Calcul parallèle appliqué aux écoulements de fluides complexes

Hugues Digonnet — Olivier Basset — Luisa Silva
Thierry Coupez

Ecole Nationale Supérieure des Mines de Paris
Centre de Mise en Forme des Matériaux
1, Rue Claude Daunesse F-06904 Sophia-Antipolis cedex
Hugues.Digonnet@ensmp.fr

RÉSUMÉ. *Dans ce papier, nous présentons des simulations numériques d'écoulements de fluides complexes réalisées sur une ferme de PC. Ces simulations ont été possibles grâce à une stratégie de parallélisation transparente et efficace : les développeurs n'ont pas besoin de connaître parfaitement une bibliothèque de programmation parallèle, mais ils utilisent des outils purement SPMD spécifiques pour construire leurs applications. Deux exemples d'applications ainsi que leurs résultats, pour des calculs qui nécessitent la résolution de systèmes linéaires à plus de 7 millions d'inconnues, sont analysés.*

ABSTRACT. *In this paper we present numerical simulations of complex fluid flows performed on a PC cluster. These simulations were possible thanks to a parallelization strategy that is transparent and efficient: each developer does not need to know a parallel programming library and its specific language. Instead, he uses purely SPMD tools to build their applications. Two examples are shown, as well as the computational results obtained for problems that need the resolution of linear Systems of over 7 million of unknowns.*

MOTS-CLÉS : *éléments finis, calcul parallèle, programmation orientée objet.*

KEYWORDS: *finites elements, parallel computing, object oriented programming.*

DOI:10.3166/REMN.16.703-722

1. Introduction

La parallélisation de softwares de simulation 3D est aujourd'hui essentielle, en particulier quand ils concernent des phénomènes complexes, tels que le comportement non newtonien, des mouvements d'interfaces, des changements de phase, ou des écoulements chargés de particules. Dans ce papier, nous nous concentrons sur des écoulements non newtoniens et/ou ayant des surfaces libres. Ce type de comportement est complexe, mais est aujourd'hui très bien étudié. Néanmoins, il demande de grosses ressources de calcul pour obtenir des résultats assez précis. Les équations résolues dérivent des équations de Navier-Stokes incompressibles. Un comportement viscoélastique est modélisé à travers l'introduction d'un terme élastique, par la considération d'un modèle de comportement dérivé de la mécanique moléculaire. L'extension aux cas avec surfaces libres est faite à l'aide d'une fonction de présence de chaque espèce existante, déterminée à l'aide d'une méthode du type « level-set ». Les inconnues à déterminer sont la vitesse, la pression, l'étirement, l'orientation moléculaire et la fonction de présence. La section suivante de cet article détaille la stratégie de parallélisation adoptée pour la résolution, par la méthode des éléments finis, des problèmes décrits ci-avant. La troisième section est consacrée aux applications numériques.

2. Méthodologie de parallélisation

En CFD (calcul en mécanique des fluides) l'enrichissement du comportement physique des matériaux, mais également, la prise en compte de géométries complexes ou la volonté d'obtenir plus de précision sur les résultats, entraînent inexorablement une augmentation des coûts de calcul. Dans ce contexte, l'évolution de la puissance de calcul des processeurs ne suffit pas, il faut alors avoir recours au calcul parallèle, qui consiste à faire travailler plusieurs processeurs en même temps afin de réduire les temps de calcul. Les dernières évolutions matérielles tendent à montrer que cette méthode de résolution ne pourra plus être ignorée. En effet le succès des fermes de calcul et l'apparition de processeurs multicœurs indiquent que les prochaines générations de programmes devront pouvoir s'exécuter en parallèle sous peine de ne pas utiliser pleinement leur puissance. Ce constat étant fait, il n'en reste pas moins que le développement de telles applications est généralement une affaire d'experts en parallélisme. Par contre la large majorité des développements d'une application en CFD concerne l'ajout de comportements physiques et est effectuée par des personnes n'ayant pas de connaissance du calcul parallèle (que nous appellerons développeur « classique »), mais ayant plutôt des connaissances spécifiques liées aux problèmes physiques à modéliser. Cette contrainte nous oblige à introduire un parallélisme « discret », tant du point de vue des utilisateurs que des développeurs.

Pour rester transparent vis-à-vis de l'utilisateur, nous avons choisi de conserver les mêmes fichiers de données entre une exécution séquentielle et parallèle. De même, les fichiers de résultats sont écrits dans le format choisi par l'utilisateur, de façon à ne pas l'obliger à changer de post-processeur. L'ensemble des autres tâches (plus « gourmandes » en calcul) est alors réalisé en utilisant plusieurs processeurs, comme l'illustre la figure 1.

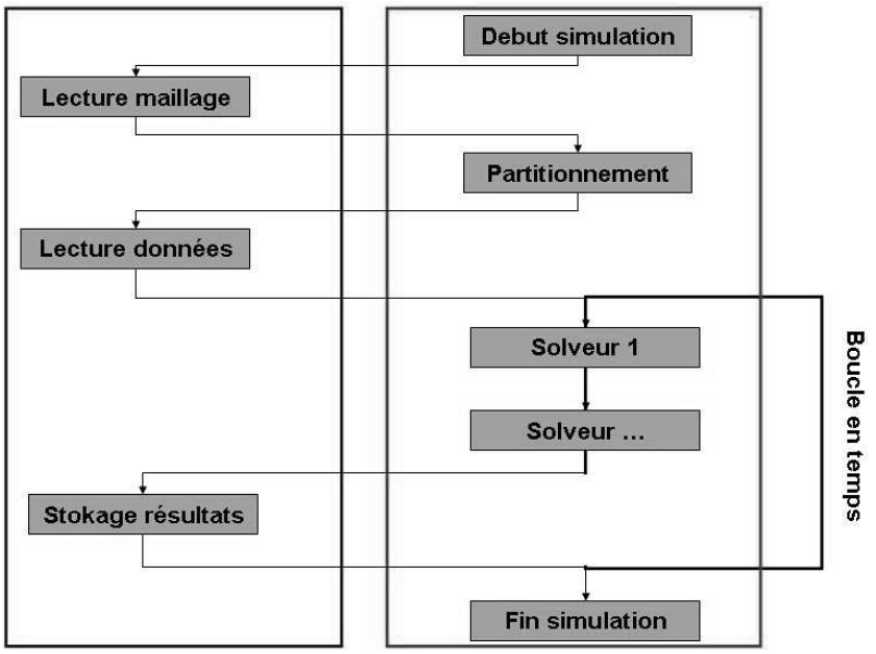

Figure 1. *Organigramme des principales tâches exécutées lors d'un calcul : à gauche, les parties exécutées sur un seul processeur, à droite, celles utilisant tous les processeurs*

