Calcul parallèle-distribué pour les problèmes multiphysiques

Application à l'aéroélasticité

Azzeddine Soulaïmani — Amine Ben Haj Ali

École de technologie supérieure
Département de génie mécanique
1100, rue Notre-Dame Ouest
Montréal (Québec) H3C 1K3
Canada

azzeddine.soulaimani@etsmtl.ca

RÉSUMÉ. *Cet article traite d'une méthodologie de résolution numérique des problèmes dits multiphysiques telle l'interaction fluide-structure. Cette méthodologie est basée sur un paradigme de décomposition fonctionnelle suivi d'une décomposition géométrique. Quelques détails techniques sont fournis pour clarifier la présentation et surtout à aider le lecteur à implémenter cette approche. Les différents algorithmes de résolution sont discutés et des résultats de performance sont montrés. L'application de l'approche préconisée au problème de l'aéroélasticité non linéaire est directe. Les résultats de simulation des instabilités aéroélastiques de l'aile Agard 445.3 se comparent bien avec ceux de la littérature.*

ABSTRACT. *A parallel-distributed computing based approach is proposed for the solution of some multiphysics problems. A functional decomposition of the global problem is used first followed by classical domain decomposition. Several technical details are presented for more clarity and for easing its implementation. All the techniques discussed in this paper are illustrated for the case of CFD-based aeroelasticity. A thorough performance study is shown along with simulation results for the Agrad 445.3 aeroelastic test case.*

MOTS-CLÉS : *problèmes multiphysiques, calcul parallèle et distribué, aéroélasticité, algorithme PGMRES.*

KEYWORDS: *Multiphysics, parallel and distributed computing, aeroelasticity, PGMRES algorithm.*

1. Introduction

La simulation numérique des écoulements tridimensionnels compressibles exige souvent de grandes ressources de calcul. La discrétisation des équations de Navier-Stokes ou même d'Euler nécessite généralement un maillage fin. Le grand nombre d'équations à résoudre pour ce genre d'applications engendre l'épuisement des ressources des machines traditionnelles en termes de capacité de mémoire, d'où le vif intérêt pour le calcul parallèle qui a le potentiel de repousser les limites des systèmes existants en mémoire et en puissance de calcul. L'apport du calcul parallèle devient particulièrement ressenti pour les problèmes multiphysiques (Rifai, Johan, Wang, Grisval, Hughes, Ferencz, 1999, Felippa, 2001). Parmi ces problèmes on peut citer : l'interaction fluide-structure en aéronautique, le couplage thermo-mécanique en turbomachines, les problèmes de vibro-aéroacoustique, etc. En effet, la résolution de ce type de problèmes nécessite un couplage entre les équations gouvernantes intrinsèques à chaque discipline. Ceci engendre, en plus de l'accroissement de la charge de calcul, des difficultés particulières de mise en œuvre informatique.

Ce papier présente une approche basée sur le calcul parallèle-distribué pour la résolution des problèmes multiphysiques couplés. Le papier est divisé comme suit : après cette brève introduction, l'approche préconisée est détaillée. L'aéroélasticité constitue un exemple concret où cette approche est appliquée avec succès. Dans la troisième section, un algorithme de résolution par sous-domaines des problèmes multiphysiques est présenté. Dans la dernière section, on présente les résultats de performance pour des problèmes d'écoulements compressibles Euler autour d'ailes rigides et flexibles.

2. Approche de résolution des problèmes multiphysiques

Un problème est dit multiphysique s'il met en interaction deux ou plusieurs phénomènes physiques. Les problèmes multiphysiques présentent des difficultés majeures. La connaissance pertinente de chaque discipline est indispensable pour bien définir, formuler et résoudre le problème global. De plus, la formulation numérique de ce type de problèmes aboutit généralement, à des systèmes algébriques de grandes tailles. La résolution « monolithique » des problèmes multiphysiques est à éviter pour plusieurs raisons informatiques et numériques. En effet, l'incapacité et l'insuffisance des systèmes informatiques actuels représentent un handicap majeur aussi bien au niveau du stockage qu'au niveau de la puissance de calcul. Les matrices générées sont souvent mal conditionnées. Pour pallier ces difficultés mathématiques, numériques et informatiques, une subdivision du problème global en sous-problèmes s'impose intuitivement. La résolution des sous-problèmes peut se faire de manière parallèle. Toutefois, le coût pour obtenir la solution dépend du choix de la stratégie de décomposition et de l'algorithme de couplage.

Avant d'entamer les détails de décompositions des problèmes multiphysiques, nous présentons un rappel de quelques stratégies du parallélisme.

2.1. *Approches SPMD et MPMD*

Dans l'approche SPMD (single program, multiple data (figure 1) (Leopold, 2001, Foster, 1995, Gropp, Lusk, Skjellum, 1996), on exploite le parallélisme des données. Celles-ci sont décomposées en structures régulières sur lesquelles un même programme est exécuté. Les performances obtenues dépendent du réseau de communication et de l'algorithme numérique de résolution. Cette approche est largement utilisée en calcul scientifique surtout pour les problèmes à un seul « domaine physique ».

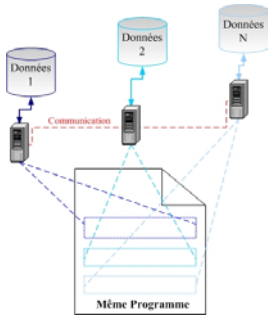

Figure 1. *Approche SPMD*

Dans l'approche MPMD (figure 2), les données sont décomposées (comme en SPMD) mais chaque processeur exécute un programme différent. Une caractéristique fondamentale dans cette approche est l'aspect asynchrone dans l'exécution. On cherche donc à équilibrer la charge sur les différents processeurs par le biais de la décomposition des données et par la décomposition du parc des processeurs disponibles.

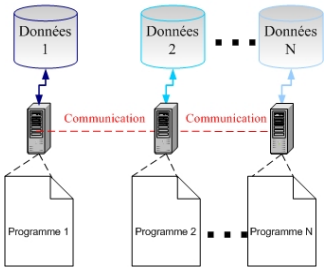

Figure 2. *Approche MPM*

2.2. *Etude de cas : l'aéroélasticité*

Le couplage entre l'écoulement et les mouvements des structures flexibles peut conduire à des instabilités qui mettent en jeu l'intégrité de la structure. Le flottement (flutter) est un phénomène d'instabilité dynamique de la structure flexible sous l'effet d'un écoulement à haute vitesse. Il est indispensable de s'assurer de l'absence d'instabilité, surtout pour les structures avioniques.

Différentes stratégies de résolution parallèle des problèmes multiphysiques, et en particulier pour le problème de flottement des ailes d'avions, ont été développées (Farhat, Lesoinne, 1996, Farhat, Lesoinne, Le Tallec, 1998, Soulaïmani, Ben Salah, Saad, 2002-1). Ces stratégies peuvent être globalement classées selon les deux approches de parallélisme précédemment rappelées.

La figure 3 illustre un algorithme séquentiel (intuitif) de résolution du problème de l'interaction fluide-structure. L'algorithme séquentiel est basé sur une alternance entre le résoluteur du champ fluide (CFD) et le résoluteur de la structure (CSD), intercalé par un appel d'une interface appelée « matcher ». L'interface assure les deux fonctions :

– projeter les pressions calculées par le fluide sur les noeuds de la structure ;

– mettre à jour les conditions aux limites du fluide en tenant compte des vitesses de déplacements de la structure.

En fait, puisque la structure se déplace, il arrive souvent qu'on adopte un maillage dynamique pour le fluide (*i.e.* on adopte une description cinématique arbitraire Eulerienne-Lagrangienne ALE (Soulaïmani, Saad, 1998)). Il est donc nécessaire de faire appel à un troisième module qui calcule le mouvement du maillage fluide (Mesher) à partir des déplacements de la structure.

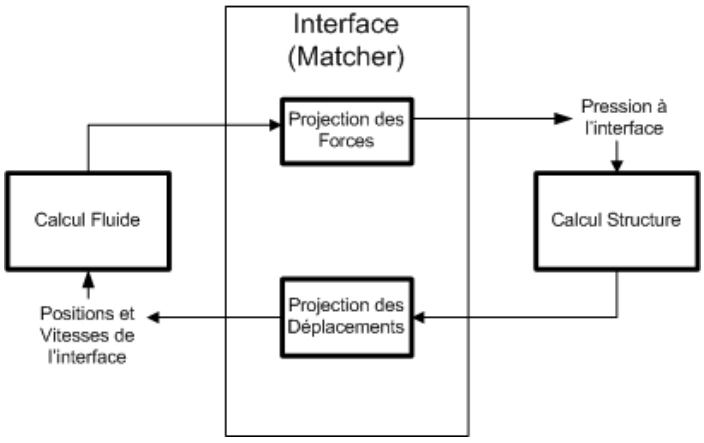

Figure 3. *Algorithme séquentiel : interaction fluide-structure*

Un premier algorithme itératif de résolution du couplage fluide-structure peut être décrit par les étapes suivantes.

« ALG1: Algorithme itératif de couplage fluide-structure »

1. Initialiser la géométrie, le champ d'écoulement et les sollicitations
2. **Boucle** dans le temps, de ***Ipas*** = 1, ***Npas*** :
3. **Itérations** de ***Iter*** = 1, ***Niter***
4. Calculer les déplacements du maillage compte tenu des déplacements de l'interface.
5. Calculer le champ fluide sur le nouveau maillage. Obtenir le champ de pression et projeter les pressions sur la structure.
6. Vérifier le critère de convergence sur le couplage fluide-structure.
7. **Fin** de la boucle sur les itérations.
8. **Fin** de la boucle sur le temps.

REMARQUES. — 1- Pour étudier la stabilité aéroélastique, à l'étape 1, on part d'un champ fluide et d'une configuration géométrique du maillage et on perturbe la structure durant un intervalle de temps (par exemple, en imposant une force sous forme d'un Dirac). 2- Les itérations en (3) sont utilisées pour obtenir un meilleur couplage entre les champs lorsque le pas de temps est relativement grand.

L'algorithme ALG1 peut être implémenté en séquentiel comme il peut être parallélisé selon l'approche SPMD en utilisant la décomposition du domaine (figure 4) où les quatre séquences de calcul (CFD, CSD, mouvement de maillage, projection des forces et des déplacements) sont exécutées par tous les processeurs (figure 5).

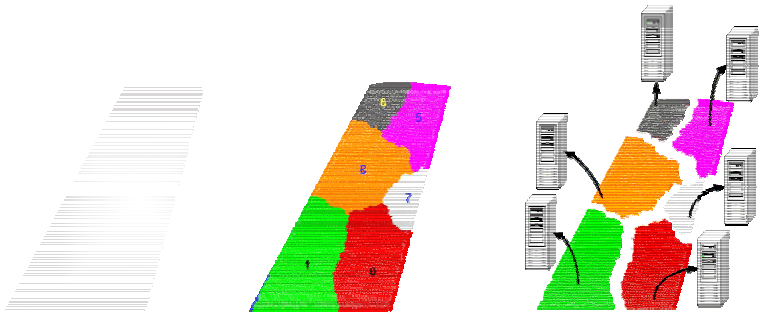

Figure 4. *Décomposition du domaine*

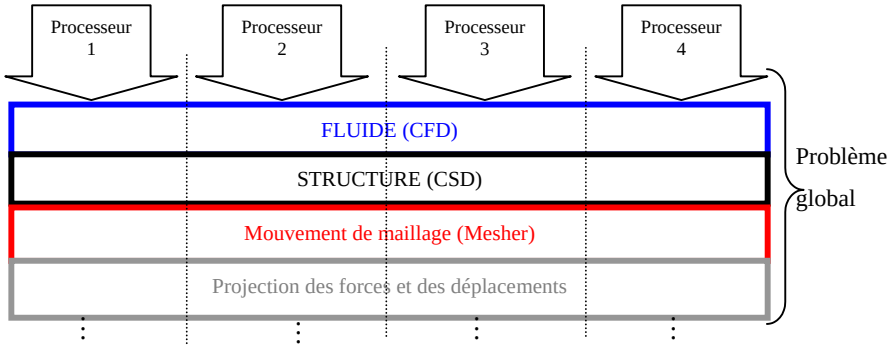

Figure 5. *Approche SPMD pour la résolution d'un problème multiphysique*

Une deuxième manière de paralléliser l'algorithme séquentiel ALG1 est de diviser le problème global en plusieurs domaines « fonctionnels », par exemple un pour le fluide et un autre pour la structure et le mouvement du maillage, de telle façon que les étapes 4-5 et 5-6 puissent se faire en parallèle (figure 6). Autrement dit, les opérations fondamentales sont subdivisées de manière naturelle. Chaque domaine fonctionnel est exécuté sur un domaine géométrique qui est à son tour divisé en plusieurs sous-domaines. Le parc de processeurs disponibles est alors divisé en plusieurs *familles de processeurs*, chaque famille est affectée à un domaine fonctionnel. C'est cette approche (ou paradigme) que nous adoptons par la suite et qui est implémentée dans notre code de calcul **PFES**.

