Résolution numérique de l'équation des ondes par une méthode d'éléments finis d'ordre élevé sur calculateur parallèle

Pascal Havé, Michel Kern, Christophe Lemuet

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Résolution numérique de l'équation des ondes par une méthode d'éléments finis d'ordre élevé sur calculateur parallèle

Pascal Havé — Michel Kern — Christophe Lemuet

N° 4381

Février 2002

THÈME 4

ISSN 0249-6399 ISRN INRIA/RR--4381--FR

INRIA
ROCQUENCOURT

Résolution numérique de l'équation des ondes par une méthode d'éléments finis d'ordre élevé sur calculateur parallèle

Pascal Havé , Michel Kern , Christophe Lemuet

Thème 4 —Simulation et optimisation
de systèmes complexes
Projet estime

Rapport de recherche n° 4381 —Février 2002 —36 pages

Résumé : Nous présentons une méthode d'éléments finis d'ordre élevé permettant de résoudre l'équation des ondes acoustiques dans le domaine espace–temps par un schéma explicite. Nous construisons un nouvel élément fini permettant la condensation de masse, sans perte de précision. Cette construction est basée sur la recherche d'une formule de quadrature symétrique, dont les noeuds forment un élément fini. Après discrétisation en temps par une méthode explicite, nous obtenons une méthode de résolution parallèle, basée sur une décomposition en sous-domaines. Les sous-domaines peuvent être non-structurés. Nous décrivons la mise en oeuvre de la méthode dans un logiciel en `C++`, utilisant la bibliothèque de communications `MPI` pour la portabilité. La méthode est validée sur des exemples simples, et également dans des cas de géométrie complexe. Nous donnons enfin des résultats de performance parallèle

Mots-clés : équation des ondes, éléments finis, calcul parallèle, décomposition de domaine

Ce travail a été réalisé dans le cadre d'une convention de recherche Dassault–INRIA

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 —Télécopie : +33 1 39 63 53 30

Numerical Solution of the Wave Equation Using Higher Order Finite Elements on Parallel Computers

Abstract: We present a higher order finite element method for solving the acoustic wave equation in the space–time domain, using an explicit scheme. We build a new finite element allowing mass lumping without loss of accuracy. This construction is based on a symmetric quadrature formula, whose nodes form a finite element. After discretization in time using an explicit scheme, we are led to an parallel solution algorithm, based on a subdomain decomposition. A `C++` code implementing the method, and using the `MPI` communication library for portability, has been written. The method is validated against simple examples, and also in complex geometries. Finally we present parallel performance results.

Key-words: wave equation, finite element, parallel computing, domain decomposition

1 Introduction

Nous présentons dans ce rapport un logiciel parallèle pour simuler les phénomènes de propagation des ondes acoustiques dans des milieux hétérogènes complexes. Ce logiciel est basé sur de nouvelles méthodes d'éléments finis d'ordre élevé permettant la condensation de masse sans perte de précision, tout en gardant un schéma explicite. Il a également été conçu pour tirer parti des possibilités du calcul parallèle sur les calculateurs à mémoire partagée.

Les problèmes de propagation d'ondes font souvent intervenir des domaines ou soit la géométrie elle même est complexe, soit les propriétés physiques varient d'une façon qu'il peut être difficile à décrire par un maillage structuré. Citons les deux exemples qui ont motivé ce travail :

Aero-acoustique La réduction du bruit de jet d'un avion au décollage est un sujet d'actualité. Une méthode simple à mettre en oeuvre a été proposée par Lighthill [14], et consiste à découpler le calcul du champ de vitesse (par une résolution des équations de Navier-Stokes) de celui de la pression acoustique, qui nous concerne ici. Dans cet exemple, la géométrie du domaine peut être régulière où non, mais le champ de vitesse intervenant dans l'équation des ondes sera *a priori* quelconque.

Géophysique La prospection pétrolière par les méthodes sismiques utilise des techniques de plus en plus sophistiquées. La géométrie du sous-sol est décrite par un modeleur (tel que `http://www.ensg.u-nancy.fr/GOCAD/`), et on cherche à prendre en compte les effets de la topographie. Dans cet exemple, tant la géométrie elle même que le champ de vitesses peuvent varier de façon complexe, présentant des discontinuités, ou des zones avec des gradients horizontaux ou latéraux.

Dans ces deux exemples, il n'est pas naturel de chercher à plaquer la géométrie ou les coefficients physiques sur une grille régulière. Les éléments finis ont été conçus pour traiter ce type de problème. Ils autorisent une géométrie théoriquement arbitraire (avec une restriction importante en pratique : il faut être capable de créer le maillage du domaine considéré, ce qui peut être difficile, en particulier en dimension 3). Ils permettent également de prendre en compte des variations brutales ou continues des coefficients physiques. Le revers de la médaille est un coût supplémentaire par rapport au méthodes de différences finies, du à la plus grande généralité de la programmation.

Les domaines considérés dans les deux exemples ci-dessus sont le plus souvent non-bornés, ce qui veut dire en pratique que le domaine de calcul sera de grande dimension. Le paramètre le plus important est la *longueur d'onde*, qui peut être fixée par la source ou la condition initiale. Des domaines de quelques dizaines de longueurs d'onde sont un minimum pour représenter correctement les phénomènes. Par ailleurs il est bien connu [1, 18] qu'une bonne résolution des phénomènes ondulatoires requiert 10 points par longueur d'ondes (moins pour les méthodes d'ordre plus élevé, nous reviendrons sur ce point). On voit alors qu'un modèle typique aura environ 100 points dans chaque dimension, soit un million de degrés de libertés.

