élevé que celui pour lequel ces résultats sont calculés.
Entrées :
src : tableau entier contenent les adresses des points sources
src_lon, src_lat : deux tableaux réels contenant respectivement les longitudes et les latitudes des points de la grille source
Sorties : src (modifié)

considérant :
indice, et indice2 : deux tableaux entiers de taille nombre de points
size_grid : la taille de la grille
size_adresse : la taille du tableau src

0- Initialiser les tableaux intermédiaires.

```
src2=src
src_lon2=src_lon
src_lat2=src_lat
```

1- Appliquer l'étape 1 de l'algorithme de découpage et stocker les nouveaux indices dans le tableau « indice » après chaque déplacement d'un point
Exemple :

```
src_lon(j)=src_lon2(j-(width_new*(width-i-1)))
indice(j-(width_new*(width-i-1)))=j
```

2- Appliquer l'étape 2 de l'algorithme de découpage et stocker les nouveaux indices dans le tableau « indice2 » après chaque déplacement d'une partition.
Exemple :

```
src_lon((j+1)*size_part:(j+2)*size_part-1)=src_lon2((k+1)*size_part:(k+2)*size_part-1)
indice2((k+1)*size_part:(k+2)*size_part-1)=((j+1)*size_part :(j+2)*size_part-1)
```

3- Mettre à jour le tableau indice (puisque les points ont été déplacés une 2ème fois).

```
do i=0,size_grid-1
  indice(i)=indice2(indice(i))
  indice(i)=indice(i)+1 //car les adresses vont de 1 à size_grid
end do
```

4- Mettre à jour le tableau src.

```
do j=0,size_address-1
     src(j)=indice(src2(j)-1)
end do
```

Annexe C : Algorithme de création du maillage

Partie 1. Connecter les points de la première catégorie :

```
SUBROUTINE create_mesh_init(lon,lat,width_partition,n_vertex,n_elements,coordinates,connectivity,&
start_lon,end_lon,start_conn,end_conn, start_cord,end_cord)

 DOUBLE PRECISION,DIMENSION( :)              :: lon,lat    ! Les longitudes et les latitudes
```

```
 INTEGER                          :: n_vertex                  !  Le nombre de coins
 INTEGER                          :: n_elements                ! Le nombre de mailles
 DOUBLE PRECISION,DIMENSION( :)   :: coordinates     ! Le tableau de coordonnées créé pour CWIPI
 INTEGER,DIMENSION( :)            :: connectivity              ! Le tableau des connectivités créé pour CWIPI
 INTEGER                          :: width_partition           ! Largeur d'un partition
  INTEGER                         ::  i,j
 DOUBLE PRECISION                         :: pi, conv_deg_rad,phi,teta   ! Variable necessaires pour passer des              !
coordonnées sphériques aux coordonnées cartésiennes
 ! !début et la fin d'un diamant, comme la routine est appelée pour un seul diamant
INTEGER                       :: start_lon,end_lon,&
                                 start_conn,end_conn,&
                                 start_cord,end_cord
 !------------------!

 !Initialisations
 pi=acos(-1.)
  !calculer les connectivités!
 j=start_conn
 do i=start_lon,end_lon-(width_partition+1)
 if (mod(i,width_partition) .ne. 0) then
 connectivity(j)=i
  connectivity(j+1)=i+width_partition
  connectivity(j+2)=i+width_partition+1
  connectivity(j+3)=i+1
 j=j+4
  end if
 end do
!Calculer les coordonnées
 conv_deg_rad=pi/180. !pour convertir du degré en radian
 j=start_cord
 do i=start_lon,end_lon
  teta = lon(i)*conv_deg_rad
  phi = lat(i)*conv_deg_rad
  coordinates(j)  =COS(phi)*COS(teta)
  coordinates(j+1)=COS(phi)*SIN(teta)
  coordinates(j+2)=SIN(phi)
  j=j+3
 end do

END SUBROUTINE create_mesh_init
```

