Etude préliminaire de faisabilité pour la fusion des coupleurs OASIS et PALM

Rapport technique CERFACS TR-CMGC-09-90

Auteurs : Andrea Piacentini, Sophie Valcke, Thierry Morel

Date : 28 septembre 2009

Nombre de pages : 16

Contact : piacentini.palm@gmail.com , valcke@cerfacs.fr , morel@cerfacs.fr

Table des matières

1 Introduction

Le coupleur OASIS3 est actuellement l'outil de référence pour l'implémentation de modèles couplés en climatologie. Ses points de force sont la portabilité, la mise à disposition d'outils d'interpolation et de traitement algébrique ciblés pour le couplage géophysique et, très important, son utilisation dans une vaste communauté scientifique. Des faiblesses au niveau des performances et de la

simplicité d'utilisation, surtout dans le cas de couplages avec un nombre élevés de modèles et de champs échangés, ont déterminé l'exploration de nouvelles approches pour les développements ultérieurs d'OASIS. La version OASIS4, par une réécriture du noyau de communication et du *transformer*, cherche à répondre à l'exigence de performances accrues sur machines massivement parallèles. La version actuelle d'OASIS4 ne dispose pas encore de toutes les fonctions d'OASIS3 et n'a pas encore remplacé la version 3 dans la communauté climatique.

Le coupleur PALM s'est bien affirmé dans le domaine des couplages dynamiques où les modalités d'exécution des codes ou de l'algorithme de couplage peuvent évoluer au cours de la simulation. Il est aussi utilisé pour l'assemblage d'algorithmes complexes à partir d'unités de calcul plus simples, en particulier pour l'assimilation de données. Ses points de force sont la flexibilité, la convivialité de son interface graphique et la possibilité de suivre en temps réel le déroulement d'un couplage avec des outils de monitorage et de débogage.

La force de travail qui peut être consacrée au développement d'outils dans l'équipe Global Change du CERFACS risque de ne pas pouvoir assurer à moyen et long terme le développement en parallèle de deux outils aussi perfectionnés que PALM et OASIS. Les évolutions récentes des accords de développement d'OASIS avec NEC Laboratories Europe rendent cette situation encore plus délicate. Il est donc important de comparer les principes de fonctionnement des versions actuelles des deux coupleurs, leurs similarités et différences, de façon à estimer la faisabilité de la fusion des deux outils dans un seul produit.

Si la piste de la fusion est retenue, il faudra ensuite affiner l'estimation de la force de travail nécessaire pour mener à bien la fusion dans des temps compatibles avec les prévisions des besoins des utilisateurs.

Cette étude préliminaire vise à établir si la modularité et la flexibilité de PALM peuvent suffire pour assurer toutes les fonctions d'OASIS3 et si la gestion du parallélisme dans PALM peut évoluer de sorte à garantir le niveau de performances d'OASIS4.

L'analyse est effectuée à partir de la documentation d'OASIS3 (version prism_2-5 rapport technique CERFACS TR/CMGC/06/73, septembre 2006) et d'OASIS4 (version OASIS4_0_2 rapport technique CERFACS TR/CMGC/06/74, août 2006) et de discussions approfondies avec les responsables des développements des deux coupleurs.

1.1Organisation du document

Un coupleur flexible et performant est nécessairement un code complexe. Il n'est pas facile de donner une organisation linéaire à ce document.

Le plan de ce document est fonctionnel : les coupleurs sont analysés par tâches et composants, mais, puisque les différents aspect sont étroitement liés, les renvois internes sont nombreux.

Les trois coupleurs suivent la même philosophie : respecter le plus possible l'intégrité et l'indépendance des codes à coupler et fournir des outils pour le pilotage de l'application couplée, pour les échanges des données (basés dans

tous les cas sur le protocole *end-point*) et pour les traitements algébriques effectués sur les champs échangés. Le respect de la convivialité d'utilisation est aussi un critère important de conception.

Nous pouvons donc reconnaître les composants logiques suivants

•Le(s) lancement(s) des codes à coupler et leur pilotage

•Les communications et leur monitorage

•Les transformations algébriques et leur pilotage

•L'interfaçage entre codes et algorithme de couplage

•L'interfaçage homme machine

Le plan du document suit cette décomposition avec une section dédiée à l'analyse de chaque composant et un résumé des discussions d'approfondissement sur chaque point critique.

L'analyse comporte un aspect de comparaison fonctionnelle, qui vise à assurer que les structures logiques et informatiques de PALM puissent répondre aux exigences d'un couplage climatologique sans demander aux utilisateurs un changement de modalité de travail trop important. Puisque OASIS4 n'assure pas encore toutes les fonctions d'OASIS3, nous ferons souvent référence à la documentation d'OASIS3. Par contre, en ce qui concerne les performances, nous prendrons en compte les améliorations et les optimisations apportées dans OASIS4, ou, du moins, prévues dans sa définition.