La transparence du point de vue du développeur est plus délicate à mettre en œuvre, mais l'utilisation d'une programmation orientée objets et de canevas de développement permet à des développeurs sans connaissance de langage de programmation parallèle d'introduire de nouvelles fonctionnalités en dérivant celles-ci d'une interface prédéfinie et utilisable dans un contexte parallèle (Digonnet *et al.,* 2003). Par conséquent, ils peuvent profiter automatiquement d'une forte réduction en termes de temps de calcul, procurée par l'utilisation de plusieurs processeurs, lors de leurs tests de validation. Ce sont ces canevas que nous allons brièvement présenter ci-après. Dans un premier temps nous nous intéressons au stockage et à la résolution de très larges systèmes linéaires qui sont les plus consommateurs de mémoire vive et de temps CPU. Cette parallélisation représente déjà une importante

réduction des temps de simulation pour un nombre de processeurs réduit (voisine de 2 sur 4 processeurs (Digonnet *et al.,* 2003)). Si on veut utiliser efficacement des fermes de calcul comportant un plus grand nombre de processeurs il faut augmenter la partie parallélisée du code de simulation. Dans cette optique, la construction du système à résoudre doit être également faite en parallèle. Cette partie est plus délicate à mettre en œuvre car cette construction dépend fortement du problème physique traité. La méthodologie présentée par la suite montre comment la programmation orientée objet permet de contourner ces difficultés en cachant au développeur l'implémentation parallèle derrière une interface commune avec la version séquentielle. De même, la mise en place d'interfaces de développement qui s'intègrent dans l'application parallèle permet à des développeurs « classiques » d'implémenter des modèles physiques en dérivant ces interfaces qui servent de pont entre un module parallèle fixe et un ensemble de modèles évolutif.

2.1. *Une interface pour le stockage et la résolution de systèmes linéaires*

Dans le contexte d'un code éléments finis implicite, les problèmes physiques aboutissent généralement à la résolution d'un ou plusieurs systèmes linéaires de la forme Ax = b ou A et b sont construits en ajoutant, respectivement, les contributions de l'élément local à la matrice et au second membre, et x le vecteur global des inconnues (par élément ou par nœud selon le problème traité).

Notre première tâche fut donc de mettre d'un côté le calcul des contributions locales à l'élément (qui dépendent du problème physique à traiter), et de l'autre, le stockage et la résolution du problème global (indépendant du problème physique). Pour cela nous avons créé une interface « SystemeLineaire » qui lie les contributions locales à une librairie de stockage/résolution de systèmes linéaires. Cette interface contient uniquement la déclaration de certaines fonctions qui autorisent l'assemblage et la résolution de systèmes linéaires sans aucune restriction d'implémentation. Les personnes qui veulent construire un nouveau solveur (physique) n'interagissent qu'avec cette interface et non avec l'implémentation parallèle de cette dernière, qui est faite par un expert en calcul parallèle dans une classe dérivée. Cette interface doit être relativement générale pour rester ouverte à de nouvelles et plus performantes méthodes de résolution. Les principales fonctionnalités sont :

– le nombre d'inconnues locales ainsi que leurs localisations sont libres (inconnues nodales ou par éléments),

– systèmes linéaires symétriques ou non symétriques,

– divers choix de résolution (GMRES, Résidu conjugué...), et de preconditionnement (aucun, bloc diagonale, ILU...).

Pour l'implémentation de cette interface, nous avons choisi PETSc (McInnes *et al.,* 2002), qui contient un large nombre de méthodes de résolution de systèmes linéaires symétriques ou non, en séquentiel ou en parallèle, ainsi que différents preconditionneurs. La première classe dérivée de « SystemeLineaire » fut la version

séquentielle, puis une deuxième implémentation parallèle a été faite. Cette dernière s'appuie sur une partition du maillage en sous-domaines, chacun affecté à un processeur. Cette partition de maillage est effectuée grâce à un partitionneur parallèle réalisé dans le cadre du projet européen DRAMA (Basermann *et al.,* 2000). L'utilité de cette interface devient évidente : la même fonction a le même comportement dans sa version séquentielle ou parallèle, le développeur classique use de façon transparente de la version séquentielle ou parallèle sans se soucier des éventuelles procédures parallèles qui peuvent exister. Il n'a qu'à spécifier s'il veut utiliser un ou plusieurs processeurs.

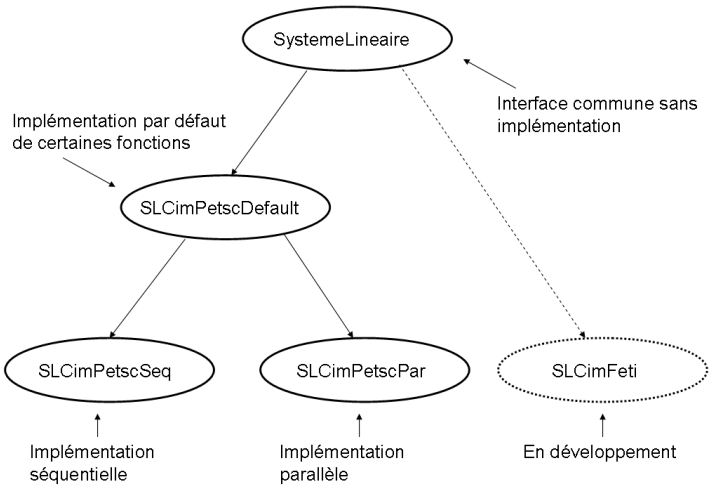

Figure 2. *Hiérarchies de classes qui dérivent et implémentent « Système Linéaire » dans un contexte séquentiel et parallèle* via *PETSc (méthode Feti encore en développement)*

L'efficacité de ce type d'interface dépend principalement de la facilité à l'utiliser. Pour cela, un nombre restreint de fonctions bien documentées, associées à une bonne évolutivité et à l'ajout de nouvelles fonctionnalités (par exemple la résolution de systèmes non symétriques) jouent un rôle important.

2.2. *Un solveur linéaire générique*

Pour augmenter la partie parallèle du code global, afin d'améliorer les accélérations, il est nécessaire de paralléliser le calcul et l'assemblage des contributions locales tout en conservant son caractère « invisible ». Premièrement, nous devons stocker les données dans un contexte parallèle tout en gardant cette distribution transparente pour le développeur. Comme précédemment, nous avons caché l'implémentation derrière une interface appelée « Champ ». Cette interface

peut être utilisée pour stoker la vitesse, la pression, mais également l'orientation moléculaire et même la gravité. Là aussi, il existe deux implémentations principales : une séquentielle, l'autre parallèle qui est distribuée sur l'ensemble des processeurs. Ces deux implémentations sont elles-mêmes dérivées en fonction du degré d'interpolation : P0, P1... et de la nature des inconnues (nodales, par élément). La fonction la plus intéressante de « Champ » est « SetLocal » qui, à partir d'un indice d'élément, permet de construire une représentation élémentaire de ce dernier « SimplexChamp ». La même fonction existe pour le « Maillage », pour créer un « Simplex ». Cette fonction est, évidemment, implémentée dans les deux contextes : séquentiel et parallèle. Cette structure permet de découpler tout travail, à effectuer localement, de l'implémentation globale utilisée.