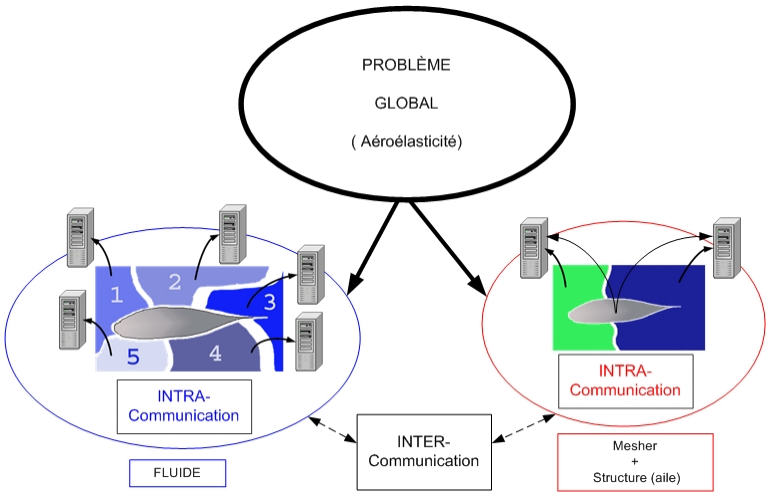

Figure 6. *Décomposition fonctionnelle*

2.3. *Quelques aspects d'implémentation*

Le paradigme énoncé ci-dessus est basé sur les décompositions géométrique et fonctionnelle englobant les deux approches **SPMD** et **MPMD**. La communication entre les processeurs de la même famille est appelée intracommunication. Celle entre les familles est appelée intercommunication. Plusieurs algorithmes numériques de résolution peuvent être considérés pour appliquer cette approche au problème global ainsi divisé. Les communications sont établies en faisant appel à la bibliothèque **MPI** (Gropp, Lusk, Skjellum 1996). Dans ce qui suit nous montrons quelques détails sur la réalisation de l'intracommunication et l'intercommunication avec la librairie **MPI**. Le programme **PFES** fait appel à quatre communicateurs.

– Le communicateur *MPI_COMM_WORLD* est le communicateur par défaut de **MPI**. Il permet de communiquer des informations entre tous les processeurs.

– Le communicateur *Family* est un sous-communicateur de *MPI_COMM_WORLD*. *Family* permet la communication entre les processeurs de la même famille (l'intracommunication). On peut le créer en faisant l'appel suivant : *CALL MPI_COMM_SPLIT(MPI_COMM_WORLD,myfamily-1,myproc-1,Family,ierr)* où *myfamily* est le numéro de la famille.

Il convient de donner un nom à chaque domaine fonctionnel (cela correspond à la notion de groupe selon la terminologie de **MPI**), cela se fait par l'appel suivant :

CALL MPI_COMM_GROUP(Family, FamilyGroup, ierr)

Le domaine fonctionnel *FamilyGroup* possède donc le communicateur *Family*.

– Le communicateur *Diplomacy* permet la communication entre les leaders des familles (intercommunication). De la même façon, on le crée par les appels suivants :

!---- Connaître le groupe associé au communicateur *MPI_COMM_WORLD*
CALL MPI_COMM_GROUP (MPI_COMM_WORLD,WorldGroup,ierr)
!---- Créer le groupe des leaders
CALL MPI_GROUP_INCL (WorldGroup, nbfamily,leader, leadersGroup, ierr)
!---- Créer le communicateur entre les leaders
CALL MPI_COMM_CREATE (MPI_COMM_WORLD,leadersGroup, Diplomacy, ierr)

Les voisins d'un sous-domaine sont évidemment les sous-domaines avec lesquels il partage des nœuds. Chaque sous-domaine a donc besoin de communiquer avec ses voisins.

– Pour éviter les communications point à point, un sous-communicateur de *Family* appelé *Neighbors* est créé. Contrairement aux autres communicateurs, le communicateur *Neighbors* est non *classique*. Ce dernier peut être vu comme une table de communicateurs. Chaque élément de cette table est utilisé par un processeur bien déterminé afin d'envoyer des informations à ses voisins. La complexité de l'implémentation de *Neighbors* vient du fait que les différents groupes qui constituent les voisins ne sont pas indépendants, en effet un processeur peut

appartenir à plusieurs groupes en même temps. De ce fait, il est indispensable de construire un communicateur dédié à chaque processeur.

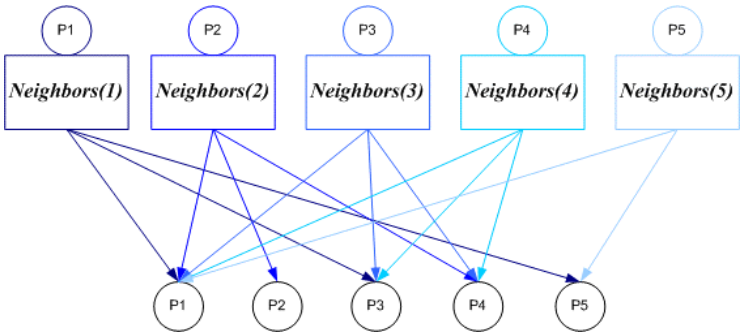

Figure 7. *Table des communicateurs intervoisins Neighbors*

Pour la création du sous-communicateur *Neighbors* chaque processeur a besoin de connaître la liste de ses voisins et les voisins de chaque processeur de la famille. Pour cela chaque processeur construit les tables suivantes :

– La liste de ses voisins : *Voisins*,
par exemple pour le processeur **P1** de la figure 7 la liste des voisins est :
Voisins = [1, 3, 5] (figure 8)

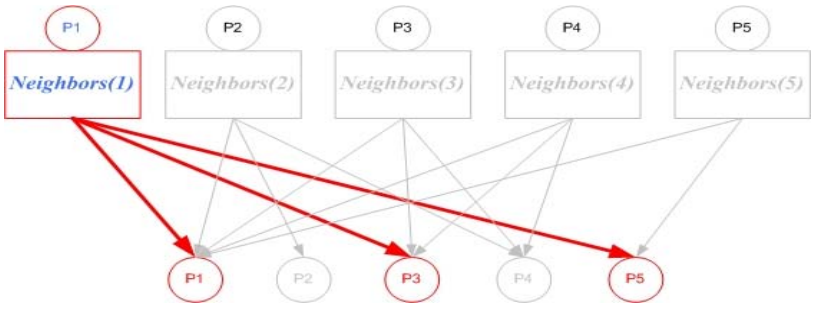

Figure 8. *Voisins du processeur 1*

– Le nombre cumulatif des voisins de chaque processeur de la famille : *Pointeur*.
Dans le cas de la figure 7: *Pointeur* = [1, 4, 7, 10, 13, 15]

– La liste de tous les voisins : *Voisinage*,

i.e., chaque processeur construit une table *Voisinage* lui permettant de connaître la liste des voisins de n'importe quel processeur de sa famille. Cette opération se réalise (efficacement) par exemple, en faisant appel à la routine *MPI_ALLgatherv* de **MPI**.

```
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
CALL MPI_ALLgatherv (voisins, nproc+1,MPI_INTEGER, Voisinage, npsize,
pointeur-1, MPI_INTEGER, FAMILY, ierr )
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

où *nproc* est le nombre de voisins et *npsize* est un vecteur qui contient le nombre de voisins de chaque processeur et qui peut être calculé à partir de *pointeur*.

Dans le cas de la figure 7, *Voisinage* = [1, 3, 5, 1, 2, 4, 1, 3, 4, 1, 5]

Grâce à cette structure chaque processeur possède toutes les informations nécessaires. Par exemple, tous les processeurs savent que :

Le processeur **P2** *possède pointeur(3)-pointeur(2) = 7 - 4 = 3 processeurs voisins. Ces processeurs sont* (figure 9) :

Voisinage (pointeur(2) **:** *pointeur(3)-1) = Voisinage (4 : (7-1))*
= Voisinage (4 : 6)
= [1, 2, 4]

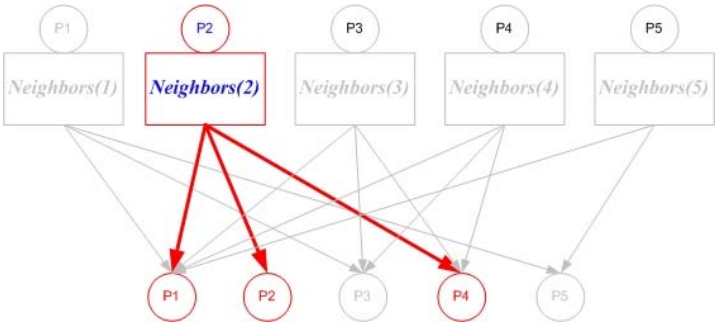

Figure 9. *Voisins du processeur 2*

On a dit que chaque processeur **i** possède une liste de voisins. On affecte alors un nom *proc_voisins*(**i**) à cet ensemble de processeurs (ou sous-groupe) et un communicateur *Neighbors*(**i**). Cela se réalise comme suit :

```
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
DO i=1, numbrother
!---- Créer les groupes de processeurs voisins
CALL MPI_GROUP_INCL(FamilyGroup,npsize(i),
voisinage(pointeur(i):pointeur(i+1)-1), proc_voisins(i), ierr)
!---- Créer le communicateur entre les voisins
CALL MPI_COMM_CREATE (FAMILY, proc_voisins(i), Neighbors(i), ierr)
ENDDO
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

(*numbrother* est le nombre de processeur du groupe *FamilyGroup*).

Montrons dans ce qui suit un exemple illustratif d'utilisation du communicateur *Neighbors*(*i*). Chaque processeur envoie à ses voisins une valeur d'une table ***msg1*,** à la réception chaque processeur voisin stocke cette valeur dans un vecteur ***proc*.**

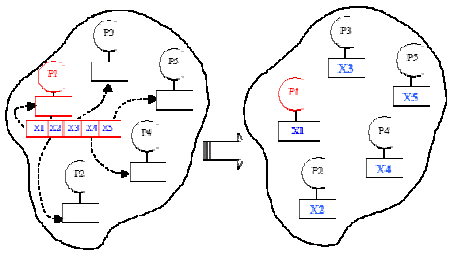

Figure 10. *MPI_SCATTER*

```
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
DO ip=1,numbrother
IF(Neighbors(ip)==0) CYCLE
Call MPI_Scatter ( msg1, 1, MPI_INTEGER, msg, 1, MPI_INTEGER,0,
Neighbors(ip), ierr
IF(myid/=ip) THEN
                j=j+1
                proc(nproc+j)=msg
               ENDIF
msg=0
ENDDO
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

(***myid*** est le rang du processeur dans la famille et ***nproc*** le nombre de ces voisins).

La procédure générale de résolution d'un problème multiphysique comme celui de l'aéroélasticité est illustrée à la figure 11.

A titre d'illustration nous montrons comment le leader du groupe fluide envoie le vecteur des pressions nodales au leader du groupe structure en utilisant le communicateur **Diplomacy** pour l'intercommunication.