2 Le(s) lancement(s) des codes à coupler et leur pilotage

La principale distinction entre OASIS et PALM, que l'on a maintes fois soulignée, est que le premier est un coupleur statique, tandis que le deuxième est dynamique. Si l'on peut rapidement dire qu'un couplage statique (où les caractéristiques du couplage -composants, champs, fréquence d'échange, etc.- ne changent pas au cours de la simulation) peut toujours être vu comme un cas particulier de couplage dynamique, il faut, en réalité, évaluer les conséquences sur l'implémentation et sur les performances de ce type de couplage.

Puisque la modalité de lancement des codes va largement déterminer aussi le protocole de communication et la répartition des informations sur l'avancement du couplage, cet aspect a une grande influence sur les autres points de l'analyse.

2.1Couplage statique

Dans un couplage statique, le lancement d'un code est effectué au début de la simulation couplée et ne répond à aucun critère conditionnel évalué *run-time*.

Le coupleur doit donc, en début de simulation :

•garantir que chaque unité dispose des ressources nécessaires à son exécution

•fournir à chaque code parallèle un contexte d'exécution et communication

privé

•adresser de façon univoque chaque processus de chaque code dans le contexte de communication global

•répliquer cette information (ou du moins le sous-ensemble nécessaire aux communications prévues) auprès de tous les codes impliqués dans des échanges de données de couplage

•s'assurer que tous les codes s'échangent toutes les informations nécessaires au déroulement des communications et des transformations (distributions parallèles, renseignements géographiques sur les grilles, etc…)

Puisque toutes ces informations n'évoluent pas en cours d'exécution, une fois que l'information est répliquée sur tous les codes à coupler, le coupleur n'a plus aucune tâche de pilotage à effectuer. On peut ainsi dire que le coupleur joue seulement un rôle de *launcher.* Ensuite il peut mettre ses ressources à la disposition des autres tâches. En particulier, dans OASIS le processeur qui a agi comme lanceur s'occupera ensuite des traitements algébriques (tout seul dans le cas d'OASIS3, avec d'autres processeurs dans le cas d'OASIS4). D'un point de vue logique on peut voir l'ensemble des processeurs qui s'occupent des traitements algébriques comme un code à coupler parmi les autres. Il sera le dernier lancé par le coupleur au moment même où il arrête d'être lanceur. Pendant le couplage, les codes disposeront dans leur mémoire locale (cf. sections suivantes) de toutes les informations nécessaires à l'exécution des traitements algébriques et au déroulement des communications.

2.2Couplage dynamique

L'approche est complètement différente lors d'un couplage dynamique : le coupleur doit toujours fournir aux codes les ressources nécessaires et les contextes de communication privé et avec les autres codes, mais il n'est pas possible de prévoir à l'avance comment et combien de fois un code va se déployer. De plus, la flexibilité des algorithmes de couplage (avec boucles itératives, exécutions conditionnelles, synchronisation réduite à l'essentiel) ne permet pas de prévoir l'état d'exécution des unités et l'ensemble des objets produits ou requis à un instant donné. Dans ce cas il est plus efficace de garder un coupleur actif pendant toute la durée du couplage, qui décide si, quand et comment lancer les codes et qui dispose à tout instant de l'image à jour de l'état du couplage dans tous ses aspects. Dans ce sens le coupleur est un vrai pilote ou *driver*. Les codes à coupler ne gardent aucune information statique dans leur mémoire locale : ils adressent au coupleur des notifications de changement d'état et des requêtes d'informations. Cette approche impose beaucoup de petites communications entre le coupleur et les codes, mais, avec une latence MPI acceptable, elle est plus performante qu'une implémentation où tous les codes cherchent à mettre à jour leur image locale de l'état du système couplé. Cette modalité, en outre, permet un suivi fin de la simulation, ce qui peut être très utile dans les phases de mise au point et débogage ou pour un monitorage en temps réel de l'exécution.

2.3Fonctions que PALM peut assurer

PALM peut fonctionner en modalité MPI1 où tous les codes démarrent en même temps que le coupleur et partagent un communicateur MPI_COMM_WORLD global, ou en modalité MPI2 avec lancement dynamique des exécutables par MPI_COMM_SPAWN. Une optimisation consiste à lancer des tâches qui se suivent, si leur codage est compatible, sous forme de subroutines d'un seul exécutable (dit *block*). Pour cette raison, on recommande dans les codes couplés de ne pas utiliser, pour la parallélisation interne de chaque exécutable le communicateur MPI_COMM_WORLD, mais un communicateur local créé par PALM et nommé systématiquement PL_COMM_EXEC (logiquement la valeur numérique correspondante à PL_COMM_EXEC diffère dans chaque exécutable). La variable est définie dans le module (ou fichier inclus) palmlib et est initialisée au démarrage. Aucun appel spécifique n'est donc nécessaire dans la partie utilisateur du code, si ce n'est la commande nécessaire pour remplacer le MPI_COMM_WORD local par PL_COMM_EXEC. (N.B. : une bonne pratique de programmation MPI dans des codes avec vocation de couplage ou dans des bibliothèques consiste à utiliser un communicateur monde privé; on utilise, au début du code, la primitive MPI_Comm_dup pour associer le communicateur privé au communicateur monde actuel MPI_COMM_WORD si le code est stand-alone ou PL_COMM_EXEC de PALM en cas de couplage).