L'utilisation d'un solveur éléments finis est divisée en trois étapes :

– créer et affecter les représentations locales des données nécessaires au calcul des contributions élémentaires « Solveur »,

– calcul des contributions locales en dérivant l'interface « SimplexSolveur »,

– assemblage des contributions dans le système global « Solveur » et « SystemeLineaire ».

Dans la première étape, l'interface « Champ » permet la représentation globale des variables du problème, mais également la création et l'affectation de sa représentation sur un élément. Lors de la seconde étape, on se place sur un élément et on calcule les contributions liées au problème physique à résoudre. L'implémentation de « SimplexSolveur » nécessite des connaissances du problème à traiter, mais aucune connaissance liée au parallélisme. Ce développement peut être fait sans difficulté par un développeur « classique ». Finalement, la dernière étape consiste à utiliser l'interface « SystemeLineaire » pour assembler chaque contribution élémentaire dans le système global.

Ceci signifie que, pour introduire un nouveau problème physique, le développeur ne travaille que dans une classe dérivée de « SimplexSolveur » dans laquelle il doit implémenter son problème, pour obtenir finalement une application totalement parallèle le traitant. Il est donc possible à un développeur classique d'augmenter la proportion de code parallèle, à l'opposé de la démarche classique où ce développeur augmente la fraction séquentielle du code. Ici, l'ajout de nouveaux développements (nouveaux problèmes physiques) augmente l'efficacité de l'application parallèle.

3. Applications à des écoulements viscoélastiques

3.1. *Résolution numérique*

Le comportement viscoélastique a été considéré à travers une loi de comportement du type Pom-Pom, multimode (McLeish *et al.,* 1998, Blackwell *et al.,* 2000, Inkson *et al.,* 1999). Le tenseur des contraintes σ est donné par :

$$\sigma = -pI + 2\eta\varepsilon(v) + \sum_{i=1}^{n} \tau_i = -pI + 2\eta\varepsilon(v) + \sum_{i=1}^{n} G_i (3\lambda_i^2 S_i - I), \quad [1]$$

où η est une viscosité de solvant (considérée petite), p est la pression hydrostatique, $\varepsilon(v)$ est la vitesse de déformation, G_i le module de relaxation du mode i, λ_i l'étirement du mode et S_i l'orientation moléculaire (tensorielle). Pour déterminer la vitesse et la pression de l'écoulement, nous résolvons un problème de Stokes perturbé par un terme élastique ($\sum G_i (3\lambda_i^2 S_i - I)$), où les S_i et λ_i sont des nouvelles inconnues, provenant de la loi de comportement choisie.

L'orientation et l'étirement de chaque mode sont déterminés par la résolution des équations d'advection suivantes :

$$\begin{aligned} &\frac{d\lambda_i}{dt} + \lambda_i [\varepsilon(v) : S_i] + \frac{1}{\theta_{S_i}} (\lambda_i - 1) = 0, \\ &\frac{dS_i}{dt} - \left(\nabla v + {}^t\nabla v\right) + 2S_i [\varepsilon(v) : S_i] + \frac{1}{\theta_{\lambda_i}} \left(S_i - \frac{I}{3}\right) = 0. \end{aligned} \quad [2]$$

où θ_{Si} est le temps de relaxation de l'orientation du mode i et $\theta_{\lambda i}$ le temps de relaxation de l'étirement du même mode.

La méthode de résolution du problème à quatre champs a été décrite dans un article précèdent (Silva *et al.,*). On retiendra seulement que le nombre d'inconnues total par incrément est pour un maillage 3D :

(4 × NombreNoeuds) + ((6 + 1) × NombreElements) × NombreModes

3.2. *Validations et performances en 2D*

Ce premier test a pour but de valider l'implémentation du modèle viscoélastique, ainsi que l'ensemble des développements effectués pour obtenir un code parallèle robuste et facilement utilisable. Le cas présenté est un benchmark classique d'écoulement viscoélastique dans une contraction 2D (Beraudo *et al.,* 1998, Sirakov, 2000).

Les dimensions de la demi géométrie sont de 10 mm en entrée et de 1,25 mm en sortie soit une contraction de 1 pour 8. Un débit correspondant à une vitesse moyenne de lmm/s est imposé sur le plan d'entré placé à x/h = -20, alors qu'une contrainte normale de zéro est imposée en sortie à x/h = 20. Le polymère utilisé est un polyéthylène de basse densité (LPDE) modélisé par un modèle Pom-Pom à 8 modes dont les caractéristiques à 160 °C sont données dans le tableau 1. Le reste

des conditions aux limites sont du type contact collant et vitesse normale nulle sur le plan de symétrie en y = 0.

Mode	$\theta_{b,i}(s)$	G_i(MPa)	q_i	$\theta_{b,i}(s)/\ \theta_{S,i}(s)$
1	6,25E-04	6,327	1	2
2	5,35E-03	5,340E-02	1	2
3	2,85E-02	2,700E-02	1	2
4	1,55E-02	1,400E-01	1	2
5	8,91E-01	7,250E-03	3	2
6	4,58E+00	2,660E-03	4	2
7	2,34E+01	5,660E-04	8	1,35
8	1,18E+02	6,183E-05	12	1,9

Tableau 1. *Paramètres non linéaires du LDPE à 160°C*

Les calculs ont été effectués sur 2 maillages différents (voir figure 3) : un maillage grossier (7 478 nœuds et 14 954 éléments) et un maillage fin (28 470 nœuds 56 938 éléments).