Enfin, la simulation dans un domaine de grande taille demande souvent des temps longs (quelques dizaines de périodes), et conserver la précision aux temps longs demande un maillage (spatial et temporel) plus précis. Il a été montré (au moins en deux dimensions) qu'une façon économique d'atteindre cette précision était de monter en ordre d'approximation au-delà des méthodes d'ordre « 2-2 » habituelles.

Toutes ces raisons militent pour l'utilisation de méthodes d'éléments finis d'ordre élevé, et des techniques du calcul parallèle pour accélérer les simulations. Les schémas d'évolution temporelles les plus couramment employés pour la propagation des ondes sont explicites, et ne demandes donc pas d'inversion de système linéaires. Une difficulté particulière des méthodes d'éléments finis est que

les éléments finis de Lagrange usuels (utilisés avec succès dans les problèmes statiques) conduisent à des matrices de masse pleines, qu'il faudra donc inverser. Ceci risque de compromettre la viabilité « économique » de ces méthodes. La thèse de N. Tordjman [17] apporte une solution satisfaisante à ce problème en deux dimensions, en introduisant une nouvelle famille d'éléments finis qui se prêtent à la condensation de masse (la matrice devient diagonale), sans perte de précision. Ce travail a été étendu (au moins pour la conception des éléments) par Hannoyer [10] et Mulder[4].

Le but de ce travail est de réunir ces divers éléments finis dans un code de simulation tirant parti des technologies logicielles récentes, comme la bibliothèque d'échange de message MPI pour le calcul parallèle, et le langage `C++`, avec sa riche bibliothèque standard pour les structures de données. Si les « nouveaux » éléments finis ont déjà été validés en deux dimensions, c'est la première fois qu'ils seront mis en oeuvre en dimension 3.

Nous commencerons par rappeler l'origine physique du problème, et sa formulation mathématique. Nous présenterons brièvement les éléments finis utilisés dans l'approximation, puis nous indiquerons les points saillants de la mise en oeuvre informatique. Enfin nous présenterons quelques exemples numériques permettant de se faire une idée des performances du logiciel tant sur les résultats que sur le parallélisme.

2 Analyse mathématique

2.1 Modèles continu, semi-discret, discret

Dans ce travail nous étudions les performances d'une méthode pour résoudre l'équation des ondes en milieu hétérogène. Étant donné un ouvert polygonal Ω de $\mathbf{R}^2$ ou de $\mathbf{R}^3$ (nous notons $\Gamma = \partial\Omega$), et un temps $T > 0$, nous cherchons une fonction $u : \Omega\times]0,T[\to \mathbf{R}$ solution de l'équation

$$\frac{1}{\rho c^2}\frac{\partial^2 u}{\partial t^2}(x,t) - \operatorname{div}\Big(\frac{1}{\rho}\operatorname{grad} u(x,t)\Big) = f_i(x,t), \quad \forall x \in \Omega, \quad \forall t \in]0,T[, \tag{1}$$

avec les conditions aux limites

$$\frac{\partial u}{\partial \nu}(x,t) + \alpha(x)\frac{\partial u}{\partial t}(x,t) + \beta(x)u(x,t) = f_b(x,t), \quad \forall x \in \Gamma, \quad \forall t \in]0,T[\tag{2}$$

et les conditions initiales

$$u(x,0) = u_0(x), \quad \frac{\partial u}{\partial t}(x,0) = u_1(x), \quad \forall x \in \Omega. \tag{3}$$

Le modèle (1)-(3) est l'équation des ondes acoustiques. L'inconnue u représente la pression, la fonction c est la vitesse et ρ est la densité. Dans le cas (courant) où la densité ρ est constante, on retrouve l'équation des ondes usuelle.

La condition aux limites (2) est une condition générale regroupant les conditions de Neumann (avec $\alpha = \beta = 0$) et une condition absorbante du premier ordre (avec $\alpha = 1, \beta = 0$). Bien entendu, les fonctions c, ρ, α et β peuvent être discontinues.

Précisons les hypothèses usuelles sur ces fonctions permettant d'obtenir un résultat d'existence (on trouvera un tel résultat, très classique, dans [8]) :

$$\begin{aligned}
c \in L^\infty(\Omega),\ 0 < c_* \le c(x) \le c^* < \infty, \quad \forall x \in \Omega \\
\rho \in L^\infty(\Omega),\ 0 < \rho_* \le \rho(x) \le \rho^* < \infty \quad \forall x \in \Omega, \\
\alpha \in L^\infty(\Gamma),\ 0 < \alpha_* \le \alpha(x) \le \alpha^* < \infty, \quad \forall x \in \Omega \\
\beta \in L^\infty(\Gamma),\ 0 < \beta_* \le \beta(x) \le \beta^* < \infty \quad \forall x \in \Omega.
\end{aligned}$$

Nous effectuons tout d'abord une semi-discrétisation en espace. Nous considérons une « triangulation » régulière $\mathcal{T}_h$ de l'ouvert Ω (en 3D, $\mathcal{T}_h$ sera formée de tétraèdres) :

$$\overline{\Omega} = \bigcup_{K\in\mathcal{T}_h} K,$$

et nous définissons l'espace d'approximation (de dimension finie)

$$V_h(\Omega) = \left\{ v \in C^0(\bar{\Omega}), v \in H^1(\Omega) \text{ tel que } \forall K \in \mathcal{T}_h,\ v_{|_K} \in P \right\} \tag{4}$$

ou P est un espace de polynômes qui sera précisé plus bas.