Partie 2. Créer les mailles reliant entre des points de la 2ème catégorie (créer des mailles en remplissant les vides entre les grands diamants):

```
SUBROUTINE fill_border_mesh(connectivity,connectivity_index,width_partition,&
elements_trgle,elements_pentgn)

use palmlib
```

```
IMPLICIT NONE
!-------------------!
 !Les paramètres
 INTEGER,DIMENSION( :)          :: connectivity, connectivity_index  ! Les connectivités et les index !     !de connectivités
définit pour CWIPI
 INTEGER                        :: width_partition ! Largeur de la partition
!-------------------!
 INTEGER                        :: neighbor_r,neighbor_s  ! respectivement, diamant voisin droit et gauche
 INTEGER                        :: i,k,j,index
 DOUBLE PRECISION,ALLOCATABLE,DIMENSION(:) :: edge_r,edge_s ! l'ensemble des points situé   ! dans les bords du
diamant actuel respectivement droite et gauche
DOUBLE PRECISION,ALLOCATABLE,DIMENSION(:)  edge_neighbor_r,edge_neighbor_s  ! l'ensemble    !des points
situé  dans les bords  des diamants voisin respectivement droite et gauche, et qui sont          ! voisins géographiquement aux bords
du diamant actuel
 INTEGER                        :: elements_trgle,elements_pentgn ! Le nombre de mailles de types                    !triangle/pentagone
 !---------------!
  !les allocations
  allocate(edge_r(width_partition))
  allocate(edge_s(width_partition))
  allocate(edge_neighbor_r(width_partition))
  allocate(edge_neighbor_s(width_partition))
 !---------------!

 !pour les 5 1ers diamants
 index=(width_partition-1)*(width_partition-1)*10*4+1!l'indice de début de l'ajout des
 !mailles
 do i=0,4
  call neighbor_diamand(i,neighbor_r,neighbor_s)

  call get_edge_r(i,edge_r,width_partition)
  call get_edge_s(i,edge_s,width_partition)

  call  get_first_line(neighbor_r,edge_neighbor_r,width_partition)
  call  get_first_line(neighbor_s,edge_neighbor_s,width_partition)
  !ajouter les mailles à la fin du tableau connectivité

 do k=1,width_partition-1
    connectivity(index)  =edge_s(k)
    connectivity(index+1)=edge_neighbor_s(k)
    connectivity(index+2)=edge_neighbor_s(k+1)
    connectivity(index+3)=edge_s(k+1)
  index=index+4
  end do
do k=1,width_partition-1
    connectivity(index)  =edge_r(k)
    connectivity(index+1)=edge_neighbor_r(width_partition-k+1)
    connectivity(index+2)=edge_neighbor_r(width_partition-k)
```

```
   connectivity(index+3)=edge_r(k+1)
 index=index+4
 end do

 end do
!pour les 5 derniers diamants
 do i=5,9
 call neighbor_diamand(i,neighbor_r,neighbor_s)

 call get_edge_r(i,edge_r,width_partition)
 call get_edge_s(i,edge_s,width_partition)

 call  get_first_column(neighbor_r,edge_neighbor_r,width_partition)
 call  get_first_column(neighbor_s,edge_neighbor_s,width_partition)
!ajouter les mailles à la fin du tableau connectivité

 do k=1,width_partition-1
   connectivity(index)  =edge_s(k)
   connectivity(index+1)=edge_neighbor_s(width_partition-k+1)
   connectivity(index+2)=edge_neighbor_s(width_partition-k)
   connectivity(index+3)=edge_s(k+1)
 index=index+4
 end do
 do k=1,width_partition-1
!   print*,'index=',index
   connectivity(index)  =edge_r(k)
   connectivity(index+1)=edge_neighbor_r(k)
   connectivity(index+2)=edge_neighbor_r(k+1)
   connectivity(index+3)=edge_r(k+1)
 index=index+4
 end do

 end do
! ajouter les triangles et les pantagones qui manquent entre
! les sommets des grands diamands
!cette fonction crée les mailles reliant les coins des diamants (triangles, pentagones)
call add_trgle_pentgn(connectivity,connectivity_index,elements_trgle,&
          elements_pentgn,index,width_partition)  ! Index indique l'indice de ladernière maille ajoutée
deallocate(edge_r)
 deallocate(edge_s)
 deallocate(edge_neighbor_r)
 deallocate(edge_neighbor_s)
END SUBROUTINE fill_border_mesh
```

9. Références

[1] http://kiwi.atmos.colostate.edu/pubs/CISE.pdf

[2] http://kiwi.atmos.colostate.edu/group/dave/at604pdf/Chapter_12.pdf

[3] http://fr.wikipedia.org/wiki/Diagramme_de_Vorono%C3%AF

[4] http://fr.wikipedia.org/wiki/Grille_gaussienne

[5] http://www.cerfacs.fr/globc/PALM_WEB/

[6] https://verc.enes.org/oasis

[7] (DAVID A. RANDALL et al, CLIMATE MODELING WITH SPHERICAL GEODESIC GRIDS,2003)

10. Tables des figures