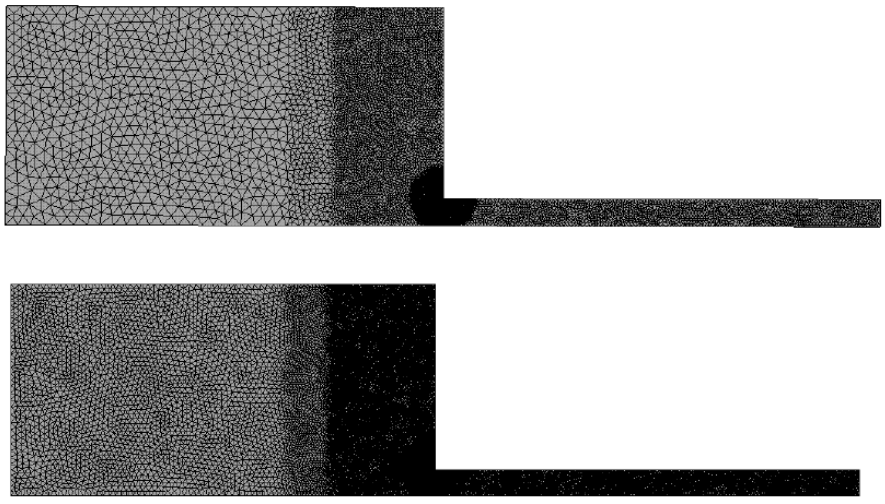

Figure 3. *Deux maillages tests : un maillage grossier qui contient 7 478 nœuds et 14 954 éléments ; et un maillage fin de 28 470 nœuds et 56 938 éléments*

Pour analyser les performances parallèles des solveurs développés, nous avons exécuté ce cas-test sur les 2 maillages avec l'utilisation de 1 à 16 processeurs. Ce cas-test a été exécuté sur une ferme de PC composé de PentiumIII à 1 Ghz avec 512 Mo de mémoire et un réseau de communication rapide Myrinet.

Le nombre de pas de temps à dimension variable pour atteindre l'état stationnaire est de 10 000 et les premiers pas de temps sont de l'ordre de 10^{-4}s. Les tableaux suivants représentent le temps passé par l'application à partitionner le domaine, calculer l'écoulement (à chaque pas de temps), sauvegarder les résultats (tout les 100 pas de temps) et le temps total. La seule opération qui reste séquentielle est l'écriture des fichiers résultats qui consiste à rassembler les données sur un seul processeur avant de les écrire.

Nb processeurs	1	2	4	8	16
Partionnement (s)	0,6	1,3	2,2	3,5	5,0
Calcul (s)	195123	102495	53140	28512	17048
Stockage (s)	641	684	683	680	682
Total (s)	195765	103180	53824	29195	17735
Calcul : Speed-Up (efficacité)	1 (1)	1,903 (0,952)	3,672 (0,928)	6,843 (0,855)	11,445 (0,715)
Total : Speed-Up (efficacité)	1 (1)	1,897 (0,948)	3,657 (0,932)	6.705 (0,838)	11.038 (0,690)

Tableau 2. *Temps d'exécution des modules du code sur le maillage grossier*

Nb processeurs	1	2	4	8	16
Partitionnement (s)	2,3	9,3	14,5	16,9	23,1
Calcul (s)	1309432	659500	356803	178374	93458
Stockage (s)	2616	2636	2621	2619	2624
Total (s)	1312051	662146	359439	181011	96106
Calcul : Speed-Up (efficacité)	1 (1)	1,986 (0,993)	3,670 (0,918)	7,341 (0,917)	14,011 (0,876)
Total : Speed-Up (efficacité)	1 (1)	1,981 (0,990)	3,650 (0,912)	7,248 (0,906)	13,652 (0,853)

Tableau 3. *Temps d'exécution des modules du code sur le maillage fin*

Nous observons pour les deux maillages une réduction importante du temps de calcul avec l'augmentation du nombre de processeurs utilisé. Le temps d'écriture des fichiers résultats reste constant (il s'effectue en séquentielle). L'unique partie qui augmente avec le nombre de processeurs est la création de la partition du maillage. On remarque, tout de même, que le temps consacré au partitionnement reste largement négligeable vis-à-vis du reste. Au final, les accélérations obtenues sont très bonnes, en particulier sur le maillage fin, où l'on passe d'un temps de simulation de **15 jours 4 heures et 28 minutes** à **1 jour 2 heures et 42 minutes** soit une réduction voisine de 14 pour 16 processeurs (ceci même si le temps passé à stocker les résultats reste invariant à 43 minutes). Il faut souligner que le calcul prend en compte les 8 modes du modèle Pom-Pom : 8 orientations (tenseur) et 8 étirements (scalaire) ainsi que la résolution du problème de Stokes généralisé en vitesse/pression, ce qui revient à calculer 1,5 million d'inconnues (au travers 17 systèmes matriciels) par pas de temps.

3.3. *Ecoulement viscoélastique*

a) Autour d'une sphère

On considère l'écoulement tridimensionnel d'un polymère autour d'une sphère, et confiné entre quatre plans (figure 4). Pour des raisons de symétrie, seulement le quart de la géométrie a été considéré. Sur le plan d'entrée, une vitesse selon l'axe normal a été imposée, de façon à obtenir un nombre de Weissenberg de 11,6. Le comportement viscoélastique a été ajusté sur une loi de comportement du type Pom-Pom, avec un seul mode.

Le tableau ci-dessous résume les données du calcul effectué. Les temps de calcul obtenus montrent la faisabilité des calculs réalisés (9 milliards d'inconnues déterminés *via* 16 660 systèmes matriciels comportant de 1 à 6 millions de degrés de liberté résolus en à peine 10 heures !).

Maillage **209 953** nœuds et **1 221 448** éléments
Nombre d'incréments 980
Nombre d'inconnues par incrément / total 9 389 948/ 9 202 149 040
Nombre de processeurs 32
Temps de calcul 9 heures et 39 minutes

Tableau 4. *Principales valeurs caractéristiques pour le calcul d'un écoulement viscoélastique*

Au niveau qualité de ces résultats, la figure 4 montre la norme de la vitesse, l'orientation moyenne des chaînes moléculaires, et la première différence de contraintes normales dans l'écoulement. On observe une bonne concordance avec les profils obtenus dans la littérature (Sirakov, 2000).

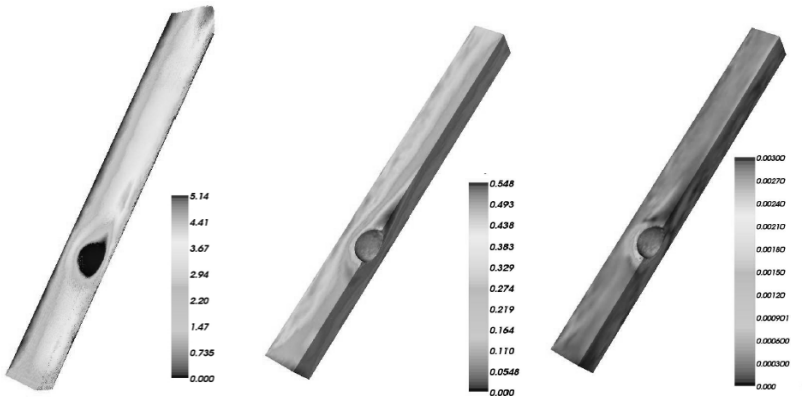

Figure 4. *Caractéristiques de l'écoulement, de gauche à droite : norme de la vitesse, orientation moyenne des chaînes moléculaires et première différence de contraintes normales*

b) Sur une contraction

Nous considérons ici l'écoulement dans une filière relativement complexe et totalement 3D (figure 5). Les conditions aux limites sont similaires au cas précédent : le débit est imposé en entrée et correspond à une vitesse moyenne de l0 mm/s, alors qu'une contrainte normale nulle est imposée en sortie. Des conditions aux limites du type contact collant sont imposées partout ailleurs.