Supposons connue une base $(\phi_p)_{1\le p\le N}$ de V_h (avec $N = \dim V_h$). Bien entendu, les fonctions ϕ_p seront les fonctions de bases d'un élément fini, précisé au paragraphe 2.2. En notant $u_h(t)$ la solution approchée, le problème semi-discret s'écrit, sous forme matricielle :

Trouver $u_h(t) : [0,T] \to \mathbf{R}$ tel que :

$$M_{1,h}\frac{d^2u_h}{dt^2} + M_{2,h}\frac{du_h}{dt} + (K_{1,h}+K_{2,h})u_h = F_{1,h}+F_{2,h} \tag{5}$$

$$u_h(0) = u_{0,h}, \quad \frac{du_h}{dt}(0) = u_{1,h} \tag{6}$$

où les matrices et les vecteurs sont définis par

$$\begin{cases} (M_{1,h})_{p,q} = \displaystyle\int_\Omega \frac{1}{\rho c^2}\,\phi_p\phi_q\,dx \\ (M_{2,h})_{p,q} = \displaystyle\int_\Gamma \frac{1}{\rho}\alpha\,\phi_p\phi_q\,d\sigma(x) \\ (K_{1,h})_{p,q} = \displaystyle\int_\Omega \frac{1}{\rho}\,\nabla\phi_p\nabla\phi_q\,dx \\ (K_{2,h})_{p,q} = \displaystyle\int_\Gamma \frac{1}{\rho}\beta\,\phi_p\phi_q\,d\sigma(x) \\ (F_{1,h})(t)_p = \displaystyle\int_\Omega \phi_p\,f_i(.,t)\,dx \\ (F_{2,h})(t)_p = \displaystyle\int_\Gamma \frac{1}{\rho}\,\phi_p\,f_b(.,t)\,d\sigma(x) \end{cases} \tag{7}$$

Avant de détailler la forme que prend l'espace d'éléments finis P, nous donnons le schéma discrétisé en temps. Nous nous contentons du schéma saute-mouton, explicite et du second ordre en temps, dans le cas d'un pas de temps constant Δt, et nous notons $t^n = n\Delta t$. Nous notons U_h^n une approximation de $u_h(t^n)$, et nous approchons les dérivées temporelles par des différences finies centrées :

$$\begin{aligned} \frac{d^2U_h}{dt^2}|_{t_n} &= \frac{U_h^{n+1} - 2U_h^n + U_h^{n-1}}{\Delta t^2} + O(\Delta t^2) \\ \frac{dU_h}{dt}|_{t_n} &= \frac{U_h^{n+1} - U_h^{n-1}}{2\Delta t} + O(\Delta t^2). \end{aligned}$$

En définissant les vecteurs

$$\begin{cases} F_{1,h}^n = F_{1,h}(t^n) \\ F_{2,h}^n = F_{2,h}(t^n) \end{cases} \tag{8}$$

On obtient alors le système suivant :

$$M_{1,h}\frac{U_h^{n+1}-2U_h^n+U_h^{n-1}}{\Delta t^2}+(K_{1,h}+K_{2,h})U_h^n+M_{2,h}\frac{U_h^{n+1}-U_h^{n-1}}{2\Delta t}=F_{1,h}^n+F_{2,h}^n \qquad \forall n\geq 1 \tag{9}$$

avec les conditions initiales

$$\begin{aligned} U_h^0 &= U_{0,h}, \\ U_h^1 &= U_{0,h}+\Delta t U_{1,h} \end{aligned}$$

Nous avons utilisé l'équation continue pour obtenir la deuxième condition initiale afin d'obtenir un schéma globalement d'ordre 2.

Bien que le schéma (9) soit explicite, nous voyons qu'il faut tout de même inverser la matrice de masse à chaque pas de temps. Si cette matrice est quelconque, cette inversion représente un surcoût par rapport à une méthode de différences finies. Pour que l'utilisation des éléments finis reste compétitive, il faut que la matrice de masse devienne *diagonale*. Bien entendu, ceci doit se faire sans perte de précision, sans cela l'utilisation d'éléments d'ordre élevé ne se justifierait plus.

Pour obtenir une matrice de masse diagonale, on remplace l'intégrale

$$I_T(f)=\int_T f(x)\,dx$$

par une formule de quadrature

$$Q_T(f)=\sum_i w_i f(\xi_i)$$

ou les points ξ_i sont les *noeuds* de la formule de quadrature, et les w_i sont les poids. Dans le cas de la matrice de masse, la fonction f devient le produit de deux fonctions de bases. Si les noeuds de la formule de quadrature coïncident avec les degrés de liberté de l'élément fini, la matrice de masse devient automatiquement diagonale. Il reste à assurer plusieurs conditions, explicitées dans [7] :

- Les noeuds de la formule de quadrature doivent être unisolvants pour un certain espace de polynômes (à déterminer).
- L'élément fini ainsi obtenu doit être continu (autrement dit conforme H^1).
- La formule de quadrature doit être suffisamment précise pour ne pas dégrader l'ordre de convergence de l'élément fini.
- Les poids de la formule de quadrature doivent être positifs, pour assurer la stabilité du schéma d'évolution.

Le premier point est précisé par un résultat de Ciarlet [5] qui indique le degré des polynômes que la formule de quadrature doit intégrer exactement pour conserver l'ordre de convergence. Pour diminuer le nombre de paramètres, on ne considère que des formules de quadratures symétriques, c'est-à-dire dont que l'ensemble des noeuds est invariant par le groupe de symétrie du tringle ou du tétraèdre.