```
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!--- le leader du groupe fluide envoie le vecteur des pressions nodales au leader du
! groupe structure-mesh
 IF(myid==1)THEN
 CALL MPI_BARRIER(DIPLOMACY,ierr)
 CALL
MPI_Bcast(PRES_INTERF,NCLI,MPI_double_precision,0,DIPLOMACY,ierr)
 ENDIF
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

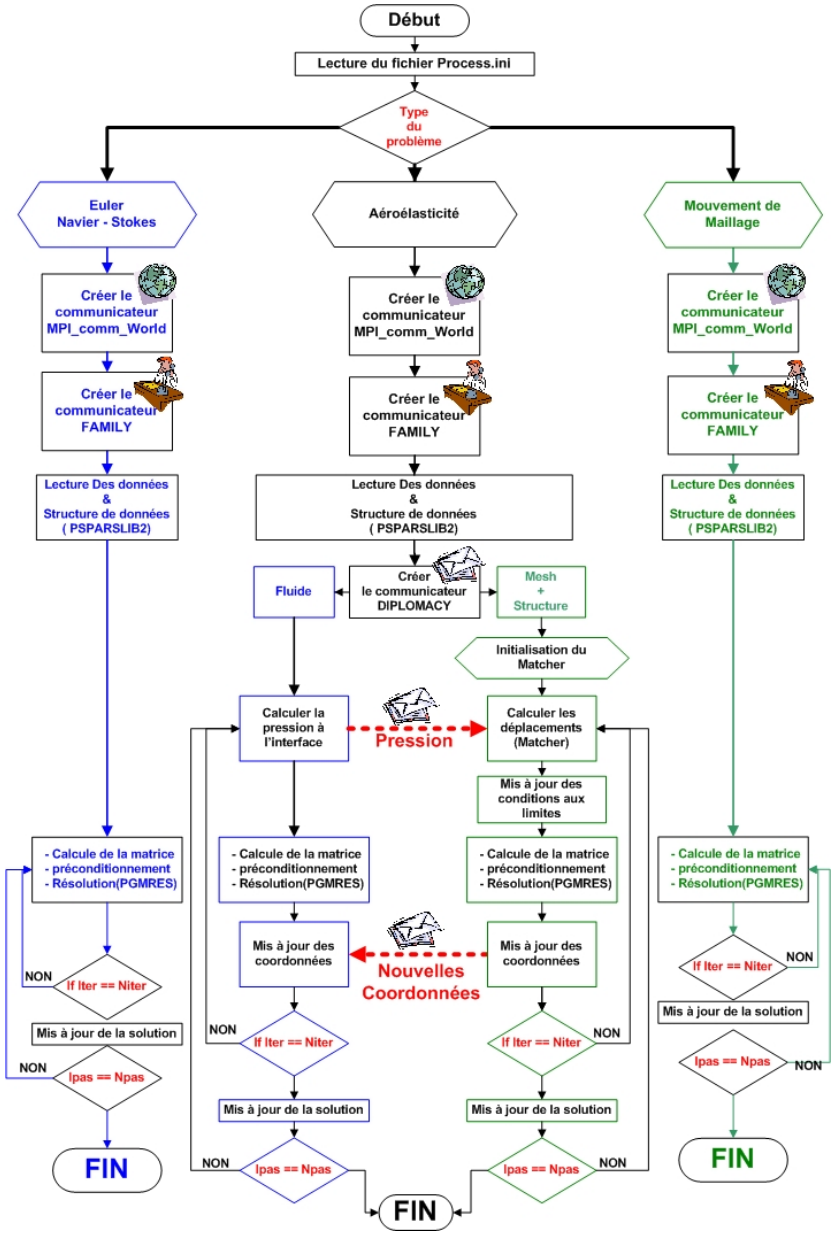

Figure 11. *Organigramme général du logiciel PFES*

Enfin, nous montrons un exemple d'intracommunication : le leader du groupe structure possède le vecteur VDLF des déplacements qu'il distribue à tous les membres de son domaine en utilisant le communicateur Family,

```
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
CALL MPI_BARRIER(FAMILY,ierr)
CALL MPI_Bcast(VDLF,3*dimVDLF,MPI_double_precision ,0,FAMILY,ierr)
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

3. Sructure de données pour la décomposition du domaine

Une façon commode qui permet de résoudre les systèmes découlant de la discrétisation des équations aux dérivées partielles, est l'approche de décomposition du domaine (géométrique). Le domaine est partagé en plusieurs sous-domaines, chaque processeur prend en charge un ou plusieurs sous-domaines. Les solutions partielles sont par la suite combinées, pour former une approximation de la solution du système global (Quarteroni, Valli, 1999, Felippa, Park, Farhat, 2001).

Pour implémenter un code basé sur l'approche de décomposition du domaine, on a besoin d'outils pour décomposer le domaine, associer chaque sous-domaine à un processeur, préparer les structures de données et pour résoudre les systèmes discrets. **PSPARSLIB** (Kusnetov, Lo, Saad, 1999, Saad, Kuznetsov, Lo, Malevsky, Chapman, 1997) est une bibliothèque bien adaptée pour effectuer ces tâches. La première étape consiste à diviser le domaine en plusieurs sous-domaines avec un outil comme **METIS** (Karypis, Kumar, 1998). La bibliothèque **PSPARSLIB** peut être considérée comme une parallélisation de la bibliothèque **SPARSKIT** (Saad, 1990). Elle offre les outils essentiels pour la résolution des systèmes découlant de la discrétisation des équations aux dérivées partielles, par décomposition du domaine. Les accélérateurs locaux sont identiques à ceux fournis dans SPARSKIT. Cette bibliothèque inclut des procédures de préconditionnement global basées sur les algorithmes « Additive-Schwarz », « Multiplicative Schwarz » et « complément de Schur ». PSPARSLIB utilise le stockage Morse (ou Compressed Sparse Row) de la matrice globale, *i.e.* uniquement les valeurs non nulles sont stockées. Durant le processus de résolution, chaque processeur (sous-domaine) doit échanger les valeurs calculées aux nœuds de l'interface avec ses voisins. Pour permettre un tel échange d'une façon efficace, il est important de déterminer la liste des nœuds d'interface (figure 12) avec les autres sous-domaines voisins.

Une étape, très importante dans le processus de préparation de la structure de données, est le stockage des informations concernant les équations relatives aux nœuds situés aux interfaces entre les différents sous-domaines. Les équations locales sont ordonnées de telle manière à trouver facilement les équations relatives aux nœuds d'interface (classées en dernier, voir figure 13). Plusieurs ajouts ont été apportés à la bibliothèque PSPARSLIB pour l'adapter à la méthode des éléments finis. En effet, PSPARSLIB a été conçue pour la résolution d'un système de type

Ax = b où **A** est une matrice déjà construite ce qui n'est pas le cas dans les codes d'éléments finis. Comme tout code d'éléments finis, le code PFES construit la trace de **A** de chaque sous-domaine (matrice locale) par l'assemblage des matrices élémentaires. La matrice globale **A** n'est jamais construite.

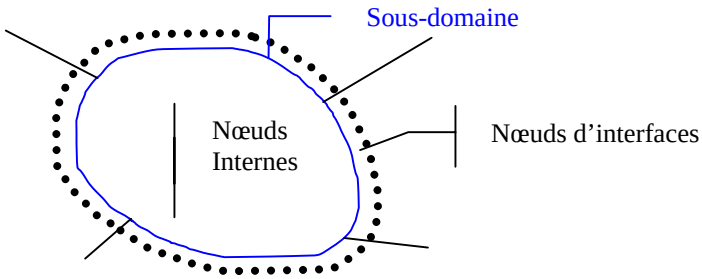

Figure 12. *Nœuds internes et d'interfaces*

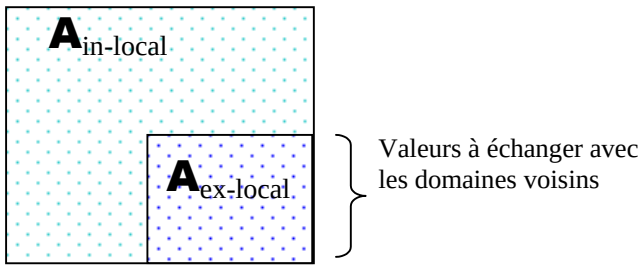

Figure 13. *Matrice locale du domaine* $\boldsymbol{A}_{local}$ *: blocs interne et externe*

Pour implémenter correctement la décomposition de type Schwarz-additive, un recouvrement d'au moins une couche d'éléments est nécessaire. Pour obtenir le bloc matriciel $\mathbf{A}_{ex\text{-}local}$ bien assemblé, chaque sous-domaine calcule sa contribution à $\mathbf{A}_{ex\text{-}local}$ et l'envoie à ses voisins pour un assemblage local. Cet échange se fait exclusivement entre les voisins du même domaine, et ceci grâce au communicateur ***Neighbors***.

4. Algorithmes de résolution

4.1. *Algorithme Schwarz additive*

En 1870, **Schwarz** (Akin 1999, Karypis 1998, Quarteroni, 1999, Soulaïmani, 2001) a proposé la procédure suivante : le domaine Ω est divisé en plusieurs sous-domaines $\{\Omega_1,\Omega_2, \ldots\Omega_n\}$, les sous-domaines Ω_i sont traités l'un après l'autre où une solution est obtenue pour un système local dont les conditions aux limites proviennent des solutions les plus récentes des autres sous-domaines. La solution globale du problème $\mathbf{A}x=b$ est trouvée par un assemblage des solutions locales.

Algorithme Schwarz additive

1) **For** i = ,….s :

2) calculer $\delta_i = \mathbf{R}_i^T \mathbf{A}_i^{-1} \mathbf{R}_i \left(b - \mathbf{A}x\right)$

3) **Fin** de la boucle sur les domaines

4) $x_{new} = x + \sum_{i=1}^{s} \delta_i$

où s est le nombre de domaines, $\mathbf{R}_i$ est l'opérateur de restriction associé au sous-domaine i et $\mathbf{A}_i = \mathbf{R}_i \mathbf{A} \mathbf{R}_i^T$ la matrice locale associée au sous-domaine.

$\mathbf{R}_i$ est une matrice de dimension $n_i \times n_i$ qui ne contient que des 0 et des 1, pour plus de détails voir (Saad, 1996-2). Pour les points d'interfaces, on calcule une combinaison linéaire des solutions locales δ_i.

4.2. *Algorithme de résolution PGMRES*

La résolution des systèmes linéaires creux de grande dimension est un problème courant en calcul scientifique. Parmi les méthodes itératives de résolution par projection, un algorithme bien adapté est **GMRES** « Generalized Minimal Residual » développé par Saad et Shultz (Saad, Schultz, 1996). Pour augmenter la performance de la résolution itérative, un préconditionneur de type **ILUT** (Saad, Schultz 1996, Soulaïmani, Ben Elhajali, Feng 2002-1) est utilisé. Pour le cas des problèmes non linéaires stationnaires ou instationnaires, l'algorithme de résolution utilisé est basé sur une procédure de marche dans le temps (time-marching) combinée à un algorithme de type Inexact-Newton et la version *non linéaire* de GMRES préconditionné (Soulaïmani, Ben Salah, Saad 2002-1) (Remarque : dans le cas d'un problème non linéaire la matrice **A** représente la Jacobienne).

La version parallélisée de GMRES, appelée **PGMRES**, regroupe les mêmes étapes que dans l'algorithme séquentiel. Cependant, quelques opérations, tel que le produit matrice-vecteur, ne peuvent être réalisées avec la même simplicité. Le fait que les données soient partagées entre les processeurs impose des étapes de communications supplémentaires.