Puisque le contexte global de communication entre codes est variable dans le temps, les communications directes entre exécutables lancés par MPI_COMM_SPAWN passent par des intercommunicateurs établis par des ports. Dans ce cas le *driver* assure la fonction de publication des ports.

2.4Différences sur lesquelles travailler

En alternative au lancement par MPI1, OASIS peut travailler en modalité MPI2 avec lancement simultané de tous les exécutables par MPI_COMM_SPAWN_MULTIPLE. Ceci a l'avantage de créer un communicateur global qui inclut tous les exécutables de façon tout à fait analogue à la modalité de fonctionnement sous MPI1 et d'éviter donc le recours au *driver* pour la gestion des ports. PALM peut être facilement complété avec le rajout de cette modalité de fonctionnement.

Il faudra donc poursuivre le travail que Th. Morel et A. Thévenin ont déjà entamé pour rendre les activités du *driver* complètement modulaires et activables selon le type de couplage. On peut imaginer que toute action que le *driver* accomplit dynamiquement ait sa contre-partie statique qui aurait une phase d'initialisation à la charge du lanceur (simplifiée et optimisée par tout ce que l'interface de définition du couplage aura pu précalculer – cf. section sur l'interface homme machine) et une modalité d'exécution pendant la simulation entièrement déportée sur la librairie d'interfaçage des unités des codes ou des unités d'algèbre (cf. section sur les transformations algébriques).

A titre d'exemple on peut considérer le mécanisme par lequel les unités repèrent l'autre extrémité d'une communication *end-point.* Chaque processus qui appartient à une unité source doit savoir si l'unité cible correspondante est active, dans ce cas si elle a déjà posté la requête pour la communication en question, comment repérer dans le contexte de communication global les

processus cibles impliqués dans la réception, ou sinon, vers quels processeurs de l'espace de stockage temporaire envoyer l'objet produit. Dans le cas d'un couplage dynamique, l'unité source interroge le *driver* pour obtenir la réponse en temps réel. Le service de « routage » est actif en permanence sur le *driver* qui doit tourner pendant toute la simulation.

Dans le cas d'un couplage statique, la table des patterns de communication peut être calculée une seule fois au début de la simulation par le *launcher* avec la collaboration des unités (ou, pour la précision, de leur librairie d'interfaçage) et les inputs de l'interface de définition de couplage. Cette table est ensuite distribuée aux interfaces des unités. Dans le cas d'une synchronisation forte, on pourra déterminer d'avance les communications directes et les communications à bufférisér. Dans ce cas, l'unité source, pour se procurer les informations utiles, n'aura plus besoin d'interpeller le *driver*, mais elle trouvera les réponses directement dans la mémoire allouée à l'interface. Toutes les structures de données propres à la communication resterons donc inchangées, mais des fonctions de service différentes s'occuperont de les remplir.

De même on peut rendre optionnels le monitorage et l'analyse des performances d'une simulation et les faire correspondre à des services du *driver*, aussi bien que le *scheduler*, le *pecker*, la gestion dynamique de l'espace de stockage temporaire, etc. Si aucun service n'est activé en temps réel, le processeur du *driver* peut être rendu disponible.

3 Les communications et leur monitorage

3.1La gestion des communications dans OASIS

Le protocole de communication dans OASIS est de type *end-point* : le correspondant n'est pas indiqué dans la ligne d'appel aux fonctions de communication, mais il est déterminé lors de la configuration du couplage. Puisque les codes peuvent être parallèles, l'échange d'un champ de couplage doit être décliné en un ensemble de communications au sens MPI entre les différents processus des codes. La phase de calcul des patterns de communication s'effectue en début de simulation. Dans le cas d'échanges impliquant des transformations (cf. section sur les transformations algébriques), le pattern doit prendre en compte les halos de calcul pour les transformations non locales. Dans OASIS4 ce calcul est effectué par les interfaces des codes dans les appels d'initialisation. L'association se base sur l'hypothèse que les champs sont associés à un repère géographique par une grille régulière ou réduite ou sous forme de nuage de points.

Les communications peuvent être:

•directes entre codes qui utilisent la même discrétisation spatiale pour des champs qui ne nécessitent pas de transformation (N.B. : les codes peuvent avoir une distribution parallèle différente)

•avec le *transformer* pour des champs à interpoler

•avec le disque pour des entrées/sorties ou pour des champs de redémarrage

Les envois ne sont jamais bloquants. Ce résultat est obtenu de façon différente pour les communications avec le *transformer* et pour les communications directes. Le *transformer* reçoit un champs source comme s'il s'agissait d'une communication directe provenant du code source, applique les transformations nécessaires et, si le champs doit être reçu plus tard, le stocke dans sa mémoire jusqu'à la requête de l'unité cible.

Les communications directes s'appuient sur la bufférisation interne de MPI (d'une forme qui dépend de l'implémentation de MPI). Pour optimiser l'envoi côté source, le champs est copié dans une variable locale de l'interface d'où il est envoyé par un Send MPI asynchrone. Après la terminaison du Send asynchrone (que l'interface peut vérifier lors de ses futures invocations) la variable locale est libérée.