On se convainc assez aisément que les éléments finis de Lagrange usuels ne peuvent pas convenir. la position des noeuds étant fixée, la détermination d'une formule de quadrature revient au calcul des poids, ce qui constitue un système linéaire. Malheureusement, on constate dans chaque cas particulier (au moins pour les premiers éléments, considérés ci-dessous) que certains poids sont négatifs ou nuls, ce qui conduit à un schéma temporel instable.

La solution adoptée est de modifier l'élément fini (et l'espace de polynômes associés), en ajoutant des degrés de libertés. Le détail de cette construction est présenté, en 2D dans [7, 17], et dans [4, 10] en 3D. Nous nous contenterons ici de résumer ces travaux en décrivant les éléments obtenus. Les deux premiers critères peuvent être satisfaits pour des positions « génériques » des degrés de liberté, et une liste de candidats obtenue par une recherche combinatoire [4]. Le critère de Ciarlet conduit alors à un système d'équations non-linéaires, qui ne peut être résolu que numériquement. Il faut alors vérifier *a posteriori* que les poids sont bien positifs, et les noeuds à l'intérieur de l'élément.

Nous pouvons améliorer le calcul des noeuds et des poids en remarquant que le système à résoudre est polynômial, et en utilisant des techniques du calcul symbolique. Nous avons ainsi pu démontrer que les solutions présentées aux paragraphes suivants sont les plus simples possibles.

2.2 Description des nouveaux éléments finis

2.2.1 Élément d'ordre 4 en 2D

Cet élément a été obtenu par N. Tordjman dans sa thèse [17]. Il se déduit de l'élément P_2 usuel en ajoutant le centre de masse aux degrés de libertés, et une fonction « bulle » (polynôme du $3^{\text{ème}}$ degré nul sur les arêtes) à l'espace des polynômes. Il s'agit d'une somme directe, et il y a 7 degrés de libertés. L'élément est représenté sur la figure 1.

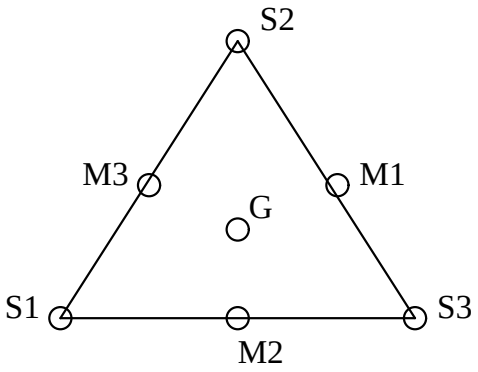

FIG. 1: Élément d'ordre 4 en 2D

Les propriétés de cet élément, noté $\tilde{P}_2$ sont résumées dans la propriété suivante :

Proposition 1. *Les poids de la formule de quadrature sont donnés par :*

$$\omega_S = 1/20, \quad \omega_M = 2/15, \quad \omega_G = 2/9. \tag{10}$$

Les fonctions suivantes forment une base de l'élément $\tilde{P}_2$:

1. *La fonction de base associée au sommet S_i est*

$$E_{S_i} = \lambda_i(2\lambda_i - 1) - 3b; \tag{11}$$

2. *La fonction de base associée au milieu M_{ij} des sommets S_i et S_j est*

$$E_{M_{ij}} = \lambda_k\lambda_l - 3b, \text{ avec } k \neq i,\ l \neq i; \tag{12}$$

3. *La fonction de base associée au centre de masse G de l'élément est*

$$E_G = 27b. \tag{13}$$

2.2.2 Élément d'ordre 6 en 2D

Cet élément a également été obtenu dans la thèse de N. Tordjman [17]. Par rapport à l'élément P_3 usuel, il faut cette fois « éclater » le centre de gravité en 3 points distincts, et déplacer les points sur les arêtes. Il y a donc 12 degrés de liberté. L'espace des polynômes s'écrit cette fois comme $\tilde{P}_3 = P_3 + bP_1$, mais la somme n'est pas directe. L'élément est représenté sur la figure 2.

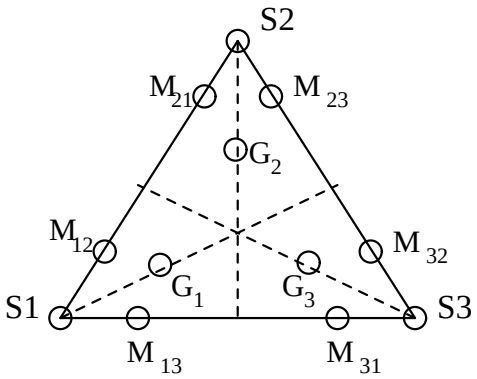

FIG. 2: Élément d'ordre 6 en 2D

Les propriétés de cet élément, noté $\tilde{P}_3$ sont résumées dans la propriété suivante :

Proposition 2. *Les paramètres géométriques de l'élément sont donnés par :*

$$\alpha = 1/2 + \frac{1}{63}\sqrt{735+336\sqrt{7}} - \frac{1}{126}\sqrt{735+336\sqrt{7}}\sqrt{7} \tag{14}$$

et

$$\beta = 1/3 + 1/21\sqrt{(7)} \tag{15}$$

Les poids de la formule de quadrature sont

$$\begin{aligned} \omega_S &= -\frac{1}{360}\sqrt{7} + 1/45 \\ \omega_M &= \frac{1}{90}\sqrt{7} + \frac{7}{360} \\ \omega_G &= \frac{49}{180} - \frac{7}{360}\sqrt{7}, \end{aligned} \tag{16}$$

Les fonctions suivantes forment une base de l'élément $\tilde{P}_2$:

1. *La fonction de base associée au sommet S_i est*

$$E_{S_i} = p_i - \frac{8b}{\beta(1-\beta)^2(3\beta-1)}\left[A_i\left(\lambda_i - \frac{1-\beta}{2}\right) + B_i\sum_{l\neq i}\left(\lambda_l - \frac{1-\beta}{2}\right)\right] \tag{17}$$

2. *La fonction de base associée au milieu M_{ij} des sommets S_i et S_j est*

$$E_{M_{ij}} = p_{ij} - \frac{8b}{\beta(1-\beta)^2(3\beta-1)}\left[A_{ij}\left(\lambda_i - \frac{1-\beta}{2}\right) + B_{ij}\left(\lambda_j - \frac{1-\beta}{2}\right) + C_{ij}\left(\lambda_k - \frac{1-\beta}{2}\right)\right] \tag{18}$$

où nous avons noté

$$p_{ij} = \frac{\lambda_i\lambda_j}{\alpha(1-\alpha)(2\alpha-1)}\left(\alpha\lambda_i - (10\alpha)\lambda_j + (1-2\alpha)\lambda_k\right)$$
$$A_{ij} = p_{ij}(G_i), \quad B_{ij} = p_{ij}(G_h), \quad C_{ij} = p_{ij}(G_k).$$

2.2.3 Élément d'ordre 4 en 3D

Cet élément a été obtenu simultanément par Hannoyer [10] et Mulder [16]. Sa construction est plus délicate qu'en 2D, puisqu'il faut rajouter 13 degrés de libertés à l'élément usuel, pour un total de 23 ! L'espace des polynômes s'écrit $\tilde{P}_2 = P_2 + BP_0 + \sum_{i=1}^{4} b_i P_1$, ou B est une fonction « bulle » (produit des quatre coordonnées barycentriques, nulle sur les quatre faces du tétraèdre), et b_i est la fonction « bulle sur la face i » (produit des 3 coordonnées barycentriques de la face, nulle sur les 3 arêtes de cette face). L'élément est représenté sur la figure 3.

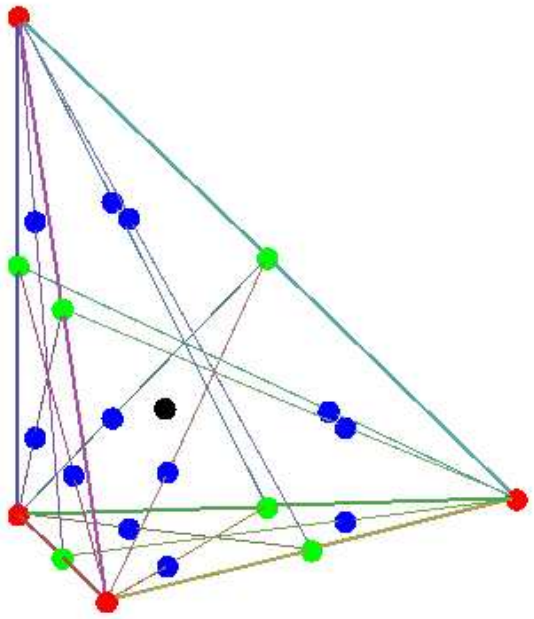

FIG. 3: Élément d'ordre 4 en 3D

Le seul paramètre géometrique libre de cet élément est la position des 3 points sur les faces du tétraèdres (noté β), et il y a 4 poids de quadrature. Ces nombres ont été calculés analytiquement par F. Hannoyer, qui a obtenu :

$$\begin{cases} \beta &= \dfrac{2}{9} + \dfrac{1}{9}\sqrt{13} \\ w_T &= \dfrac{29+17\sqrt{13}}{1680} \\ w_S &= \dfrac{13-3\sqrt{13}}{1680} \\ w_M &= \dfrac{8-2\sqrt{13}}{105} \\ w_G &= \dfrac{32}{105} \end{cases}$$

Les fonctions de base ont été calculées par Ch. Lemuet [13], et nous reproduisons ici ses résultats. Notons :

$$B = \lambda_1\lambda_2\lambda_3\lambda_4, \qquad b_i = \prod_{\substack{k=1,\\ k\neq i}}^{4} \lambda_k, \text{ pour } i = 1,\dots,4$$

Proposition 3. *Les fonctions suivantes forment une base de* $\widetilde{P_2}$ *:*

1. *La Fonction de base associée au sommet* S_i *est*

$$E_{S_i} = P_i - \sum_{l\neq i}^{4}\sum_{t\neq l}^{4} \frac{2P_i(G_{lt})}{(3\beta-1)b_l(G_{lt})} b_l(\lambda_t - \frac{1-\beta}{2}) + C_iB \tag{19}$$

$$\text{avec} \quad \left\{ \begin{array}{rcl} P_i & = & \lambda_i(2\lambda_i - 1) \\ C_i & = & \dfrac{24(2\beta-1)(3\beta-2)}{(1-\beta)^2\beta(3\beta-1)} + 32. \end{array} \right.$$

2. *La Fonction de base associée au point* G_{ij} *de coordonnées barycentriques* $\lambda_i = 0, \lambda_j = \beta$ *et* $\lambda_k = (1-\beta)/2$ *pour* $k \neq i, j$ *est*

$$E_{G_{ij}} = \frac{8}{\beta(1-\beta)^2(3\beta-1)} b_i(\lambda_j - \frac{1-\beta}{2}) + \frac{8(1-2\beta)}{\beta(1-\beta)^2(3\beta-1)} B \tag{20}$$