Gardons à l'esprit que la décomposition du domaine n'est qu'un moyen pour surmonter les limites des machines. L'objectif principal étant de résoudre le problème global, une convergence globale doit être assurée à la fin du procédé de résolution.

Un calcul du résidu global s'impose.

$$\mathbf{Res}_{glob} = \mathbf{A}\,x - \mathbf{b} \qquad [1]$$

Le résidu global $\mathbf{Res}_{glob}$ ne peut être calculé directement. Un assemblage des résidus locaux, i.e. fournis par les sous-domaines, est alors indispensable.

$$\mathbf{Res}_{glob} = \text{Assemblage des } \mathbf{Res}_{\text{loc}} = \Re(\mathbf{Res}_{\text{loc}}) \qquad [2]$$

Soit R_i l'opérateur de restriction associé au sous-domaine i, on sait que :

$$\mathbf{A}_i = \mathbf{R}_i\,\mathbf{A}\,\mathbf{R}_i^T \qquad [3]$$

et pour tout vecteur global z_{glob} sa restriction sur le sous domaine i est $\mathbf{R}_i\ z_{glob} = z_i$.

On a aussi $\mathbf{R}_i^T\ z_i = z_{glob}$. On peut alors écrire le résidu global comme suit :

$$\mathbf{Res}_{glob} = \sum_{i=1}^{m} \mathbf{R}_i^T \mathbf{Res}_{loc} \qquad [4]$$

La version parallèle de GMRES nécessite aussi le calcul du produit scalaire (w,v), qui est défini par :

$$(\mathbf{w},\mathbf{v})_{Glob} = \sum_{i=1}^{m} (\mathbf{R}_i, w_i)_{Loc}\,(\mathbf{R}_i, \mathbf{v}_i)_{Loc} \qquad [5]$$

ainsi le carré de la norme du résidu global est :

$$\| \mathbf{Res} \|^2_{Glob} = \sum_{i=1}^{m} \| \mathbf{Res} \|^2_{Loc\,i} \quad . \qquad [6]$$

Pour assurer la continuité de la solution aux interfaces entre les différents sous-domaines, les degrés de liberté partagés sont moyennés. Cette opération ainsi que l'assemblage du résidu sont facilement implémentées grâce à la structure de données générée par la bibliothèque **PSPARSLIB**.

L'algorithme parallèle de résolution d'un problème non linéaire de type $\mathbf{F}(x)=0$ est détaillé ci-dessous (voir aussi la figure 14).

« ALG2: Parallel Schwarz-Newton-GMRES »

1. Décomposer le maillage.

2. Préparer la structure de donnée.

3. **Boucle** dans le temps, de ***Ipas*** = 1, ***Npas*** :

4. Calculer et factoriser la matrice de préconditionnement **M** (après chaque *n* pas de temps, n étant un paramètre entier).

5. **Itérations** de Newton, de ***Iter*** = 1, ***Niter***

6. Calculer le résidu initial $\mathbf{r}_0 = \mathbf{F}(x_0)$, $\beta = \left\| r_0 \right\|^2$ et.

7. Définir la ***m*** (***m*** +1) matrice $\overline{H_m}\left\{h_{i,j}\right\}_{1\le i\le m+1,\ 1\le j\le m}$ et l'initialiser.

8. Pour ***j*** = 1 jusqu'à ***m*** :

9. Calculer $z_j = \mathbf{M}^{-1} v_j$

10. Calculer la représentation locale de la solution perturbée, communiquer les valeurs aux interfaces pour construire la représentation globale en moyennant ses valeurs d'interfaces, et calculer $\mathbf{F}(x_0 + \varepsilon\, z_j)$.

11. Calculer le produit matrice vecteur $\mathbf{A}\, z_j$ par différences finies

$w_j = \dfrac{\mathbf{F}(x_0+\varepsilon\, z_j) - \mathbf{F}(x_0)}{\varepsilon}$ avec ε un réel petit

12. **Pour *i*** = 1 jusqu'à ***j*** :

13. Calculer localement $h_{i\,j} = (w_j, v_i)$ est sommer toutes les valeurs locales.

14. Calculer $w_j = w_j - h_{i\,j}\, v_i$

Fin boucle sur ***i***

15. Calculer localement $h_{j+1,\,j} = \left\| w_j \right\|^2$ et faire la somme sur tous les domaines.

Si $h_{j+1,\,j} = 0$ alors ***m*** = ***j***, **GOTO** ligne **17**

16. $v_{j+1} = \dfrac{w_j}{h_{j+1,\,j}}$

Fin boucle sur ***j***

17. Définir $\mathbf{Z}_m = \left[z_1, z_2, ..., z_m\right]$ et $\overline{H_m}\left\{h_{i,j}\right\}_{1\le i\le m+1,\ 1\le j\le m}$

18. Calculer y_m de façon à minimiser $\left\| \beta\, e_1 - \overline{H}_m\, y \right\|^2$.

Calculer localement $x_m = x_0 + \mathbf{M}^{-1}V_m y_m$ et communiquer les valeurs aux interfaces entre les sous-domaines, pour calculer la moyenne. Cette opération garantit la continuité de la solution globale entre les sous-domaines.

19. Si la condition de convergence est satisfaite **STOP**

sinon $x_0 = x_m$, **GOTO** ligne **6**

20. Fin des itérations de Newton

21. **Fin** de la boucle sur le temps

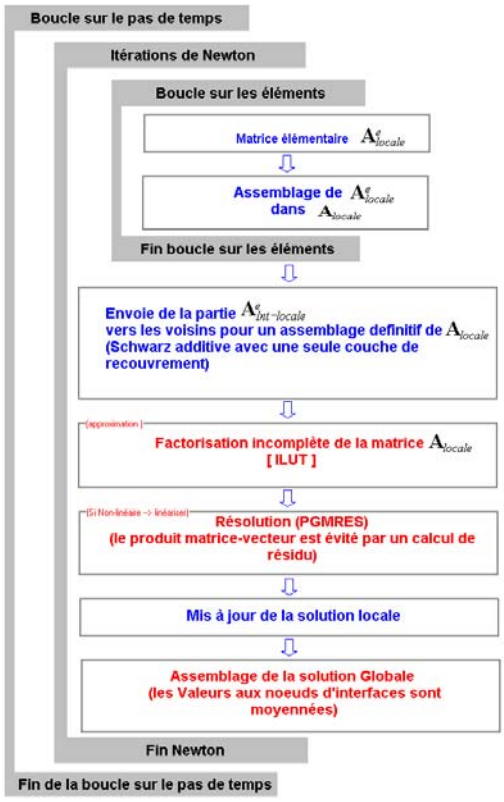

Figure 14. *Algorithme de résolution parallèle d'un problème non linéaire*

5. Application à l'interaction fluide-structure

Le programme **PFES** est un programme général capable de résoudre une grande variété de problèmes physiques et multiphysiques. Nous renvoyons à (Soulaïmani, Fortin 1994, Soulaïmani, Saad, Rebaine 2001, Soulaïmani, Saad, Rebaine 2000, Soulaïmani, Ben Elhajali, Feng 2002-2, Soulaïmani, Ben Elhajali, Feng 2002-3)

pour plus de détails sur les formulations mathématiques. En résumé, les techniques numériques utilisées sont les suivantes :

Fluide :

– la méthode des éléments finis stabilisée de type **SUPG** pour la discrétisation des équations gouvernantes (Euler ou Navier-Stokes moyennées) ;

– une discrétisation en temps du second ordre de type Gear ;

– l'algorithme **PGMRES** pour la résolution des systèmes discrets.

Les équations d'Euler pour un fluide compressible sont résolues sur un maillage mobile. Elles sont exprimées en fonction des variables conservatives **V** = (densité ρ, quantité de mouvement **U**, énergie totale **E**) comme suit :

$$\mathbf{V}_{,t} + \mathbf{F}_{i,i}^{conv}(\mathbf{V}) - w\,\mathbf{V}_{,i} = \Im \qquad [7]$$

ou sous la forme quasi linéaire :

$$\mathbf{V}_{,t} + (\mathbf{A}_i - w\mathbf{I})\,\mathbf{V}_{,i} = \Im \qquad [8]$$

où w est le vecteur vitesse du domaine, $\mathbf{A}_i$ sont les matrices jacobiennes du vecteur flux de convection.

Selon la méthode **SUPG**, la formulation variationnelle faible est donnée par :

$$\int_{\Omega} \left\{ \mathbf{W} . \left[\mathbf{V}_{,t} + \mathbf{F}_{i,i}^{conv}(\mathbf{V}) - \Im \right] \right\} d\Omega + \sum_e \int_{\Omega e} \left\{ \left(\mathbf{A}_i^t . \mathbf{W}_{,i} \right) \tau \left[\mathbf{F}_{i,i}^{conv}(\mathbf{V}) - \Im \right] \right\} d\Omega e = 0 \qquad [9]$$

où τ est une matrice de stabilisation.

Mouvement du maillage :

– une formulation cinématique arbitraire Eulérienne-lagrangienne est utilisée pour adapter le mouvement de maillage aux déplacements de la structure. Le mouvement du maillage est régi par les équations de l'élasticité non linéaire ;

– l'algorithme **ALG2** pour la résolution des systèmes discrets.

Plusieurs choix peuvent être considérés pour définir le mouvement du domaine. Dans le programme **PFES**, le mouvement de maillage est défini par les équations d'élasticité non linéaire : on cherche le déplacement **x** entre la configuration Ω(t) du maillage et celle à un instant ultérieur Ω(t+Δt) solution de :

$$\mathrm{div}\,(\mathbf{P}(\mathbf{x})) = 0 \quad (\text{ dans le domaine } \Omega\,(t)\,) \qquad [10]$$

où **P** est le tenseur des contraintes de Piola-Kirchhoff (PK1). Le domaine $\Omega(t)$ est donc considéré comme un matériau élastique qui subit de petites déformations mais de grandes rotations. Ainsi, le second tenseur de Piola-Kirchhoff (PK2) $\mathbf{S} = \mathbf{P}.\ \mathbf{F}^{-t}$, (avec **F** le tenseur de déformation) est linéaire en fonction du tenseur de déformation de Green $\mathbf{E}= (\mathbf{F}^t\ \mathbf{F} - \mathbf{I})/2$. On écrit alors : $\mathbf{S} = \mathbf{C}\ \mathbf{E}$ avec **C** la matrice d'élasticité dont les coefficients sont de l'ordre de l'inverse du volume de l'élément considéré.