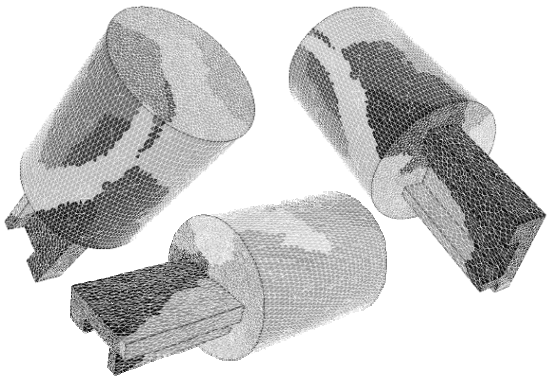

Figure 5. *Géométrie et partition en 8 sous-domaines de la contraction 3D*

Le matériau utilisé est le même LDPE que dans le cas de la contraction 2D (tableau 1), mais ici avec uniquement les 4 premiers modes. Pour ce qui est du calcul, nous effectuons 1 000 pas de temps pour obtenir l'état stationnaire, ce qui correspond à un temps de simulation de 30 heures sur 8 processeurs avec un maillage à 65 651 nœuds et 380 849 éléments (environ 3 millions de degrés de liberté). Ce temps de calcul reste tout à fait raisonnable. On remarquera que ce calcul 3D n'a pas pu être exécuté sur un seul processeur, en raison d'une mémoire insuffisante pour stocker le système linéaire à résoudre.

Le haut de la figure 6 représente les lignes de courant lorsque l'état stationnaire est atteint. On observe le caractère 3D de l'écoulement. Le bas de la figure montre sur un plan de coupe la biréfringence (à gauche) ainsi que la distribution du nombre de Weissenberg local (à droite). La première différence de contrainte normale atteint son maximum au niveau de la contraction comme dans le cas 2D. Même si en moyenne le nombre de Weissenberg reste petit, il peut atteindre localement 135.

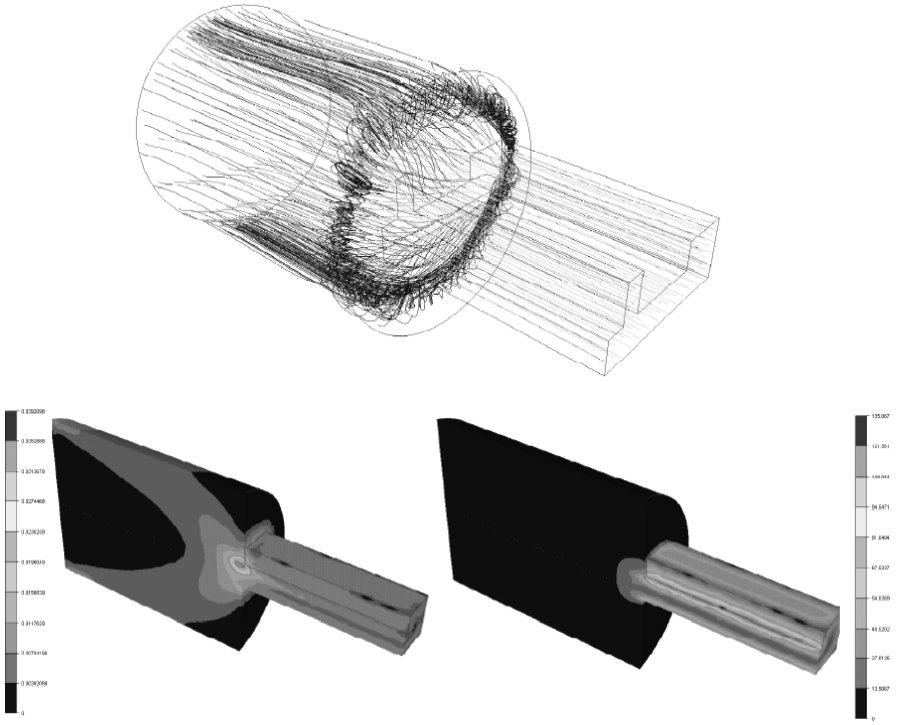

Figure 6. *Lignes de courant à l'état stationnaire ainsi qu'un « cliché » de biréfringence et la distribution du nombre de Weissenberg local sur une coupe transverse*

4. Applications aux équations de Navier-Stokes avec surface libre

4.1. *Résolution numérique*

On s'intéresse maintenant à des écoulements gouvernés par les équations de Navier-Stokes :

$$\begin{cases} \rho \dfrac{dv}{dt} - \nabla.[2\eta\varepsilon(v)] + \nabla p = \rho g \\ \nabla v = 0 \end{cases} \quad [3]$$

Ce système d'équations en vitesse/pression est linéarisé, puis résolu par une méthode éléments finis mixtes basée sur le MINI-élément P1+/P1 (Coupez, 1996). On s'intéresse plus particulièrement ici à l'évolution des interfaces séparant les différentes phases du fluide. Nous considérons donc deux phases dans un réservoir :

– l'une dense, un liquide qui s'écroule : $\rho_1 = 1\ 000$ kg/m^3, $\eta_1 = 10$ Pa.s

– l'autre moins dense : $\rho_2 = 10$ kg/m^3, $\eta_2 = 10$ Pa.s

Ces deux phases sont représentées par une fonction α continue de type « level-set » telle que :

– $\alpha > 0$ dans la phase 1

– $\alpha < 0$ dans la phase 2

– $\alpha = 0$ à l'interface.

La fonction α est déterminée par résolution d'une équation de transport, donnée par :

$$\frac{d\alpha}{dt} = \frac{\partial \alpha}{\partial t} + v.\nabla \alpha = 0 \quad [4]$$

où v est le champ de vitesse, solution des équations de Navier-Stokes précédentes. Cette équation est résolue par une discrétisation éléments finis P1 par élément et continue, stabilisée par une méthode de type SUPG (Adalsteinsson *et al.,* 1995, Sussman *et al.,* 1996).

Le fait d'avoir deux phases nous oblige à homogénéiser la densité ρ et la viscosité η utilisées lors de la résolution du problème mécanique [3]. Ces deux grandeurs seront prises constantes par élément, et l'homogénéisation se fait à l'aide de la fonction α, au travers d'une loi de mélange pondérée :

$$\begin{cases} \eta = f(\alpha)\eta_1 + (1 - f(\alpha))\eta_2 \\ \rho = f(\alpha)\rho_1 + (1 - f(\alpha))\rho_2 \end{cases} \quad [5]$$

où f est une fonction comprise entre 0 et 1.