Les pseudo-communications avec le disque (I/O et *restarts*) sont prises en charge par la librairie spécialisée d'I/O parallèles mpp_io.

A chaque communication est associée une étiquette temporelle, dont le but est de pouvoir différencier le nombre d'appels des primitives de communication du nombre de communications réellement effectuées. De plus, cette étiquette permet de gérer les associations temporelles avec les décalages (*lag*) aussi bien que les accumulations et les moyennes sur une période.

Les objets qui peuvent être échangés sont des tableaux d'entiers ou de réels (simple ou double précision) jusqu'à 3 dimensions (jusqu'à 2 dimensions dans OASIS3).

3.2 La gestion des communications dans PALM

Le schéma de communication dans PALM vise à couvrir un éventail plus large de types de couplage. En particulier, pour gérer les aspects dynamiques du couplage, la gestion du routage des communications est à la charge du *driver*.

Le protocole de communication est de type *end-point* avec envoi non bloquant et redistribution (*remapping*) des objets échangés entre unités avec distributions parallèles différentes. Dans le formalisme PALM il n'y a pas de repère géographique associé aux objets. Ils peuvent être de n'importe quel type informatique et avoir de 0 à 7 dimensions. Les profils des objets aux deux extrémités d'une communication doivent êtres les mêmes (tout en sachant qu'un objet peut représenter un sous-ensemble d'une variable – concept de sous-objet). Dans ce formalisme une interpolation est vue comme un couplage avec une unité de traitement ayant un objet en entré sur la grille de l'objet source et un objet en sortie sur la grille de l'objet cible. Ce formalisme rend très lourde la description d'un couplage géophysique (cf. section sur l'interface utilisateur). Le type informatique et le profil de l'objet sont à la base de la définition d'espace. Chaque objet (identifié par son nom) est une instance d'un espace. Sur ce principe de hiérarchisation on prévoit différentes instances temporelles d'un objet, identifiées par un *time stamp*. Si l'algorithme le demande, on peut prévoir différentes instances de chaque instance temporelle, identifiées par un *tag* entier. Par exemple on peut produire le même champs pour la même échéance temporelle avec deux méthodes de calcul différentes que l'on différencie par l'attribut *tag*.

Puisqu'aucune hypothèse n'est faite sur l'ordre de réception des objets produits ni sur le nombre de fois qu'un objet pourra être reçu (qui n'est d'ailleurs peut être pas connu d'avance), la gestion de la bufférisation dans PALM est beaucoup plus évoluée. Elle ne s'appuie jamais sur la *mailbox* de MPI (dont la taille risque d'être insuffisante sur des machines à mémoire distribuée) mais sur des espaces de stockage temporaire (collectivement appelés *mailbuff*) explicitement gérés par le *driver* et optimisés de façon à réduire le nombre de communications MPI (ex. accès direct aux mêmes zones mémoire par les unités à l'intérieur d'un *block*, choix de la distribution parallèle la plus efficace pour des communications avec plusieurs réceptions, recrutement dynamique des processeurs dédiés au stockage – dits *memory slaves* – avec *garbage collection* pour en minimiser le nombre).

Les envois seront directs ou bufférisés en fonction de la situation dynamique qui se présente au moment de la communication. La bufférisation explicite et permanente de certains objets peut être demandée par l'utilisateur. Les objets sont bufférisés avec leurs attributs *time* et *tag*, de façon à pouvoir les récupérer dans n'importe quel ordre.

3.3Les différences dans la gestion des instances temporelles

OASIS utilise les temps des communications seulement pour décider si déclencher un envoi ou une réception. Ce n'est qu'une façon d'associer le pas de temps du modèle au repère global dans lequel les communications sont définies. Les objets sont donc envoyés, traités et reçus pour un temps de couplage donné (le temps interne au modèle peut ne pas correspondre au temps de couplage par l'introduction d'un décalage ou *lag*). On fait une hypothèse de synchronisation logique de l'application qui conduit à recevoir les champs une seule fois par temps de couplage et dans le même ordre dans lequel ils ont été produits. Ce mécanisme est simple, mais l'hypothèse de synchronisation est plutôt forte. Des différences de vitesse de production et de réception (plusieurs pas de temps produits avants que l'on ne commence à les recevoir) vont se traduire par une utilisation plus ou moins intense de la *mailbox* MPI (cf. section suivante).

Dans PALM les objets échangés et manipulés gardent toujours une trace des attributs *time* et *tag* qui en identifient l'instance. Ceci permet dans les couplages dynamiques de différencier le moment et l'ordre de production du moment et de l'ordre de réception.