En somme, les équations [10] sont résolues pour les déplacements **x** en considérant les conditions aux limites. A l'interface fluide-structure on a la condition de glissement : $\mathbf{w} \cdot \mathbf{n} = \mathbf{u} \cdot \mathbf{n}$ où **u** est la vitesse du fluide. La vitesse du maillage **w** est obtenue en appliquant un schéma de différences finies (identique à celui utilisé pour les équations d'Euler) à l'équation différentielle $w=x_{,t}$.

Structure :

– les déplacements de la structure sont calculés par superposition modale dans le cas de l'élasticité linéaire, l'intégration en temps utilise le schéma classique de Newmark.

Couplage fluide-structure :

L'algorithme ALG1 parallélisé selon l'approche MPMD (figure 11) est utilisé. Remarquons qu'afin d'assurer un bon couplage à l'interface, le fluide envoie les pressions et le couple structure-Mesher envoie la nouvelle géométrie et ce durant plusieurs itérations d'un même pas de temps. La pression envoyée peut être une combinaison linéaire de la pression calculée au pas précédent et celle prédite durant le pas courant. Une simple moyenne a donné de très bons résultats.

6. Résultats

Dans cette section les résultats des études de performance en écoulements Eulérien et en aéroélasticité seront présentés et discutés.

6.1. *Tests de performance*

Plusieurs tests de performances ont été réalisés. Le but est de mettre à l'épreuve le code PFES et d'étudier les différents facteurs qui entrent en jeu afin d'optimiser sa performance.

6.1.1. *Problème non couplé*

Une série de tests de performance ont été menés sur une grappe d'ordinateurs personnels, appelée Thunderbird (Azami, 2001). Chaque processeur de la grappe possède une vitesse d'horloge de 1.7 GHz et dispose de 512 Mo de mémoire vive.

Le test consiste à simuler l'écoulement autour de l'aile Agard 445.3 (Yates, 1987) en mode rigide et dans les conditions suivantes :

– Nombre de pas de temps : 100
– Type du pas de temps : fixe
– Nombre d'itérations de Newton par pas : 1
– La matrice est calculée et factorisée à chaque pas de temps
– Un maillage de 37965 nœuds, 177 042 éléments (figure 15)
– Ecoulement Eulérien.

Figure 15. *Maillage fluide (38K nœuds)*

Seul le nombre de processeurs utilisé change d'un test à l'autre. Les temps d'exécution CPU et les temps réels sont alors mesurés. Ces mesures ont été prises pour différentes décompositions du domaine (de 3 à 16 sous-domaines).

Le temps d'exécution CPU est le temps écoulé pendant les calculs sans prendre en considération, ni le temps de la communication, ni le temps d'attente appelé aussi le *temps mort.* Le temps réel est le temps qui sépare l'instant où le code a été lancé, à l'instant où la tâche demandée est accomplie. En d'autres termes, le temps réel est la période pendant laquelle l'utilisateur doit attendre pour avoir les résultats désirés. Les éléments sont partagés entre les processeurs le plus équitablement possible.

La figure 16 montre l'évolution du temps de calcul CPU des processeurs avec le nombre des sous-domaines.

Idéalement, pour une décomposition donnée, les temps CPU des processeurs devraient êtres identiques. Cependant, on remarque une légère différence entre les performances individuelles des processeurs. Cette différence peut être expliquée par la variation du nombre des nœuds locaux selon les processeurs.

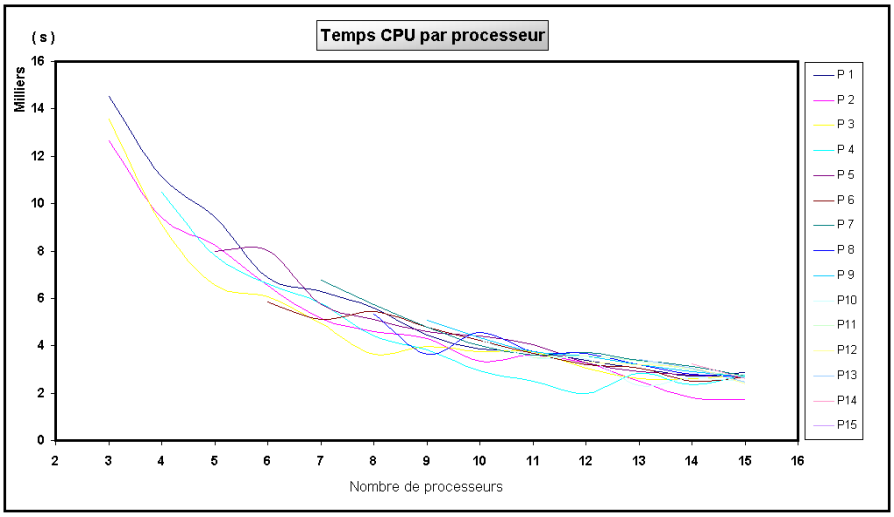

Figure 16. *Temps de calcul CPU en fonction du nombre des sous-domaines*

Le but des tests est de mesurer le *speed up* et l'efficacité du duo, la machine de calcul et le code PFES. Une façon de voir le *speed up* CPU global est de calculer ce dernier à partir du plus grand temps CPU des différents processeurs. En effet à cause des communications, les processeurs doivent êtres synchronisés, Le temps de calcul CPU global est alors déterminé par le processeur le moins rapide. La figure 17 présente le temps CPU ainsi calculé.

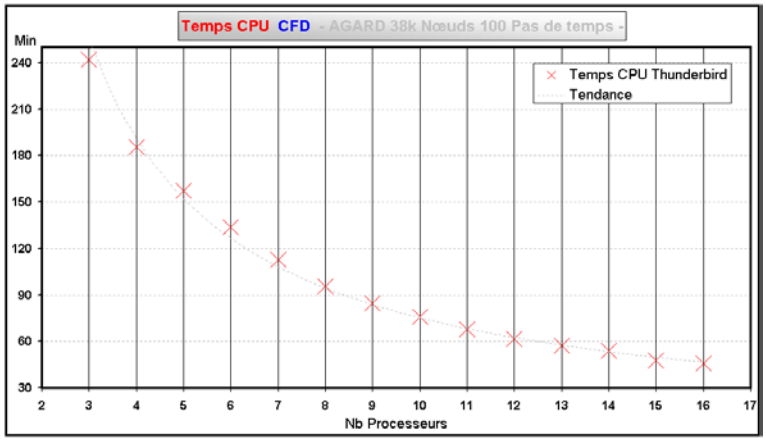

Figure 17. *Temps CPU pour un problème non couplé*

La figure 17 illustre clairement le gain en termes de temps CPU. En effet, pour 100 pas, le temps de calcul avec 3 processeurs est de 240 minutes, alors qu'avec 16 processeurs il est réduit à moins de 50 minutes. Le gain en CPU est encore plus crucial dans le dans le cas aéroélastique où des centaines de pas de temps sont nécessaires pour la recherche du point neutre de la stabilité. La figure 18 donne une idée sur la mémoire utilisée par processeur et pour chaque décomposition.

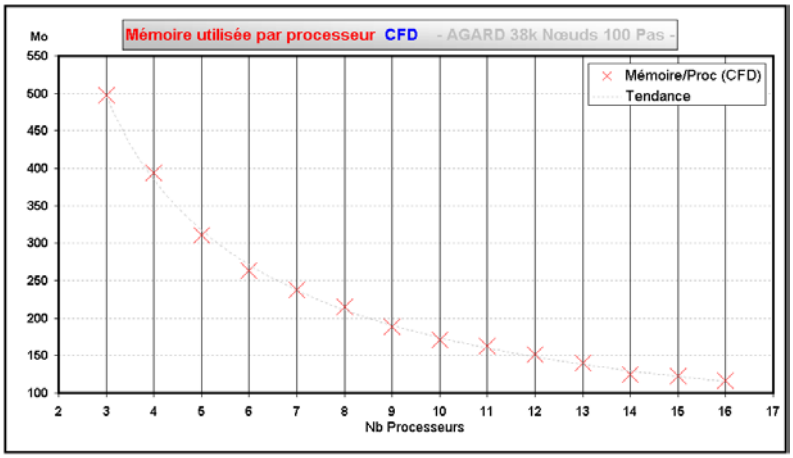

Figure 18. *Mémoire utilisée par processeur en fonction du nombre des sous-domaines*

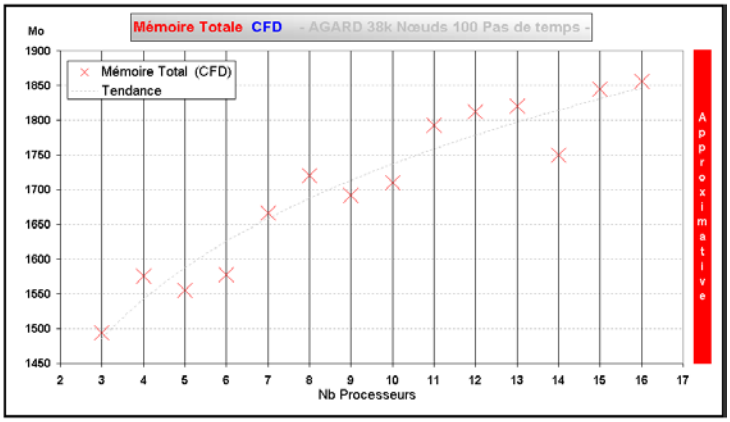

Figure 19. *Mémoire totale en fonction de la décomposition*

La quantité de mémoire vive par processeur limite le nombre minimal de sous-domaines à 3. La mémoire localement utilisée est nettement réduite par l'utilisation de multiples processeurs. Outre la rapidité des processeurs, cette réduction représente un avantage majeur des *clusters* par rapport aux machines parallèles à mémoire partagée. La quantité de mémoire par processeur peut être augmentée à moindre coût.

Étudions maintenant le comportement de la mémoire globale utilisée en fonction du nombre de sous-domaines. Les processeurs n'utilisent pas nécessairement la même quantité de mémoire. L'augmentation est engendrée par le nombre croissant des nœuds d'interfaces entre les sous-domaines. Mais cette variation est assez modérée.

Une manière plus claire de voir la performance, est le calcul du *speed up*. Le *speed up* est fonction du temps de calcul du code séquentiel T_s (s'il est disponible).

$$speedup(N)=\frac{T_S}{T_P(N)} \qquad [11]$$

$T_P(N)$ étant le temps obtenu avec N processeurs.

La mémoire d'un seul processeur est limitée à 512 Mo. Cette quantité ne permet pas de calculer le temps de calcul T_s. Une façon simple de l'estimer est de supposer que le *speed up* de 3 processeurs est égale à 3. La figure 20 présente le *speed up* en fonction du nombre de sous-domaines.

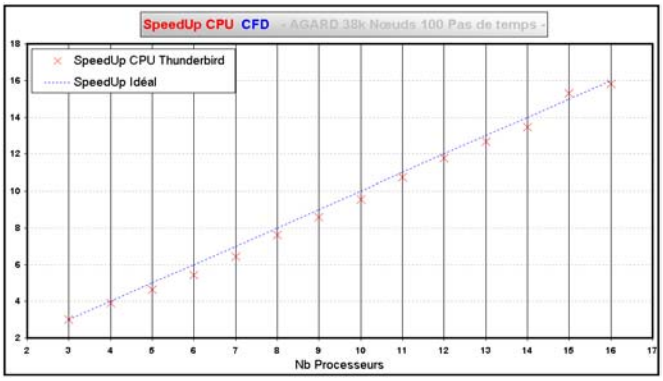

Figure 20. Speed up *CPU dans le cas de problème non couplé*

Le *speed up* ainsi calculé est très proche du *speed up* idéal ce qui est signe de haute performance. Un *speed up* idéal correspond à une réduction du temps de calcul de n fois, si n processeurs sont utilisés. Le *speed up* idéal n'est pas une limite absolue. Dans certains cas, il peut être atteint ou même dépassé, en particulier dans

le cas des problèmes multiphysiques où la décomposition fonctionnelle est intelligemment combinée avec la parallélisation des données. Dans certains cas, la convergence du code parallèle peut être plus rapide que celle du code séquentiel ce qui améliore le *speed up*.