Les deux solveurs éléments-finis (Navier-Stokes et transport) ont été implémentés suivant la méthode de parallélisation présentée dans la première section de ce document. Le calcul des matrices et seconds membres locaux est fait par dérivation d'une classe d'interface. Le premier solveur entraîne la résolution d'un système linéaire comportant 4 inconnues par nœud, tandis que le second n'en comporte qu'une seule.

4.2. *Ecroulement d'une colonne de liquide sur un plan horizontal rigide*

Ce cas-test est très souvent utilisé pour valider les modèles d'étude des écoulements instationnaires très peu visqueux à surface libre. On considère une colonne d'eau au repos, retenue par une membrane de papier ciré (imperméable) extrêmement fine, dans un réservoir parallélépipédique transparent (figure 7). A un instant donné, la membrane est libérée et la colonne d'eau s'écroule.

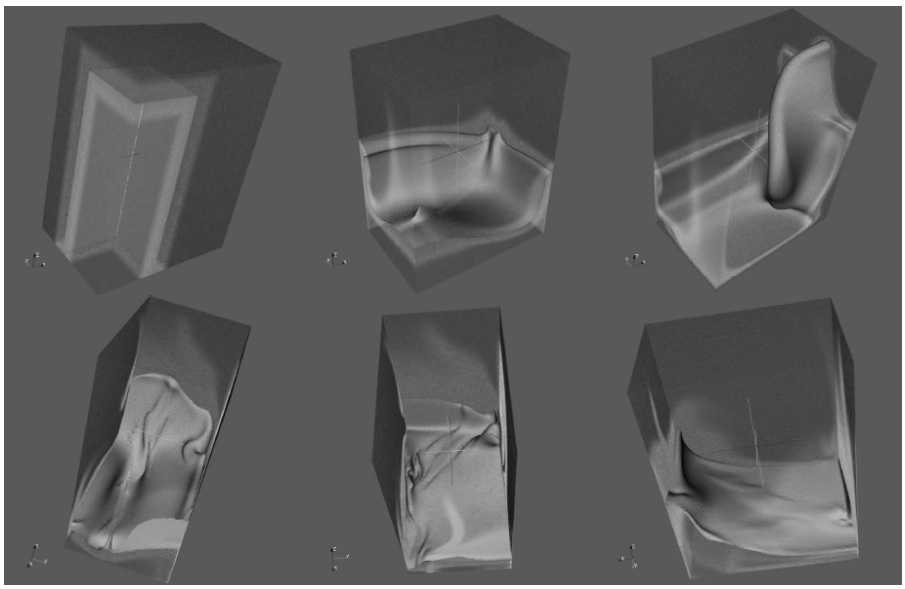

Figure 7. *Evolution de la surface libre à l'intérieur du réservoir, à six instants donnés. Simulation réalisée sur un maillage à plus de 2 millions de nœuds*

Une première série de calculs a été réalisée pour évaluer les accélérations obtenues. Le maillage retenu est un maillage à *125 972 nœuds* et *716 523 éléments*. Ce maillage permet de tester l'application de 1 à 32 processeurs. Les accélérations obtenues permettent de juger la qualité de l'application dans l'utilisation maximale des machines de calcul à plusieurs processeurs. Les résultats présentés dans le

tableau ci-après ont été obtenus sur notre deuxième ferme de PC (Pentium IV 2,8 Ghz, 2 Go de mémoire et réseau rapide Myrinet) en utilisation courante de chaque processeur (non exclusive). Si nous considérons la simulation de tout l'écroulement, nous passons d'une durée de *1 jour 18 heures et 36 minutes* sur un seul processeur à *1 heure 19 minutes* sur 32 processeurs, ce qui représente une accélération de 32.4 sur 32 processeurs. De plus, en ne prenant en compte que la partie résolution de notre application, nous atteignons une accélération de 40 sur 32 processeurs. Cette super linéarité est due à l'utilisation non exclusive de notre ferme de PC mais également aux effets de cache mémoire.

Nb processeurs	1	2	4	8	16	32
Global (s)	153383	65876	32957	16217	8162	4735
Assemblage NS (s)	21460	11353	5902	3114	1731	918
Résolution NS (s)	115023	45129	21822	10035	4379	2224
Assemblage T (s)	6447	3630	1997	1056	603	364
Résolution T (s)	144165	60620	29912	14281	6746	3525
Non mesuré (s)	9218	52556	3045	1936	1416	1210
Solveurs : Speed-Up (Efficacité)	1 (1)	2,38 (1,19)	4,82 (1,20)	10,1 (1,26)	21,4 (1,34)	40,9 (1,28)
Global : Speed-Up (Efficacité)	1 (1)	2,32 (1,16)	4,65 (1,16)	9,46 (1,18)	18,8 (1,18)	32,4 (1,01)

Tableau 5. *Temps passés dans les principales parties du code de simulation, ainsi que les accélérations observées (sur la durée totale de la simulation et sur la partie résolution par éléments finis)*

Une fois ces tests d'efficacité parallèle effectués, nous présentons ici le cas d'écroulement, mais cette fois-ci réalisé sur un maillage beaucoup plus fin. Le maillage utilisé comporte *2 148 355 noeuds* et *12 418 472 éléments*. Le calcul complet de l'écroulement a nécessité *600 incréments*, et chaque incrément fait intervenir la résolution de deux systèmes linéaires : l'un de 8.6 millions de degrés de libertés, l'autre de 2.15 millions, soit la détermination totale de *6 milliards 450 millions* d'inconnues à travers *1 200* systèmes matriciels. Le temps de cette simulation a été de *5 jours et 1 heure* sur 32 processeurs de notre ferme de PC.

4.3. *Calcul sur grilles et déversement d'un jerricane*

Ce dernier résultat montre l'utilisation de ce code dans un contexte grilles de calcul. Une grille de calcul est un ensemble de clusters reliés entre eux *via* internet. Dans le cas de notre grille « MecaGrid », l'ensemble forme un calculateur parallèle très hétérogène, pour la puissance de CPU (Pentium III lGhz et Pentium IV 2 Ghz) mais également pour le réseau (1 Gb/s entre les nœuds nina et moins de 10 Mb/s entre deux processeurs de deux centres différents).