3. *La Fonction de base associée au point* M_{ij} *situé au milieu de l'arête joignant* S_i *et* S_j *est*

$$E_{m_{ij}} = P_{ij} - \sum_{l\neq i, l\neq j}^{4}\sum_{k\neq l}^{4} \frac{2P_{ij}(G_{lk})}{(3\beta-1)b_l(G_{lk})} b_l(\lambda_k - \frac{1-\beta}{2}) + C_{ij}B \tag{21}$$

$$\text{avec} \quad \left\{ \begin{array}{rcl} P_{ij} & = & 4\lambda_i\lambda_j \\ C_{ij} & = & \dfrac{16(2\beta-1)[\beta(2-3\beta)+1]}{(1-\beta)^2\beta(3\beta-1)} - 64. \end{array} \right.$$

4. *La Fonction de base associée au centre de masse G est*

$$E_G = 256B. \tag{22}$$

Nous donnerons cette fois la démonstration, due à Ch. Lemuet [13].

Preuve. 1. **Calcul de la fonction de base associée à G**

Nous recherchons la fonction de base E_G qui s'annule en tous les degrés de liberté du tétraèdre sauf en G où elle vaut 1. Puisque tous les degrés de liberté hormis G appartiennent à au moins une face du tétraèdre, ces points possèdent au moins une coordonnée barycentrique nulle. Ainsi, en considérant la fonction $E_G = 256\lambda_1\lambda_2\lambda_3\lambda_4$, nous pouvons aisément vérifi er qu'elle prend bien la valeur 1 en G et qu'elle est nulle ailleurs. La fonction E_G ainsi défi nie est bien la fonction de base associée au centre du tétraèdre.

2. **Calcul des fonctions de base associées aux** G_{ij}

Pour plus de clarté dans l'exposé, nous allons rechercher la fonction de base $E_{G_{12}}$ associée au point $G_{12} = (0, \beta, \frac{1-\beta}{2}, \frac{1-\beta}{2})$.

Tout d'abord cherchons à annuler $E_{G_{12}}$ en tous les points S_i et m_{ij} ; il suffi t de mettre le polynôme $\lambda_2\lambda_3\lambda_4$ en facteur dans la défi nition de $E_{G_{12}}$, ce qui donne :

$$E_{G_{12}} = C\lambda_2\lambda_3\lambda_4 \times q(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$$

où q est un polynôme de degré 1 et C une constante. Il apparaît que $E_{G_{12}}$ écrit sous cette forme s'annule aussi en tous les points G_{ij} n'appartenant pas à la face contenant G_{12}.
Annulons ensuite la fonction de base aux points G_{13} et G_{14}, il suffi t pour cela de prendre $q(\lambda_1,\lambda_2,\lambda_3,\lambda_4) = (\lambda_2 - \frac{1-\beta}{2})$. La constante C est quant à elle défi nie de telle sorte que $E_{G_{12}}(G_{12}) = 1$ c'est à dire $C = \frac{8}{\beta(1-\beta)^2(3\beta-1)}$
Enfi n, il ne reste plus qu'à annuler cette fonction de base au centre G du tétraèdre, en lui rajoutant le polynôme $K\lambda_1\lambda_2\lambda_3\lambda_4$, lequel ne remet pas en cause la nullité de $E_{G_{12}}$ en tous les autres degrés de liberté (en effet, chacun des nœud possède au moins une coordonnée barycentrique nulle). La constante K est défi nie telle que $E_{G_{12}}(G) = 0$, ce qui donne

$$K = \frac{8(1-2\beta)}{\beta(1-\beta)^2(3\beta-1)}$$

Ainsi la fonction de base associée au point G_{12} est défi nie par :

$$E_{G_{12}} = \frac{8}{\beta(1-\beta)^2}\lambda_2\lambda_3\lambda_4(\lambda_2 - \frac{1-\beta}{2}) + \frac{8(1-2\beta)}{\beta(1-\beta)^2(3\beta-1)}B \tag{23}$$

Puisque les points G_{ij} appartiennent à la même classe d'équivalence, nous pouvons déterminer la forme générale des fonctions de base $E_{G_{ij}}$ par permutation des indices. Les fonctions de base associées aux points G_{ij} sont :

$$E_{G_{ij}} = \frac{8}{\beta(1-\beta)^2(3\beta-1)}b_i(\lambda_j - \frac{1-\beta}{2}) + \frac{8(1-2\beta)}{\beta(1-\beta)^2(3\beta-1)}B \tag{24}$$

3. **Calcul des fonctions de base associées aux** m_{ij}

Nous suivons globalement la même démarche que celle utilisée précédemment pour le calcul des fonctions de bases $E_{G_{ij}}$, c'est à dire que nous commençons par rechercher la fonction de base $E_{m_{12}}$ associée au point m_{12} de coordonnées barycentriques $(\frac{1}{2},\frac{1}{2},0,0)$. Étant donné la défi nition de l'espace$\tilde{P}_2$, on peut trouver $E_{m_{12}}$ sous la forme suivante :

$$E_{m_{12}} = P_2 + P_1 * [b] + P_0B = P_2 + \sum_{i=1}^{4} P_1^i b_i + P_0B.$$

Tout d'abord, $E_{m_{12}}$ doit s'annuler en tous les sommets S_i et tous les milieux m_{ij} sauf en m_{12}, nous sommes alors en mesure de trouver le polynôme P_2, car en tous ces points les polynômes b_1, b_2, b_3, b_4 et B s'annulent. Soit $P_2 = C\lambda_1\lambda_2$, $E_{m_{ij}} = 0 \quad \forall (i,j) \neq (1,2)$ et $E_{S_i} = 0 \quad \forall i$.
Nous souhaitons de plus que $E_{m_{12}}(m_{12}) = 1$, ce qui nous donne pour C la valeur 4. Ainsi

$$E_{m_{12}} = P_{12} + \sum_{i=1}^{4} P_1^i b_i + P_0B \text{ où } P_{12} = 4\lambda_1\lambda_2.$$