Une autre mesure de la performance est l'efficacité. Elle est définie par le rapport du *speed up* et du nombre de processeurs :

$$efficacité(N) = \frac{speedup(N)}{N} \qquad [12]$$

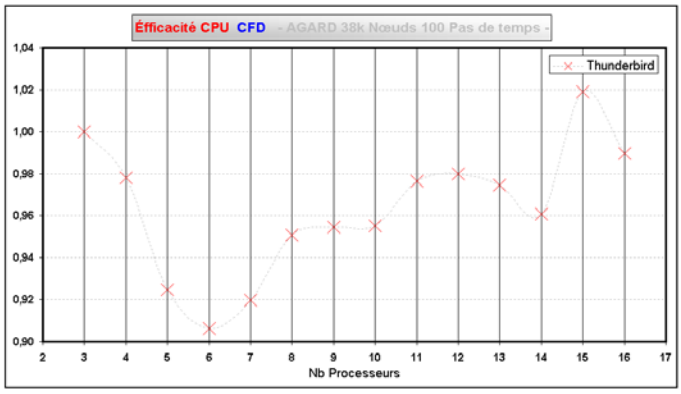

Figure 21. *Efficacité dans le cas d'un problème non couplé*

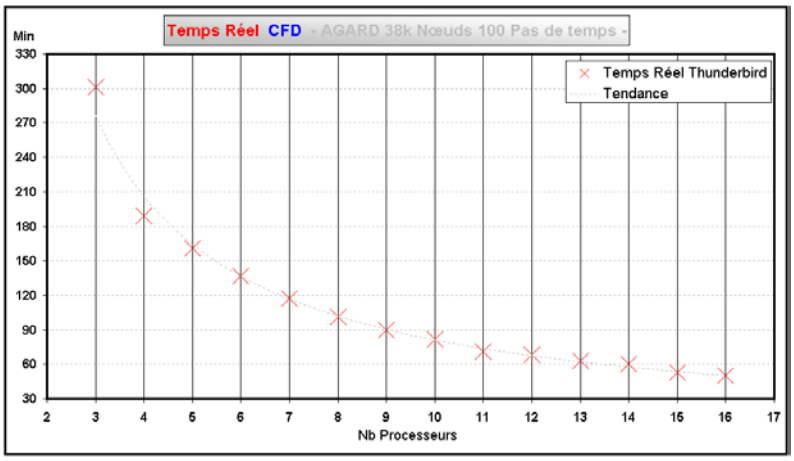

Figure 22. *Temps d'exécution réel pour un problème non couplé*

Une efficacité de 100 % correspond à un *speed up* idéal. Pour toutes les décompositions testées, l'efficacité est supérieure à 90 %, ce qui est en concordance avec la courbe du *Speed up*. Les figures précédentes illustrent sans doute, les excellentes performances du couple PFES-Thunderbird, en termes de temps de calcul CPU. Cependant, ce qui préoccupe plus l'utilisateur est le temps réel d'exécution. L'étude de l'efficacité en termes de temps réel, prend en considération le calcul, la communication et l'attente. Contrairement au temps CPU, le temps d'exécution total est le même pour tous les processeurs. Sur la figure 22, le temps d'exécution réel est donné en fonction du nombre de processeurs.

Similairement au temps CPU, le temps d'exécution réel est significativement réduit par l'utilisation de multiples processeurs. Le nombre de nœuds aux interfaces augmente avec le nombre de sous-domaines. Cette augmentation implique aussi une augmentation des données communiquées. L'utilisation d'un très grand nombre de processeurs (selon la finesse du maillage), engendrerait une augmentation excessive du nombre de nœuds aux interfaces et de la quantité de valeurs échangées. Dans ce cas, le temps de calcul et le temps de communication causés par cette augmentation peuvent influer dramatiquement les performances. Une comparaison qualitative entre la courbe du temps CPU et la courbe du temps réel donne une idée sur le coût de la communication et de l'attente.

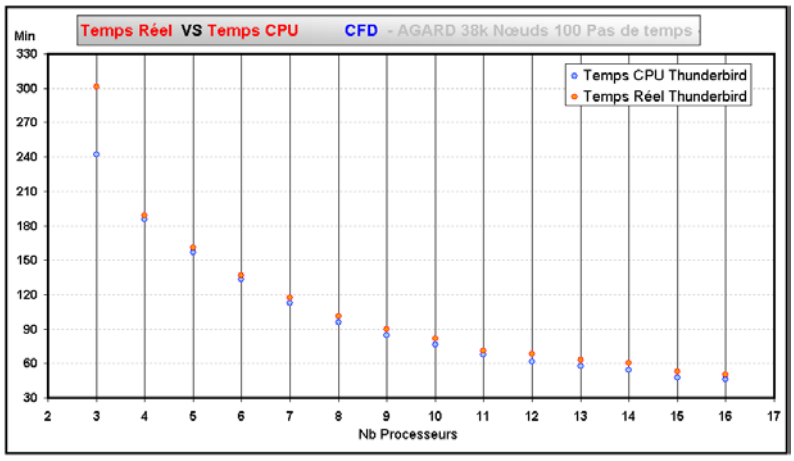

Figure 23. *Temps CPU et temps réel en fonction du nombre des sous-domaines*

D'après la figure 23, on remarque que le coût de communication reste assez faible par rapport au temps total, ce qui explique les bonnes performances. Plusieurs modes de communications offerts par la bibliothèque MPI ont été essayés. Les routines de communications collectives se sont avérées les plus rapides, stables et portables. Les routines de communication point à point sont parfois instables sur

certaines machines. Le rapport entre le temps des communications-attentes et le temps réel augmente légèrement avec le nombre de sous-domaines. Cette augmentation est le résultat de l'accroissement du nombre des nœuds aux interfaces. En effet, plus le problème est grand plus il est intéressant de le subdiviser sans subir, significativement, l'effet de la communication. La valeur exceptionnellement élevée enregistrée pour le cas de trois processeurs est probablement due au manque de mémoire, et donc d'écriture sur le disque. D'ailleurs la mémoire utilisée dans ce cas (500 Mo) est très proche de la limite de 512 Mo. Le *speed up* en termes de temps de calcul CPU donne une idée sur les performances individuelles des processeurs. La figure 24 donne le *speed up* en fonction du nombre de sous-domaines.

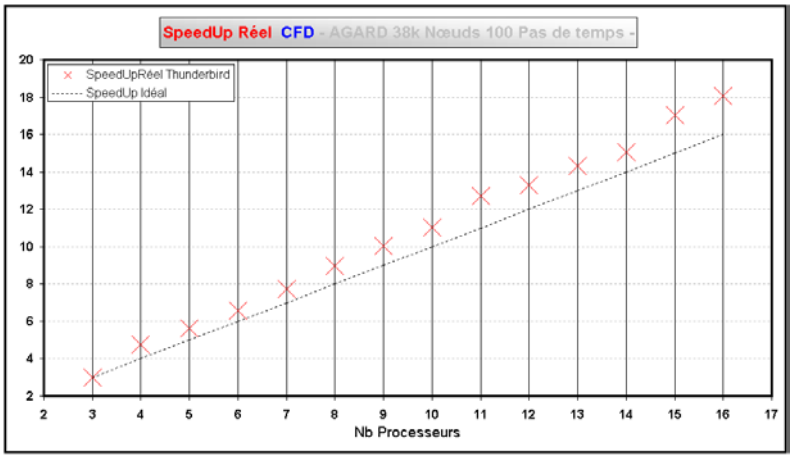

Figure 24. *Speed Up réel en fonction du nombre de processeurs*

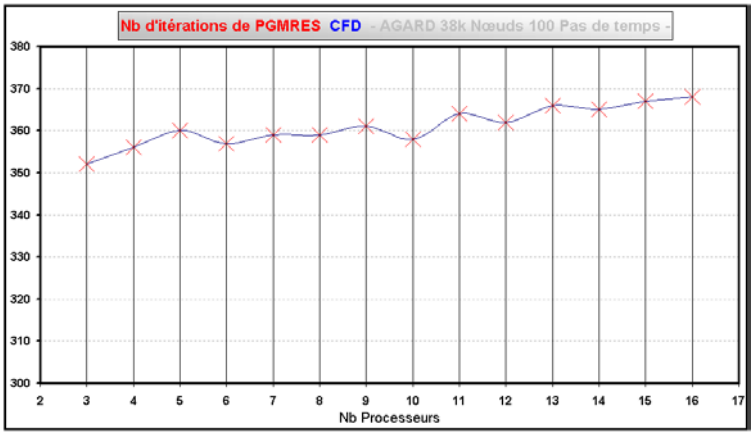

Figure 25. *Nombre total d'itérations en fonction du nombre de processeurs*

Outre le gain apporté par l'utilisation de multiples processeurs, la vitesse de convergence du code parallèle PFES a généreusement contribué à la réduction du temps d'exécution. En effet, la décomposition des données et le moyennage des valeurs aux interfaces n'ont affecté que sensiblement la convergence du code parallèle. Le nombre d'itérations de PGMRES a enregistré une très légère augmentation avec l'accroissement du nombre de sous-domaines.

Cette augmentation reste négligeable devant le gain en termes de temps de calcul surtout lors de la factorisation de la matrice pour le calcul du préconditionneur. À noter que ces deux dernières opérations coûtent beaucoup plus cher que les itérations de **PGMRES**.

Un autre test utilise un maillage plus fin (**135K** nœuds, **614K** éléments) de l'aile Onera M6 pour l'étude de la performance sur un problème d'une taille assez grande.

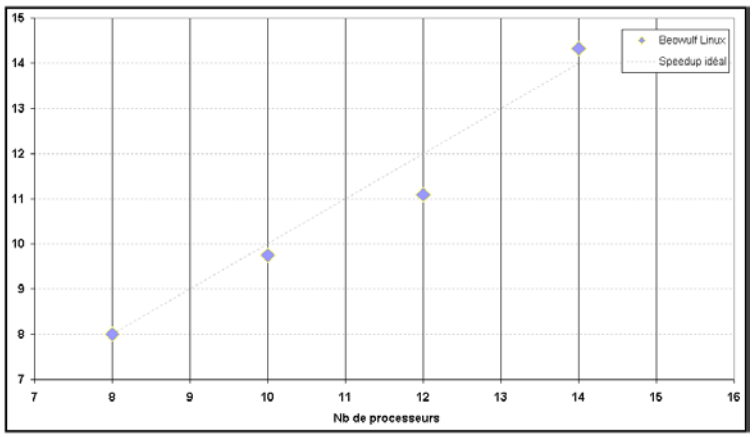

Figure 26. *Speed up dans le cas d'un maillage de 135K nœuds*

La figure 26 montre que l'évolution du *speed up* en fonction du nombre de processeurs est assez proche du comportement théorique. Le comportement faiblement non linéaire peut être expliqué par la légère différence du nombre d'éléments par processeur et par la variation non linéaire du coût de la factorisation.

6.1.2. *Problème d'aéroélasticité*

D'autres tests ont été réalisés en vue d'une étude de performance dans le cas des problèmes multiphysiques. Le même maillage de 38K nœuds de l'aile Agard ainsi que la machine Thunderbird ont été utilisés.

L'algorithme de résolution des problèmes multiphysiques adopté par PFES prévoit en plus de la décomposition du domaine physique, une décomposition

fonctionnelle basée sur l'aspect multiphysique lui-même. Pour le cas du problème d'aéroélasticité qui met en jeux un domaine dit fluide et un domaine dit structure-Mesh, les processeurs sont divisés en deux familles. Plusieurs combinaisons de nombre de processeurs attribués sont alors envisageables. Outre l'étude des performances, le but de cette deuxième série de tests est de trouver les combinaisons optimales pour chaque nombre de processeurs disponibles.

Il est à remarquer que le nombre d'équations à résoudre est beaucoup plus élevé dans le domaine fluide. Les combinaisons qui assignent un grand nombre de processeurs à la structure sont alors évitées. Le code PFES est une version parallèle du code séquentiel. Dans cette version le nombre de processeurs par famille ne peut être inférieur à 2.

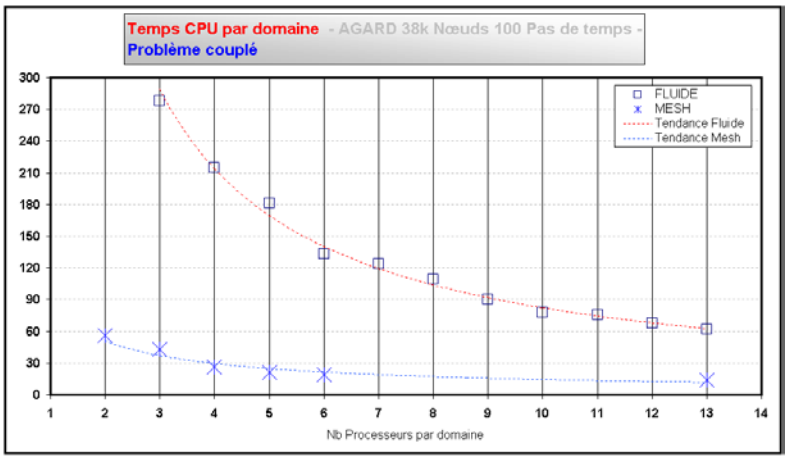

Figure 27. *Temps CPU par domaine pour un problème multiphysique*

Dans le cas des problèmes multiphysiques, la courbe du temps CPU par domaine permet à l'utilisateur (figure 27) de prendre une décision assez précise dans le choix du nombre de processeurs à allouer à chaque famille. Pour améliorer le temps de calcul réel, le temps d'attente doit être minimisé, voir réduit à 0. L'assignation des processeurs doit se faire de manière à obtenir des temps CPU les plus proches dans tous les sous-domaines fonctionnels. Selon la figure 27, pour n'importe quelle décomposition du domaine fluide de 3 à 13, le temps CPU fluide est supérieur à celui de la structure. Par conséquent, il est inutile d'assigner plus que 2 processeurs à cette dernière famille.

Remarquons que dans le cas d'une décomposition du domaine fluide en 13 sous-domaines, le temps CPU du fluide est très proche de celui de la structure décomposée en deux. Si la machine Thunderbird disposait d'un nombre plus grand de processeurs, l'assignation de plus de 13 processeurs au domaine fluide