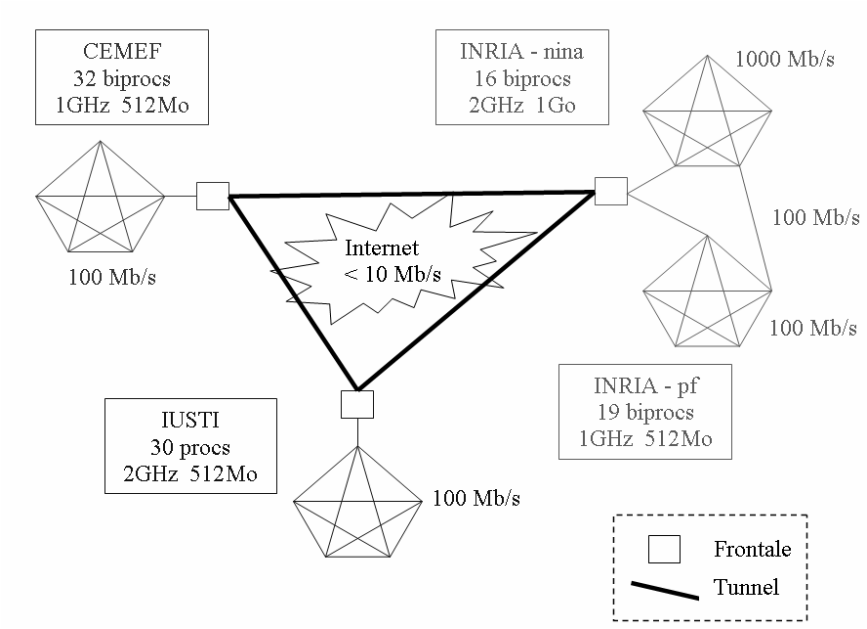

Figure 8. *Caractéristiques techniques de la grille de calcul « MecaGrid » utilisée*

L'application présentée montre l'écoulement d'un fluide par le goulot d'un jerricane. Au début de la simulation, le fluide se trouve totalement dans le jerricane qui est alors renversé : son goulot ouvert est dirigé vers le bas. Le fluide commence alors à se déverser par l'ouverture, tandis que l'air s'engouffre à son tour dans le jerricane sous forme de bulles. Ainsi, la cavité se vide peu à peu de son contenu initial et se remplit d'air, jusqu'à ce que le fluide soit totalement déversé.

La totalité du domaine est maillé avec 500 000 nœuds et 2 750 000 éléments. Les dimensions du goulot sont de (20 x 20 x 10) centimètres. Les cavités inférieures et supérieures sont de même taille, soit (1 x 1 x 0.40) mètres. Le solveur pour l'écoulement ne permet pas un pas de temps supérieure à 0.01 seconde. Un nombre très important d'incréments de temps et donc nécessaire (1 000 incréments

pour 10 secondes de simulation). Les paramètres utilisés sont les suivants : le fluide a une viscosité de 10 Pa.s et une masse volumique de 1 000 kg/m^3, l'air a une viscosité de 1 Pa.s et une masse volumique de 1 kg/m^3. Les conditions aux limites sont de type collant, c'est-à-dire qu'une vitesse nulle est imposée sur toutes les surfaces de la géométrie.

Sur la figure 9 est représentée l'interface entre le fluide et l'air aux temps t = 0 s, t = 1 s, t = 3 s, t = 6 s, t = 11 s et t = 18 s. Alors que des bulles d'air montent dans le jerricane (la cavité du haut), le fluide coule dans une cuvette (la cavité du bas) à travers le goulot du jerricane. Le niveau du fluide baisse donc dans le jerricane et monte dans la cuvette.

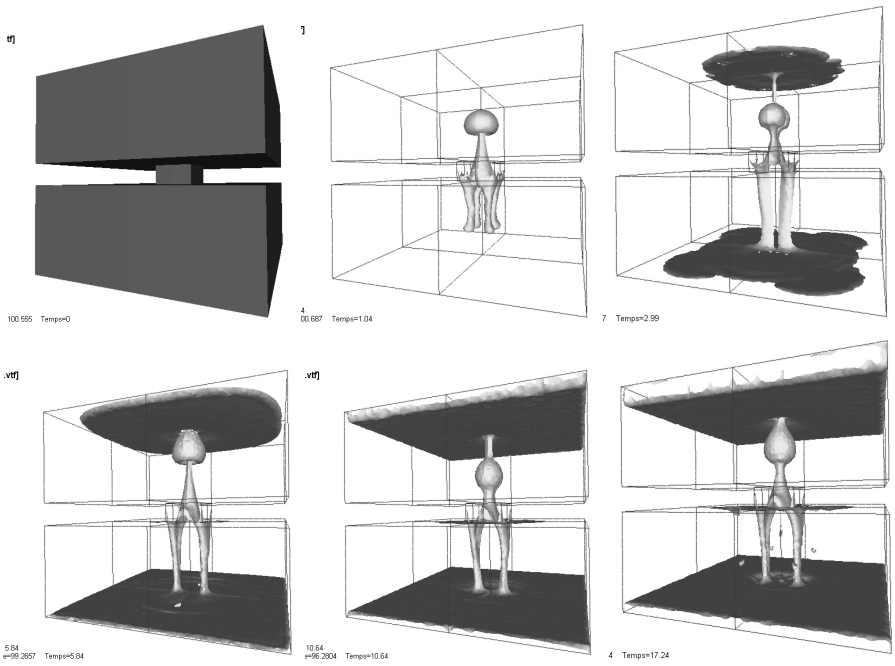

Figure 9. *Evolution de la surface libre lors du déversement d'un jerricane*

Cette application est intéressante car, premièrement, elle induit des mouvements de surface libre complexes, et deuxièmement, sa géométrie facilite un partitionnement adapté à la grille de calcul. En effet, l'utilisation d'un partitionneur hétérogène permet d'adapter automatiquement la partition en tenant compte de la vitesse de calcul des CPU mais également de la rapidité du réseau d'interconnections. Ici, l'existence du goulot (partie étroite de la cavité) permet – si le partitionnement est bien réalisé – de diminuer considérablement les communications intersites. Pour cela, le partitionneur a attribué la cavité du haut à

un site, et celle du bas à un autre site. De cette manière, c'est seulement à l'endroit de la partie étroite que les communications intersites s'opèrent, et deviennent donc bien moins importantes que les autres communications plus performantes.

Un total de 15 processeurs pris sur les 3 clusters nina, pf et iusti de MecaGrid sont choisis pour réaliser cette application. Il reste à déterminer combien de processeurs sont attribués sur chaque cluster. Chacun des 2 sites s'occupe d'une partie de la cavité afin de placer l'interface entre les deux zones du maillage au niveau de la contraction : à l'INRIA, 4 nina et 5 pf ont la partie supérieure du maillage ; à l'IUSTI, 6 processeurs ont la partie inférieure.

Les mesures de temps suivantes (tableau 6) sont relevées après 100 incréments de temps, avec des partitions homogènes puis optimisées, et les préconditionneurs ILU(0) et ILU(l).