Ensuite, nous voulons que $E_{m_{12}}$ s'annule en tous les G_{ij} dont les coordonnées sont $\lambda_i = 0$, $\lambda_j = \beta$ et $\lambda_k = \frac{1-\beta}{2}, \quad k \neq i,j$.
Il suffi t de trouver les P_1^i car en G_{ij} le dernier terme P_0B s'annule. Nous pouvons déjà constater que $E_{m_{12}}$ s'annule en tous les G_{1i} et G_{2i}, ce qui permet de prendre $P_1^1 = P_1^2 = 0$ Il reste donc à trouver P_1^3 et P_1^4
En écrivant P_1^3 sous la forme $P_1^3 = P_{1,1}^3 + P_{1,2}^3 + P_{1,4}^3$, les équations suivantes doivent être satisfaites :

$$\begin{cases} E(G_{31}) = P_{12}(G_{31}) + P_{1,1}^3(G_{31})b_3(G_{31}) = 0 \\ E(G_{32}) = P_{12}(G_{32}) + P_{1,2}^3(G_{32})b_3(G_{32}) = 0 \\ E(G_{34}) = P_{12}(G_{34}) + P_{1,4}^3(G_{34})b_3(G_{34}) = 0 \end{cases} \tag{25}$$

et

$$\left\{ \begin{array}{l} P^3_{1,1}(G_{32}) = P^3_{1,1}(G_{34}) = 0 \\ P^3_{1,2}(G_{31}) = P^3_{1,1}(G_{34}) = 0 \\ P^3_{1,4}(G_{34}) = P^3_{1,1}(G_{32}) = 0 \end{array} \right. . \tag{26}$$

Les relations ci dessus (25) et (26) sont vérifi ées avec les polynômes suivants :

$$\begin{aligned} P^3_{1,1} &= \frac{2P_{12}(G_{31})}{(3\beta-1)b_3(G_{31})} \times b_3(\lambda_1 - \frac{1-\beta}{2}) \\ P^3_{1,2} &= \frac{2P_{12}(G_{32})}{(3\beta-1)b_3(G_{32})} \times b_3(\lambda_3 - \frac{1-\beta}{2}) \\ P^3_{1,4} &= \frac{2P_{12}(G_{34})}{(3\beta-1)b_3(G_{34})} \times b_3(\lambda_4 - \frac{1-\beta}{2}). \end{aligned}$$

Nous avons ainsi déterminé le polynôme P^3_1. La recherche du polynôme P^4_1 se fait de façon similaire. Nous pouvons maintenant écrire la fonction de base $E_{m_{12}}$ sous la forme :

$$E_{m_{12}} = 4\lambda_1\lambda_2 - \sum_{l\neq 1, l\neq 2} \sum_{t\neq l} \frac{2P_{12}(G_{lt})}{(3\beta-1)b_l(G_{lt})} b_l(\lambda_t - \frac{1-\beta}{2}) + P_0 B.$$

La fonction ci-dessus s'annule en tous les S_i , m_{ij} pour $(i,j) \neq (1,2)$ et tous les G_{ij} et prend la valeur 1 en m_{12}. Il ne reste plus qu'à trouver la constante $C = P_0$ telle que $E_{m_{12}}$ s'annule en G. En résolvant l'équation $E_{m_{12}}(G) = 0$, il vient

$$C_{12} = \sum_{l\neq 1, l\neq 2} \sum_{t\neq l} \frac{2(2\beta-1)P_{12}(G_{lt})}{(3\beta-1)b_l(G_{lt})} - 64.$$

Bien que la forme de cette constante C soit un peu compliquée, il est possible de la simplifi er grâce à la forme particulière des coordonnées des points G_{lt}. Par défi nition de G_{12} on a

$$C_{12} = \frac{2(\beta-1)}{(3\beta-1)} \left(\frac{P_{12}(G_{31})}{b_3(G_{31})} + \frac{P_{12}(G_{32})}{b_3(G_{32})} + \frac{P_{12}(G_{34})}{b_3(G_{34})} + \frac{P_{12}(G_{41})}{b_4(G_{41})} + \frac{P_{12}(G_{42})}{b_4(G_{42})} + \frac{P_{12}(G_{43})}{b_4(G_{43})} \right) - 64$$

mais

$$\left\{ \begin{array}{l} P_{12}(G_{31}) = P_{12}(G_{41}) = P_{12}(G_{32}) = P_{12}(G_{42}) = 4\beta\dfrac{(1-\beta)}{2} = 2\beta(1-\beta) \\ P_{12}(G_{32}) = P_{12}(G_{43}) = 4\dfrac{(1-\beta)^2}{4} = (1-\beta)^2. \end{array} \right.$$

De ceci nous pouvons déduire que C_{12} est la constante C valant

$$C = \frac{16(2\beta-1)[\beta(2-3\beta)+1]}{(1-\beta)^2\beta(3\beta-1)} - 64.$$

Soit la fonction de base $E_{m_{12}}$:

$$E_{m_{12}} = P_{12} - \sum_{l\neq 1, l\neq 2} \sum_{t\neq l} \frac{2P_{12}(G_{lt})}{(3\beta-1)b_l(G_{lt})} b_l(\lambda_t - \frac{1-\beta}{2}) + C_{12} B.$$

Du calcul de la fonction de base $E_{m_{12}}$, nous pouvons en déduire par permutation des indices la fonction de base associée au point

$$m_{ij} = \left\{ \begin{array}{c} \dfrac{1}{2} \text{ en sa } i\text{-ième et } j\text{-ième coordonnée barycentrique,} \\ 0 \text{ ailleurs} \end{array} \right.$$

et vérifi er qu'elle est bien donnée par la formule (21).