De plus, PALM gère en interne un troisième identifiant : si une instance (*time* et tag) d'un objet est produite plusieurs fois avant qu'elle ne soit reçue, le protocole de communication LIOO (last in only out) prévoit que la dernière version remplace les précédentes dans le *mailbuff*. Si l'objet est distribué, on ne pourra, par contre, éliminer une version précédente que quand toutes les parties de la nouvelle version auront été stockées. Un attribut version est donc associé à chaque objet bufférisé. Dans le buffer on aura seulement une version d'un objet complet mais on pourrait avoir plusieurs versions de ses parties si les processus de l'unité source n'étaient pas synchronisés. L'un des exemples qui avaient motivé l'introduction de ce concept dans PALM est un chargeur d'un

flux d’observations temps réel : dans ce cas l’unité source parallèle reçoit un flux de données en temps réel et produit plusieurs champs d’observations prétraitées pour des instants donnés. Le flux de données continue d’apporter des données pour différentes échéances temporelles. Le chargeur met à jour les champs d’observations pour différents *times* et les envoie à nouveau. Si le traitement des observations est local et la distribution spatiale non homogène, les différents processus peuvent avoir une désynchronisation importante. Quand l’unité cible déclenche une réception pour un *time* donné, elle doit récupérer le plus récent des champs d’observations qui ont été entièrement traités.

Une application temps-réel couplée pourrait se retrouver dans la même situation (ex. couplage atmosphère – chimie en temps réel avec émissions de surface observées).

3.4Les différences dans la bufférisation

La bufférisation des envois dans OASIS présente deux limitations :

•pour les communications qui passent par le *transformer*, la bufférisation est explicite. La réception d’un objet bufférisé est déclenchée par une notification que l’unité cible envoie au *transformer*. Cette notification s’ajoute à la fin d’une queue de requêtes que le *transformer* traite dans l’ordre. Si elle est précédée d’une série de requêtes d’autres traitements lourds, la réception sera inutilement retardée

•pour les communications directes la bufférisation des champs est à la charge de MPI et, selon l'implémentation MPI, ils peuvent être stockés dans la *mailbox* MPI avec des limitations de taille qui dépendent de l’implémentation , de la plate-forme de calcul et de réglages MPI paramétrables. Si plusieurs instances de champs de couplage s’accumulent avant d’être reçues, elles peuvent être (selon l'implémentation MPI) stockées dans la *mailbox* avec le risque de la saturer.

Le *driver* PALM gère activement les requêtes qui lui parviennent et peut décider de répondre à la requête de réception d’un objet déjà prêt avant d’utiliser le processeur qui le stocke pour une tâche d’algèbre (fonction de *scheduling*).

PALM gère explicitement le stockage temporaire des objets en fonction de l’état instantané du système au moment où la communication a lieu. Ce mécanisme est complètement portable, il est indépendant de l’implémentation MPI. Il s’ajuste au type de plate-forme en trouvant le bon compromis entre le nombre de *memory slaves* et la quantité de mémoire que chacun peut administrer.

3.5Autres différences

Dans PALM, l'information qui décrit comment les données sont réparties sur les processeurs est appelée *distributor*. Logiquement elle correspond à ce qui est appelé *partition* dans OASIS. Par rapport à OASIS3, PALM permet de gérer une information supplémentaire : la localisation des données. En effet un champ n'occupe pas nécessairement la totalité des processeurs d'un code parallèle, ou, au contraire, il peut être répliqué à l’identique sur tous les processeurs ou sur seulement un sous-ensemble. La souplesse de la primitive

grid_def_partition dans OASIS4 a dépasse cette limitation.

PALM met à disposition les types SINGLEPROC, REGULAR, REGULAR WITH HALO et CUSTOM. Le distributeur SINGLEPROC correspond naturellement à la partition Serial d'OASIS3. Toutes les autres partitions OASIS3 sont des cas particuliers du distributeur CUSTOM, dont la logique est tout simplement d'associer les coordonnées globales des données gérées par le processeur à leur localisation dans la mémoire locale. La syntaxe PALM n'impose pas la contiguïté de stockage de l'objet local. Le formalisme par blocks d'OASIS4 est très proche du format CUSTOM de PALM.

3.6Un compromis pour la fusion

Selon l'approche de fusion proposée dans la section 2.3, les tâches du *driver* deviendraient des services avec une modalité dynamique qui s'appuie sur un *driver* actif et une modalité statique qui se base sur la mémoire locale des codes et de leurs interfaces.

Le service de *routing* aurait deux modalités (cf. l'exemple détaillé à la fin de la section 2.4) : une modalité dynamique avec requêtes au *driver* et réponses *run-time* ou une modalité statique avec une phase de calcul et de renseignement initiale et le stockage de toutes les informations de routage dans la mémoire des interfaces, qui n'auraient ensuite plus besoin de l'intervention du *driver*. N.B. : en pratique, il y a une interdépendance forte entre le routage et les traitements algébriques puisque le routage dépend du nombre de points qui composent le *stencil* d'interpolation de chaque point cible; la redistribution entre deux unités d'objets parallèles avec des distributions différentes mais associés au même repère géographique -le *repartitioning* dans la terminologie OASIS- est un cas particulier où chaque point cible correspond exactement à un seul point source. Dans OASIS4, les intersections entre les domaines des processus sources et cibles déterminant les patterns de routage sont calculées par les interfaces de couplage lors de l'initialisation du couplage.