	ILU(0)		ILU(l)	
Nb processeurs	15	15	15	15
Partition	Homogène	Optimisée	Homogène	Optimisée
Nb itérations	10 702	10 847	2 210	1904
NS assemblage	2 926	1743	3 680	2 455
NS résolution	526 659	212 962	235 504	107 899
Alpha assemblage	1 846	1 808	2 357	2 896
Alpha résolution	231	62	258	113
Total (s)	531662	216 575	241 799	113 364

Tableau 6. *Comparaison des temps d'exécution sur la grille suivant le partitionneur homogène ou optimisé et suivant le préconditionneur de type ILU(0) ou ILU(l) utilisé*

Avec ILU(0), le gain de temps apporté par le partitionnement optimisé est de 60 %, ce qui est un excellent résultat. Le temps d'exécution est passé de 6 jours et 4 heures à 2 jours et 12 heures. On observe également qu'avec le préconditionneur ILU(l), les résultats sont bien meilleurs : le temps d'exécution est réduit d'un facteur supérieur à 2 par rapport à celui obtenu avec ILU(0). Cette méthode fait diminuer le nombre d'itérations nécessaires à la convergence des systèmes linéaires d'un facteur pratiquement égal à 5. Par contre, le coût d'une itération augmente, passant de 49 secondes dans le cas homogène à 106 secondes. Il n'en reste pas moins qu'utiliser ILU(l) est bien plus avantageux que ILU(0).

Au final si l'on ajoute les deux optimisations : partitionnement hétérogène + ILU(l), le temps d'exécution est passé de *6 jours et 4 heures à 1 jour et 8 heures*, ce qui représente une division du temps de calcul par *4,7*.

Cette réduction spectaculaire s'explique par la géométrie utilisée et la façon dont on la partitionne. Avec une géométrie purement cubique, l'apport d'une partition optimisée ne joue quasiment que sur une meilleure répartition de la masse de calcul, car les communications pénalisantes à l'interface des partitions ne peuvent pas être fortement diminuées. Or, en plus de répartir équitablement la masse de calcul, une géométrie telle que celle utilisée ici permet justement de diminuer considérablement les communications intersites très pénalisantes en « coupant » le maillage dans un endroit où il y a peu de nœuds. Au total, l'addition des deux techniques d'optimisation (partitionnement hétérogène et ILU(l)) fait gagner presque 80 % sur les temps de calcul de MecaGrid.

5. Conclusions

Les résultats présentés montrent un très bon comportement parallèle de l'application : l'efficacité, vis-à-vis du code parallèle exécuté sur un seul processeur, reste à peu près constante et proche de 1, malgré le partitionnement du maillage ainsi que des lectures et écritures séquentielles des fichiers de données. Le code parallèle a d'ailleurs totalement remplacé sa version séquentielle (performance identique sur un seul processeur). L'utilisation d'une programmation orientée objet nous a permis de cacher toute instruction parallèle derrière une interface commune entre version parallèle et séquentielle. De ce fait, la vitesse de développement de nouvelles fonctionnalités se trouve augmentée (implémentation dans un contexte séquentiel et validation complète sur des calculateurs parallèles). Cette parallélisation discrète a été appliquée avec succès à la simulation d'écoulements viscoélastiques, ainsi qu'au calcul de l'évolution d'une surface libre régie par les équations de Navier-Stokes. Dans les deux cas, l'utilisation d'un calculateur parallèle nous a permis de réaliser des simulations qui auraient nécessité plusieurs dizaines de Go de mémoire RAM et de nombreux jours de calcul. L'exemple de l'écroulement d'une colonne de liquide aurait nécessité près de 40 Go de mémoire et au moins 5 mois de calcul sur une machine séquentielle à base de PIV 2.8 Ghz, ce qui montre tout l'intérêt de notre méthode permettant de réaliser cette même simulation en 5 jours.

6. Bibliographie

Adalsteinsson D., Sethian J.A., "A fast level set method for propating interfaces", *Journal of Computational Physics,* vol. 118, 1995, p. 269-277.

Arnold D.N., Brezzi F., Fortin M., "A stable finite élément for Stokes equations", *Calcolo*, vol. 21, 1984, p. 337-344.

Basermann A., Clinckemaillie J., Coupez T., Fingberg I., Digonnet H., Ducloux R., Gratien J.-M., Hartmann U., Lonsdale G., Maerten B., Roose D., Walshaw C, "Dynamic load balancing of finite élément applications with the drama library", *Applied Mathematical Modelling*, vol. 25, n° 2, 2000, p. 83-98.

Beraudo C., Fortin A., Coupez T., Demay Y., Vergnes B., Agassant J.F., "A finite élément method for Computing the flow of multi-mode viscoelastic fluids: Comparison with experiments", *Journal of Non-Newtonian Fluid Mechanics*, vol. 75, 1998, p. 1-23.

Blackwell R.J., McLeish T.C.B., Harlen O.G., "Molecular drag-strain coupling in branched polymer melts", *Journal of Rheology*, vol. 44, 2000, p. 121-136.

Coupez T., Stable-stabilized finite élément for 3d forming calculation, Rapport interne, 1996, CEMEF-ENSMP.

Digonnet H., Coupez T., "Object-oriented programming for fast-and-easy development of parallel applications in forming processes simulation", *Second MIT Conférence on Computational Fluid and Solid Mechanics*, Massachussets Institute of Technology, Elsevier, 2003, p. 1922-1924.

Digonnet H., Coupez T., « Calcul parallèle en mise en forme des matériaux », *XV^e Congrès Français de Mécanique*, 2003, Nice.

Inkson N.J., McLeish T.C.B., Harlen O.G., Groves D.J., "Predicting low density polyethylene melt rheology in elongational and shear flows with "pom-pom" constitutive equations", *Journal of Rheology*, vol. 43, 1999, p. 873-869.

McInnes L.C., Balay S., Gropp W. D., Smith B. F., Petsc users manual, Rapport technique ANL-95/11- Revision 2.1.3, 2002, Argonne National Laboratory.

McLeish T.C.B., Larson R.G., "Molecular constitutive équations for a class of branched polymers: the pom-pom model", *Journal of Rheology*, vol. 42, 1998, p. 81-110.

Silva L., Valette R., Coupez T., "3D MFE for compressible viscoelasticity with moving free surfaces", *Journal for Non-Newtonian Fluid Mechanics*, soumis.

Sirakov Y., On the Steady Flow of Compressible Viscous Fluid and its Stability with Respect to Initial Disturbance, Thèse de doctorat, Ecole des Mines de Saint-Etienne, 2000.

Sussman M., Smereka P., Osher S., "A level set méthode for Computing solutions to incompressible two-phase flow", *Journal of Computational Physics*, vol. 114, 1994 p. 146-159.