engendrerait une réduction du temps CPU qui franchirait probablement la barre de 60 minutes. Dans ce cas, une attribution de 2 processeurs à la structure ne sera plus un choix optimal car les processeurs fluides seront plus rapides, l'attribution de 3 processeurs pour la structure s'imposera. La figure 28 illustre les performances enregistrées en termes de *Speed up* CPU pour chaque domaine physique.

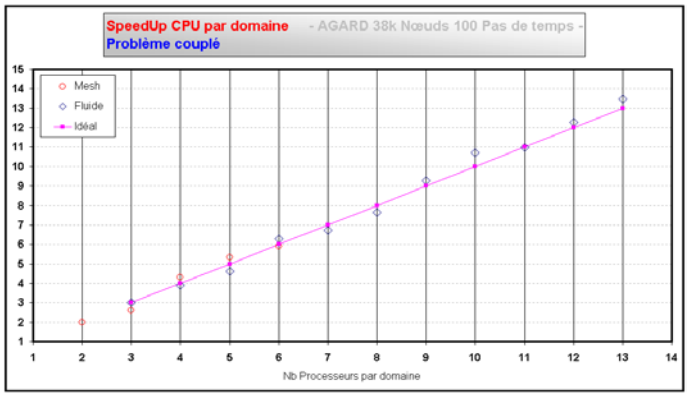

Figure 28. Speed up *CPU par domaine pour un problème multiphysique*

Les résultats trouvés révèlent des valeurs de speed up très proches de l'idéal. En effet, l'étude de *Speed up* CPU ne prend pas en compte les communications et les attentes. En termes de CPU, les deux domaines peuvent êtres alors considérés indépendants. Les résultats ainsi trouvés sont alors identiques au cas non couplé.

Bien que l'étude du temps CPU permette d'optimiser l'assignation des processeurs aux domaines physiques, l'étude de *Speed up* CPU ne reflète pas les performances du problème couplé. Une étude basée sur le temps de calcul réel s'impose. La figure 29 donne le temps d'exécution pour différentes décompositions étudiées.

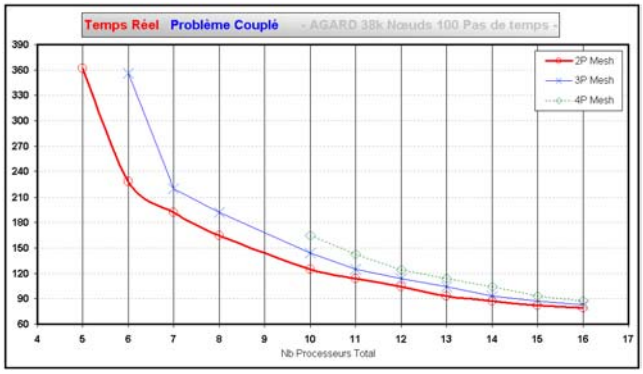

Figure 29. *Temps réels pour différentes combinaisons*

L'axe des abscisses représente le nombre total ***Nbproc*** de processeurs utilisés. Une courbe appelée « ***n***P Mesh » représente le temps réel dans le cas d'une décomposition où *n* processeurs sont assignés au domaine structure-Mesh (noté sur la figure par « Mesh »). Le nombre assigné au fluide est alors ***Nbproc*** – ***n***.

Globalement, l'utilisation de multiples processeurs engendre une réduction remarquable du temps d'exécution. Mais selon le nombre de processeurs attribués au Mesh, cette réduction est plus ou moins importante. Comme prévu par l'étude du temps CPU, l'utilisation de 2 processeurs pour le Mesh est optimale jusqu'à un nombre total de 16 processeurs. La courbe de *Speed up* suivante confirme d'une part, le choix d'une attribution de 2 processeurs à la structure et d'autre part, reflète les performances du code et de la machine dans le cas d'un problème multiphysique.

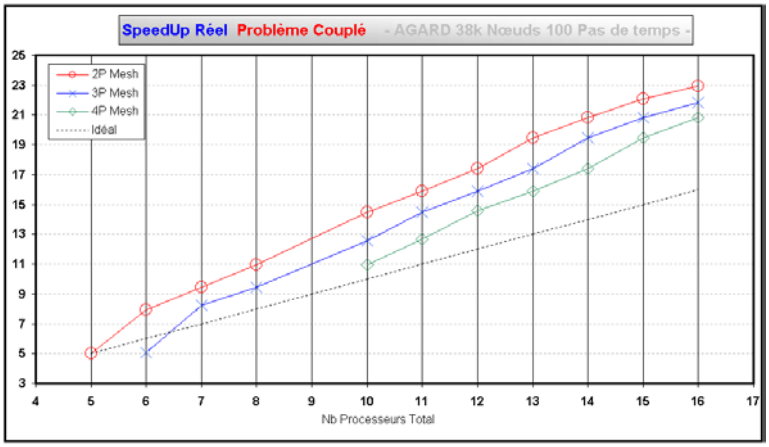

Figure 30. *Speed up réel pour différentes combinaisons*

L'utilisation de 16 processeurs engendre une réduction du temps d'exécution compris entre 21 et 23 fois selon la répartition des processeurs entre les familles. L'assignation de deux processeurs à la structure donne les meilleures performances.

L'augmentation du *Speed up* d'une façon linéaire montre que la taille du problème permet d'utiliser un nombre de processeurs supérieur à 16, sans toutefois perdre de performance.

L'efficacité a atteint une valeur de 1,5 pour 13 processeurs, ce qui reflète une très haute performance. Les valeurs d'efficacité confirment les points discutés sur la courbe du speed up.

L'utilisation de multiples processeurs a réduit sensiblement l'utilisation de mémoire par chaque processeur. Dans le cas des problèmes multiphysiques, des tables supplémentaires sont utilisées pour stocker les valeurs communiquées entre les familles.

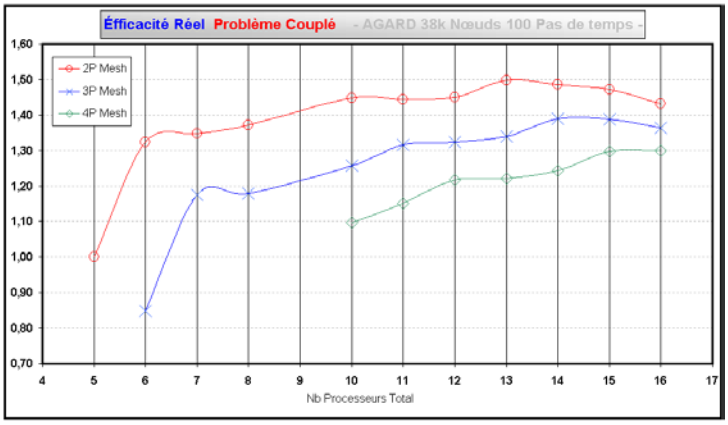

Figure 31. *Efficacité réelle pour différentes combinaisons*

Une comparaison avec le cas du problème non couplé permet de voir l'ampleur de l'augmentation de mémoire utilisée.

Une superposition des courbes de mémoire utilisée par un processeur fluide, du problème couplé et non couplé, montre une très légère augmentation de la mémoire.

La mémoire nécessaire au processeur fluide est beaucoup plus importante que celle utilisée par un processeur structure. C'est pourquoi seule la mémoire utilisée par les processeurs assignés au domaine fluide est évoquée.

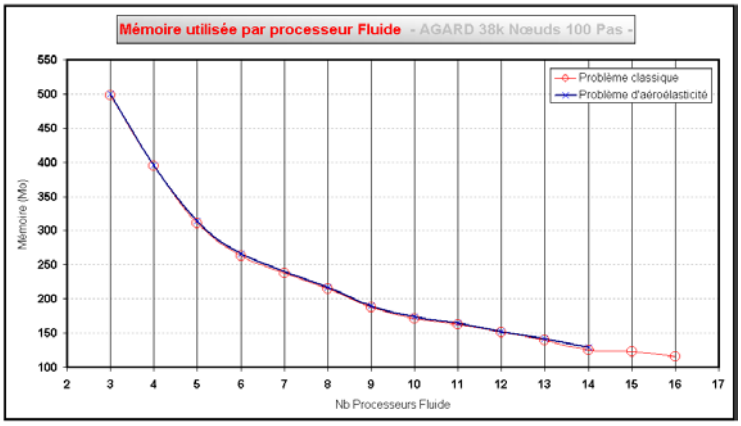

Figure 32. *Mémoire utilisée dans le cas d'un problème de CFD et un problème d'aéroélasticité*

6.2. *Application à l'étude de la stabilité aéroélastique de l'aile Agard 445.6*

Le problème de la stabilité aéroélastique l'aile Agard 445.6 (Yates, 1987) est un banc d'essai classique bien documenté. L'aile possède un profil mince NACA65A004. Le modèle considéré est le model 3 de la référence (Yates, 1987).

Pour la résolution des équations d'Euler, un maillage de 37 965 nœuds et 177 042 éléments a été utilisé. Le nombre d'équations couplées générées est 388 464. Concernant la structure, le maillage utilisé est constitué de 1 250 nœuds et 1 176 éléments coques quadrilatéraux. Les 5 premiers modes ont été obtenus à partir du code commercial ANSYS. Les fréquences propres calculées sont : 9.6 Hz, 39.42 Hz, 49.60 Hz, 96.095 Hz et 126.30 Hz. Elles sont très proches de celle rapportées dans (Yates, 1987).

Pour chaque nombre de Mach une solution d'écoulement est obtenue en mode rigide. Comme le profil est symétrique, la portance est nulle. La structure est alors perturbée par une force imposée en un point au bout de l'aile durant un pas de temps. La structure est évidemment relâchée pour étudier sa stabilité dynamique. On fait alors varier la pression dynamique de référence jusqu'à l'obtention éventuelle d'un mode instable.

La figure 33 présente une comparaison des indices de flottement obtenus par le code **PFES** avec les résultats expérimentaux (Yates, 1987) et avec quelques résultats numériques rapportés dans la littérature (Farhat, Lesoinne, 2000 ; Lee-Rauch, Batina, 1993 ; Gupta, 1996).

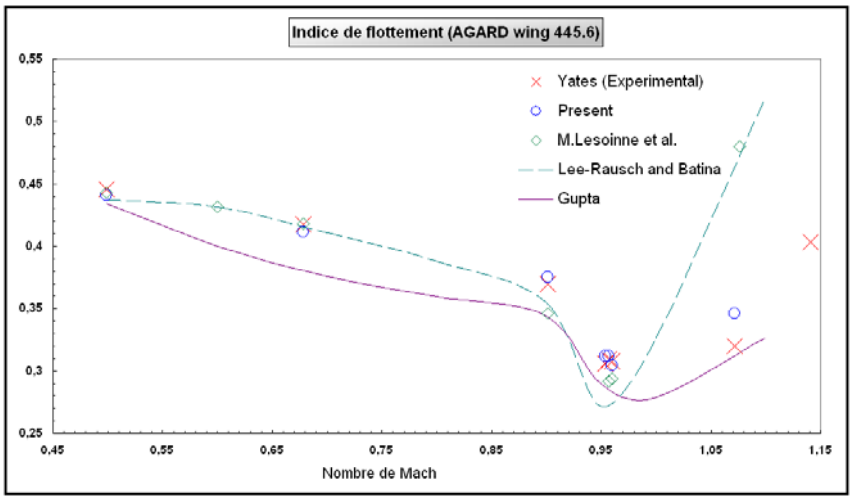

Figure 33. *Indice de flottement de l'aile Agard 445.6*

Un autre maillage plus fin a aussi été utilisé afin de valider les résultats précédents dans le cas critique (minimum d'indice de flottement) correspondant au nombre Mach = 0.96. Ce dernier maillage est constitué de 145K nœuds et de 678K éléments (figure 34).

Figure 34. *Maillage fluide (145K nœuds)*

(38K nœuds) (145K nœuds)

Figure 35. *Isobares sur l'aile Agard 445.6*

Les figures 35 montrent une comparaison des isobares sur l'aile en régime permanent à Mach = 0.96. Il est clair que le choc est plus net pour le maillage fin. En partant de cette solution, une force de 100N est appliquée durant un pas de temps en un point du bout d'aile. Le pas de temps adimensionnel utilisé (pour le fluide) est de 0.2 pour le maillage grossier. Pour le maillage fin, le pas de temps est de 0.1 durant les premiers 10 pas, puis il a été augmenté à 0.3. Cette réduction du pas de temps pour le maillage fin était nécessaire pour éviter l'écrasement des petits éléments au bout de l'aile qui subissent une grande torsion. Pour le maillage fin, on a utilisé 12 processeurs pour le domaine fluide et 3 autres pour les calculs de structure.

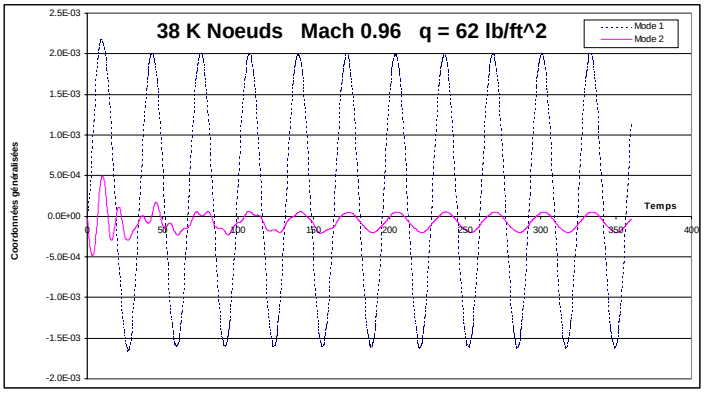

Figure 36a. *Isobares sur l'aile Agard 445.6*

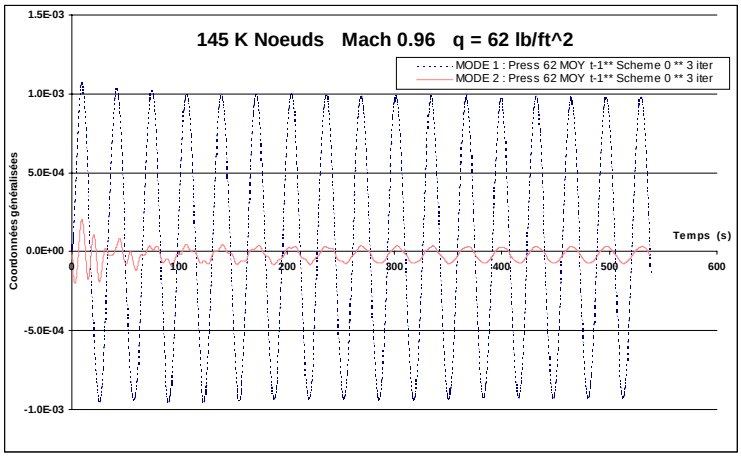

Figure 36b. *Isobares sur l'aile Agard 445.6*

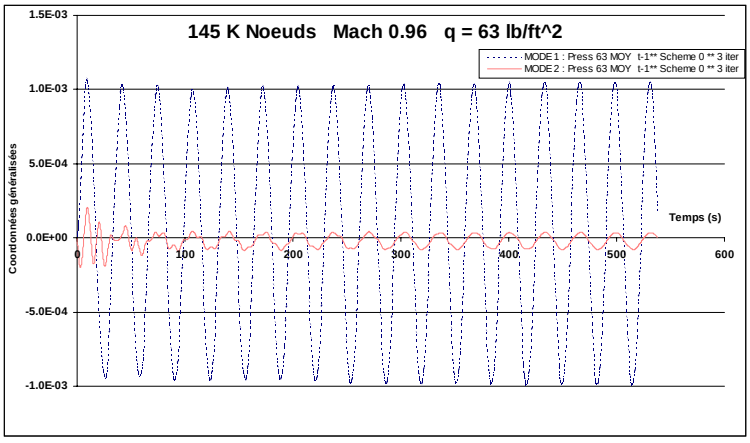

Figure 36c. *Isobares sur l'aile Agard 445.6*