De même, la gestion de la bufférisation explicite deviendrait un service optionnel. Pour un couplage dynamique ou pour un couplage statique avec une trop forte pression sur la *mailbox* MPI on activerait la gestion explicite avec *mailbuff* et *memory slaves*. Pour un couplage climatique avec avance en temps synchrone ou presque, on pourrait débrancher la bufférisation explicite et utiliser la même stratégie de couplage qu'OASIS. Il est tout de même conseillé d'introduire un contrôle sur les étiquettes temporelles pour assurer la cohérence des échanges et de traiter les requêtes qui arrivent au *transformer* avec une gestion des priorités qui anticiperait les envois des objets déjà prêts dès que la requête de réception correspondante arrive au *transformer*.

Une évolution possible prévoit l'utilisation de processus mémoire indépendants du *driver* : il s'agirait d'un pool de gestionnaires de mémoire auxquels les codes s'adresseraient directement pour stocker (PALM_Put) ou récupérer (PALM_get) les données. L'identifiant du gestionnaire auquel doit s'adresser chaque processeur côté source ou cible peut être calculé une fois pour toutes dans la phase d'initialisation. Il s'agirait d'une évolution des *memory slaves* qui s'affranchiraient du *driver*.

4 Les transformations algébriques et leur pilotage

Les performances d'un couplage climatique avec des grilles différentes pour les différents modèles dépendent largement de la vitesse des procédures d'interpolation et, plus en général, des traitements algébriques. Ceci concerne non seulement la vitesse d'exécution des opérations (ex. interpolations parallèles) mais aussi l'intégration de cette tâche avec le reste de la simulation. Pour cette raison, cette partie a des connexions strictes avec les autres aspects du couplage.

Sur ce point particulier les conceptions initiales de PALM et OASIS diffèrent de façon importante. Ceci se répercute surtout sur la couche d'interfaçage des codes et sur le formalisme de l'interface utilisateur.

4.1L'approche OASIS orientée champ

Pour schématiser on peut dire que dans la conception d'OASIS on assume que les entités qui manipulent les champs (code source, *transformer*, code cible) sont actives du début à la fin de la simulation. L'accent n'est donc pas mis sur les codes mais plutôt sur la suite des transformations opérées sur un champ de couplage, de sa production à sa réception.

Dans le code, comme dans les fichiers de description du couplage, tout suit un ordre logique de traitement pour chacun des champs échangés.

Dans la documentation d'OASIS3 la suite logique des traitements est :

•time transformations

•pre-processing transformations

•interpolation

•the "cooking" stage

•post-processing

Dans OASIS3 et OASIS4, les phases de manipulation temporelle (accumulation, moyenne, min/max) et de *pre-processing* sont effectuées par la librairie d'interfaçage de l'unité source avant envoi. Pour les deux versions du coupleur, l'interpolation est à la charge du *transformer.* Pour OASIS3, la cuisine et le *post-processing* sont également gérés par le *transformer* alors qu'ils le sont (ou le seront, voir 4.3) par la librairie de couplage cible dans OASIS4 (avec, si nécessaire des informations supplémentaires provenant de la source, ex. le flux global avant interpolation pour CONSERV).

Il faut souligner une limitation d'OASIS4 : aucun traitement du *transformer* n'agit sur plusieurs champs de couplage à la fois. Par exemple, l'opération de FILLING d'OASIS3 agit sur un champs de couplage et sur un fichier de climatologie. Dans le *transformer* d'OASIS4, on ne pourrait pas utiliser le champ envoyé par un modèle global basse résolution pour compléter le champ envoyé par un modèle régional haute résolution; cette opération devrait être faite dans la librairie d'interfaçage du côté cible.

4.2L'approche PALM orientée action

Dans PALM, au contraire, l'accent a été mis sur les actions à accomplir, représentées par des unités sur le canevas de l'interface graphique PrePALM. On raisonne en termes d'action qui peut démarrer et on s'occupe des objets qu'elle requiert en entrée et qu'elle produit en sortie.

Tous les traitements se passent à l'intérieur des unités (qu'elles soient définies par l'utilisateur ou appartenant à la boîte à outils d'algèbre) et rien ne peut arriver à un objet en cours de communication (si ce n'est du monitorage ou du débogage). Ceci impose une forte limitation. Prenons comme exemple les manipulation temporelles d'OASIS (ex. l'envoi d'un champs obtenu par moyenne temporelle ou par accumulation sur une intervalle). L'interface PALM_Put prévoit l'envoi d'une seule instance temporelle à la fois. La liste des temps de communication sert seulement à ne transmettre que quelques instances ou à introduire un décalage. Pour construire un champ moyenné ou cumulé il faut décrire explicitement toutes les communications (pour chaque pas de temps) vers le *buffer* ou vers une unité algébrique et récupérer le résultat au temps de couplage.

Par contre, cette approche n'impose aucune limitation sur le nombre d'objets qui peuvent intervenir au cours d'un traitement algébrique.

4.3Les opérations dans les interfaces

L'un des points de force d'OASIS réside dans la possibilité d'effectuer une partie des traitements dans la librairie d'interfaçage PSMILe qui est exécutée par les processeurs des codes à coupler.

En particulier, la nouveauté principale introduite par OASIS4, est la détermination par les PSMILe dans la phase d'initialisation des patterns de communication entre processus sources et cibles. C'est en effet par les PSMILe que sont faits le calcul (parallèle) des intersections de grille pour chaque couple de processus source et cible ainsi que la détermination, pour chacune de ces intersections, des ensembles de points qui interviennent dans les opérations d'interpolation côté source et côté cible (EPIOS et EPIOT).

Les PSMILe s'occupent aussi de tous les traitements qui se font sans changement de maillage ni de distribution avant envoi ou après réception. Il s'agit des manipulations temporelles, et du *pre-processing* côté source et de la cuisine et du *post-processing* côté cible. Dans le cas de codes parallèles ces actions peuvent être locales (ex. moyennes temporelles) et devraient pouvoir être collectives (ex. calcul de flux globaux pour les corrections CONSERV). Pour le moment OASIS4 ne sait pas gérer la totalité des actions prévues dans OASIS3 et n'implémente pas encore d'actions collectives.

Dans les pistes de développement et d'optimisation de PALM on a depuis longtemps pris en compte la possibilité d'exécuter des traitements algébriques dans les routines d'interface des unités. Dans le cas de PALM cette étude avait dû aborder la complexité du choix du contexte d'exécution des opérations algébriques concernant plusieurs objets à la fois et était parvenue à la formulation du concept de « *Priority Unity Space Class* » ou PUSC (cf. rapport de stage de Charles Martin). Cette piste redevient d'actualité pour optimiser les

performances des *repartitionings* et des opérations d'algèbre sur des machines massivement parallèles. Avec cette amélioration, PALM pourrait traiter toutes les opérations prévues par les PSMILe comme des cas particuliers, d'autant plus que les opérations actuellement supportées dans OASIS agissent sur un seul champ à la fois, ce qui rend banal le choix du contexte d'exécution.

4.4Les vraies transformations parallèles

Une autre nouveauté introduite par OASIS4 est l'utilisation d'un *transformer* parallèle. Pour précision, le *transformer* d'OASIS4 ne fait que mettre un nombre de processeurs à la disposition d'une série d'opérations locales indépendantes. Les traitements unitaires sont déterminés dans la phase d'initialisation lors du calcul des intersections des domaines locaux côté source et côté cible. Chaque processus du *transformer* répond ensuite comme un automate aux événements déclenchés par les interfaces PSMILe côté source ou côté cible, selon une table de correspondances événement-actions déterminée par l'utilisateur lors de la configuration du couplage. Tous les processus font référence à la même table des correspondances, ce qui garantit qu'ils réagiront tous de façon cohérente, mais le bon déroulement des opérations n'est garanti que si les appels PSMILe sont effectués par tous les processeurs concernés des deux côtés de la communication. Le *transformer* n'a à aucun moment une vision collective du déroulement global d'une opération d'interpolation. C'est une approche très agile, mais elle a deux limitations :

•si l'on imagine implémenter un traitement qui utilise des valeurs intégrales calculées en cours d'interpolation sur les champs globaux

•si l'on veut équilibrer la charge de la tâche d'interpolation de façon indépendante des distributions parallèles des codes (par exemple, l'interpolation de champs échangés entre deux codes monoprocesseur ne peut actuellement pas être parallélisée)

4.5Recommandations pour la fusion

Sans doute la recommandation principale concerne l'implémentation des PUSC dans PALM ou, du moins, d'une version dégradée qui ne traiterait qu'un objet à la fois, soit sur l'unité qui le produit, soit sur l'unité qui le reçoit, à l'instar de ce qui est fait dans OASIS4. Une implémentation complète, avec prise en compte explicite du parallélisme des unités, permettrait de dépasser les limitations actuelles du PSMILe d'OASIS4 avec un effort de développement probablement comparable à celui de l'évolution du PSMILe : on pourrait envisager de pouvoir choisir où effectuer le *repartitioning* et les interpolations : sur une machine massivement parallèle on utiliserait un *transformer* pour ne pas ralentir (ou alourdir en mémoire) les unités, tandis que sur une machine avec moins de processeurs mais plus puissants (comme les SMP vectorielles) on utiliserait l'interface du code source pour faire des économies de processeurs.

La construction des poids et des intersections pour les interpolations devrait devenir plus flexible que ce qu'il y a actuellement dans OASIS4. On peut imaginer avoir le choix entre la modalité avec calcul *run-time systématique*, telle qu'on la trouve dans la version actuelle d'OASIS4 une modalité avec relecture en entrée de run de poids soit construits lors d'une première

exécution soit prédéterminés avant le run par l'interface graphique ou tout autre moyen externe au coupleur (cette dernière option est en fait en cours d'implémentation dans OASIS4). Cette approche permettrait de réutiliser les résultats des calculs pour toutes les répétitions d'un même couplage (ex. relances temporelles), mais aussi d'intervenir explicitement sur les fichiers de poids.