Les figures 36 montrent l'évolution des coordonnées généralisées pour les deux premiers modes. Pour la pression dynamique q = 62 lb/ft^2, les résultats montrent une solution presque neutre dans le cas du maillage grossier. Elle est cependant légèrement amortie dans le cas du maillage fin. En augmentant la pression à q = 63 lb/ft^2, la réponse du premier mode est clairement divergente. Selon les résultats de Yates (Yates 1987) la pression critique à Mach = 0.96 est de 61.3 lb/ft^2. D'après nos résultats de calcul, nous pouvons conclure que le premier maillage utilisé, dit grossier, est suffisant pour reproduire avec une bonne précision les résultats expérimentaux du cas test aéroélastique de l'aile Agard 445.6. Les résultats sont plus affectés par la précision du schéma de couplage et de la discrétisation temporelle. Ces derniers constats vont être plus élaborés dans un prochain article.

7. Conclusion

Dans ce travail une méthodologie de résolution des problèmes multiphysiques est proposée. Elle a été appliquée avec succès pour le problème d'aéroélasticité.

En plus d'une parallèlisation de données, indispensable pour la résolution des problèmes de très grandes tailles, une parallèlisation fonctionnelle basée sur une approche MPMD « Multiple Program, Multiple Data » est préconisée. Cette approche offre plusieurs avantages du point de vue génie logiciel. En particulier, elle permet l'encapsulation des données et des méthodes qui est une qualité très recherchée pour l'implémentation rapide d'une nouvelle application.

Les études menées dans le cas d'une simulation d'un écoulement autour de l'aile Agard rigide et flexible ont révélé des performances très satisfaisantes. Les résultats de l'étude aéroélastique sur l'aile Agard 445.6 montrent que le code a réussi à reproduire fidèlement la courbe expérimentale d'indice de flottement.

Finalement, le code PFES, élaboré en se basant sur l'approche proposée, s'avère robuste, flexible et extensible.

8. Bibliographie

Akin J. E., "Object Oriented Programming via Fortran 90", *Engineering Computations*, vol. 16, n° 1, 1999, p. 26-48.

Azami Y., « Réalisation d'un environnement de traitement parallèle à partir d'un regroupement d'ordinateurs personnels », *Projet de maîtrise MGL, ÉTS,* 2002.

Decyk V. K., Norton C. D., Szymanski B. K., "Object-Oriented Programming with Fortran90", *Engineering Computations*, 2001.

Farhat C., High Performance Simulation of Coupled Nonlinear Transient Aeroelastic Problems, AGARD Report R-807, Special Course on Parallel Computing in CFD (l'Aerodynamique numérique et le calcul en parallèle), North Atlantic Treaty Organization (NATO), October 1995.

Farhat C. and Lesoinne M., "On the accuracy, and performance of the solution of three-dimensional nonlinear transient aeroelastic problems by partitioned procedure", *AIAA-96-1388*, 1996.

Farhat C. and Lesoinne M., "Two efficient staggered algorithms for the serial and parallel solution of three-dimensional nonlinear transient aeroelastic proble", *Comput. Math. Appl. Mech. Engrg*, 182, 2000, p. 499-515.

Felippa C.A., Park K.C., Farhat C., "Partitioned analysis of coupled mechanical systems", *Comput. Math. Appl. Mech. Engrg.*" vol. 190, 2001, p. 3247-3270.

Foster I., *Designing and Building Parallel Programs, Concepts and tools for parallel software engineering*, Addison Wesley, 1995.

Frahat C., Lesoinne M. and LeTallec P., "Load and motion transfer algorithms for fluid/structure interaction problems with non-matching discrete interfaces: Momentum and energy conservation, optimal discretization and application to aeroelasticity", *Comput. Maths. Appl. Mech. Engrg*, vol. 157, 1998, p. 95-114.

Gropp W., Lusk E., Skjellum A., "Using MPI, Portable Parallel Programming with the Message-Passing Interface", *Scientific and Engineering Computation Series*, 1996.

Gupta K.K., "Development of a finite element aeroelastic analysis capability", *J. Aircraft* 33, 1996, p. 995-1002.

Karypis G., Kumar V., "A fast and high-quality multi-level scheme for partitioning irregular graphs", *SIAM J. Sci. Comput.* 20, 1998, p. 359-392.

Kusnetov S., Lo G. C., Saad Y., "Parallel solution of general sparse linear systems using PSPARSLIB", *in choi-Hong Lai et al., editor, domain decomposition XI*, Domain decomposition Press, Bergen, Norway, 1999, p. 455-465.

Lee-Rauch E. M., Batina J. T., « Calculation of AGARD wing 445.6 flutter using Navier-Stokes aerodynamics », *AIAA paper 93-3476*, 1993.

Leopold C., *Parallel and distributed computing, A survey of models, paradigms, and approaches*, John Wiley & Sons, 2001.

Quarteroni A., Valli A., *Domain Decomposition Methods for Partial Differential Equations*, Oxford Science Publications, 1999.

Rifai S.M., Johan Z., Wang WP., Grisval JP., Hughes T.J.R. and Ferencz R., "Multiphysics simulation of flow-induced vibrations and Aeroelasticity on parallel computing platforms", *Comput. Math. Appl. Mech. Engrg.*, vol. 174, 1999, p. 393-417.

Saad Y., SPARSKIT: A basic tool-kit for sparse matrix computations, Report RIACS-90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA, 1990.

Saad Y. & Schultz M. H., "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems", *SIAM Journal on scientific and statistical computing*, vol. 7, 1996, p. 856-869.

Saad Y., "Iterative methods for sparse linear system", *PWS*, 1996-2.

Saad Y., Kuznetsov S., Lo GC., Malevsky A., Chapman A., “PSPARSLIB: A Portable Library of Parallel Sparse Iterative Solvers”, *PPSC*, 1997.

Soulaïmani A., Fortin M., “Finite element solution of compressible viscous flows using conservative variables”, *Comput. Maths. Appl. Mech. Engrg.*, vol. 190, 1994, p. 6735-6761.

Soulaïmani A., Saad Y., “An arbitrary lagrangian Eulerian finite element formulation for solving three-dimensional free surface flows”, *Comput. Maths. Appl. Mech. Engrg.*, vol. 162, 1998, p. 79-106.

Soulaïmani A., Saad Y. and Rebaine A., “Parallelization of the edge based stabilized finite element method using PSPARSLIB, in parallel computational fluid dynamics, towards teraflops, optimization and Novel Formulations”, *D. Keyes, A. Ecer, N. Satofuka, P. Fox and J. Periaux editors*, North-Holland, 2000, p. 397-406.

Soulaïmani A., Saad Y., Rebaine A., “An edge based stabilized finite element method for solving compressible flows: Formulation and parallel implementation”, *Comput. Math. Appl. Mech. Engrg*, vol. 190, 2001, p. 6735-6761.

Soulaïmani A., Ben Salah N., Saad Y., “Enhanced GMRES acceleration techniques for some CFD problems”, *International Journal of Computational Fluid Dynamics*, vol. 16 (1), 2002-1, p. 1-20.

Soulaïmani A., Ben Elhajali A. and Feng Z., “Nonlinear Computational Aeroelasticity: Formulations and Solution Algorithms”, *NATO-AVT, Meeting Proceedings RTO-MP*-089, 2002-2, p.45-01 to 45-13.

Soulaïmani A., Ben Elhajali A. and Feng Z., “A distributed computing-based methodology for computational nonlinear aeroelasticity”, *AIAA 2002-0868*, 2002-3.

Soulaïmani A., Wong T., Azami Y., Ben Elhajali A., “An Object-Oriented Approach for PC Clusters”, *To appear in Information:An International Journal*, 2002-4.

Yates E. C., “AGARD Standard Aeroelastic Configuration for Dynamics Response”, *Candidat Configuration I.-Wing 445.6. NASA TM 100492*, 1987.