5 Les interfaces

5.1L'interfaçage entre codes et algorithme de couplage

La conception des routines d'interface est étroitement liée à toutes les considérations des sections précédentes. Le résultat le plus important est que, la philosophie des communications étant la même, les appels restent très proches. Au jeu de primitives PALM, pour la gestion des communications et des dates, il faudra ajouter les primitives OASIS pour la déclaration des grilles.

L'utilisation d'une interface graphique pourrait en plus permettre d'éviter certains appels dans les codes à coupler (ex. si les grilles sont statiques et invariables, elles peuvent être déclarées dans les cartes d'identité des codes, si elles sont dynamiques elles seront définies par l'appel à une primitive, à l'instar de ce que l'on fait dans PALM avec les espaces).

En ligne générale, une recommandation peut être émise en ce qui concerne le bon compromis entre simplicité des codes à coupler et aspects dynamiques du couplage. Toute information qui n'a pas vocation à changer au cours d'un couplage ou qui ne nécessite pas d'être calculée en phase d'initialisation ou en cours de simulation devrait être renseignée lors de la description du couplage dans l'interface graphique conviviale. Ceci permet non seulement de réduire le nombre d'appels dans les sources des codes à coupler, mais aussi d'effectuer certaines optimisations (e.g. précalculer les patterns de communication) au niveau de la préparation du couplage et non *run-time*.

5.2L'interface homme machine

Les différences d'approche dans PALM et OASIS que l'on trouve aux sections 4.2 et 4.3 se traduisent dans une forme différente pour décrire l'algorithme de couplage. Il faut répercuter cette différence dans l'interface graphique, de façon qu'elle soit bien adaptée aux deux types d'applications.

Le PrePALM actuel est bien adapté à la description de couplages où l'accent est mis sur l'enchaînement d'unités de calcul qui s'échangent des objets sur la même grille côté source et côté cible.

On peut envisager une modalité PreOASIS où l'accent serait sur le flux des traitements appliqués à chaque champ de couplage. Une facon de le faire est un menu déroulant contextuel pour chaque communication, qui contiendrait la liste des propriétés (temps, *lags*, etc.) et des traitements. Les paramètres de chaque traitement seraient spécifiés dans d'autres menus déroulants en cascade. Dans cette modalité, la complexité de l'interface due à la gestion des aspects plus évolués pourrait être cachée à l'utilisateur (non affichage des

menus ou affichage en grisé).

L’interface graphique de PALM sert aussi comme outils de monitorage de l’exécution en temps réel ou comme analyseur de performances *post-mortem*. Cet aspect a une relation forte avec l’utilisation d’un *driver* actif. On laissera à l'utilisateur la possibilité d’activer les services nécessaires du driver pour disposer des informations de suivi de l’application qui lui sont nécessaires.

6 Conclusions

Techniquement on ne voit aucun point de blocage pour la fusion entre OASIS et PALM. Certains points de développement demanderaient cependant un travail considérable : la réunification de la force de travail OASIS et PALM dans un seul groupe permettrait une meilleure efficacité, mais un renfort en ingénieurs est à prévoir.

Les lignes guide à suivre sont donc :

•Réécriture modulaire du *driver* avec possibilité d’activer des services dynamiques ou de les remplacer par des fonctions de service statiques fournis dans la phase d'initialisation ou à la charge des unités

•Optimisation des méthodes de bufférisation des envois avec différents choix en alternative

•Implémentation des traitements algébriques dans les interfaces des unités selon les lignes de conception des PUSC

•Amélioration de la gestion des I/O disque et des *restarts* avec l’utilisation de mpp_io

•Réalisation d’une unité algébrique *transformer* plus complexe et capable d’opérations collectives sur les objets globaux

•Recherches des voisins et interpolations à fournir en boîte à outils avec des alternatives en plus des librairies d’OASIS actuelles

•Uniformisation des routines d’interface avec appels spécifiques pour les descriptions des grilles géographiques

•Evolution de l'interface graphique avec choix entre la modalité orientée champ et la modalité orientée action

Un argument en faveur de la fusion de PALM et OASIS est qu'il reste du travail à fournir pour compléter l’implémentation d’OASIS4 afin qu’il réponde à la totalité des besoins des utilisateurs d’OASIS3. De plus, la migration vers OASIS4 n’a pas encore eu vraiment lieu dans la communauté de modélisation climatique, à la fois à cause de la complexité accrue d'OASIS4 et de la faiblesse de la pression actuelle dans cette communauté quant à l’utilisation de machines massivement parallèles.

Par contre, il reste à évaluer si la génération d´un outil multi-fonctionnel comme celui qui résulterait de la fusion de PALM et OASIS, et donc nécessairement plus complexe que chacun des deux coupleurs considérés séparément, constituerait une réponse adéquate aux besoins des

communautés d'utilisateurs de PALM et d'OASIS aujourd'hui bien distinctes. Il faut de plus de plus quantifier l'impact qu'aurait la fusion sur les collaborations déjà mises en place pour le développement de ces outils, principalement avec les partenaires du CERFACS dans le cas PALM et avec divers acteurs de la communauté de modélisation climatique dans le cas d'OASIS.