4. **Calcul des fonctions de base associées aux S_i**
   Il est possible de retrouver ces fonctions de base en utilisant et en adaptant la méthode décrite précédemment pour le calcul des $E_{m_{ij}}$, ce qui mène alors la formule (19) de la proposition.

□

2.3 Parallélisation du produit matrice – vecteur

Nous décrivons dans cette section comment nous utilisons la décomposition de domaine pour paralléliser le produit de la matrice de rigidité par un vecteur. Pour simplifier, nous exposerons le principe dans le cas de deux sous-domaines, représentés sur la figure 4.

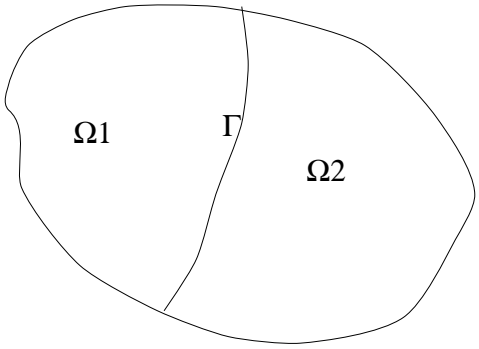

FIG. 4: Deux sous-domaines

Nous ordonnons (fictivement) les degrés de libertés en numérotant d'abord ceux intérieurs au sous-domaine 1, puis ceux intérieurs au sous-domaine 2, et enfin ceux sur l'interface. La matrice de rigidité K prend la forme suivante :

$$K = \begin{pmatrix} K_{11} & 0 & K_{13} \\ 0 & 0 & \\ K_{31} & 0 & K_{33}^1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & K_{22} & K_{23} \\ 0 & K_{32} & K_{33}^2 \end{pmatrix} \tag{27}$$

où les blocs K_{12} et K_{21} sont nuls puisque les noeuds correspondants sont dans des éléments disjoints. La décomposition du bloc K_{33} correspond à un sous-assemblage. En considérant une décomposition du vecteur U correspondant à celle de K, le produit matrice vecteur s'écrit :

$$KU = \begin{pmatrix} K_{11}U_1 + K_{13}U_3 \\ 0 \\ K_{31}U_1 + K_{33}^1 U_3 \end{pmatrix} + \begin{pmatrix} 0 \\ K_{22}U_2 + K_{23}U_3 \\ K_{32}U_2 + K_{33}^2 U_3 \end{pmatrix} \tag{28}$$

Le premier terme de la somme ne fait intervenir que des degrés de libertés du sous-domaine 1, le second que des degrés de liberté du sous-domaine 2.

Supposons maintenant que ce calcul soit réalisé sur un ordinateur à mémoire distribuée, chaque processeur ne connaissant les informations relatives à un sous-domaine. On voit que chaque processeur commence par réaliser un produit « matrice locale » par « vecteur local ». Pour que le calcul de la $3^{\text{ème}}$ composante soit correct, chaque processeur a besoin d'informations qui résident sur l'autre processeur. Il faut donc procéder à une *communication*, dans laquelle un processeur envoie à, et reçoit de l'autre les données nécessaires. Le processeur 1 envoie $K_{31}U_1 + K_{33}^1 U_3$ (c'est-à-dire les valeurs du produit correspondant aux degrés de libertés situés sur l'interface) au processeur 2, reçoit $K_{32}U_2 + K_{33}^2 U_3$ de celui-ci, et vice-versa.

L'intérêt de cette méthode est qu'il n'est en fait pas nécessaire d'assembler la matrice de rigidité globale, ni même de construire le vecteur global U. Il suffit de travailler avec des quantités *locales*, et de procéder ensuite à l'échange de données indiqué ci-dessus. Ainsi, à la phase de communications près, le code qui s'exécute sur un processeur est identique à celui qui s'exécuterait sur un ordinateur séquentiel, et le résultat obtenu est le même.

La situation est ici plus simple que pour les problèmes elliptiques pour lesquels l'équivalence ne s'obtient qu'à la convergence.

3 Mise en oeuvre informatique

Nous présentons dans cette section les principes directeurs de la mise en oeuvre d'un logiciel développé pour résoudre l'équation des ondes en utilisant les éléments finis présentés ci-dessus. On peut s'attendre à devoir traiter des problèmes de taille importante, particulièrement en 3D. En effet, les problèmes de propagation d'onde demandent une résolution minimale (la règle heuristique à l'ordre 2 est d'avoir 10 points par longueur d'onde, peut-être 5 à l'ordre 4), et si l'on veut pouvoir simuler des phénomènes sur des temps longs, on devra pouvoir prendre en compte plus de 10 longueurs d'onde dans chaque dimension du domaine. Si l'on prend, pour fixer les idées, 80 points dans chaque direction, on aboutit à 500000 degrés de libertés. Une façon naturelle d'aborder des problèmes de si grande taille est de se tourner vers le parallélisme. Cela sera facilité par le fait que nous avons un schéma d'évolution explicite en temps, ce qui conduira à une parallélisation plus simple, sans aucun système linéaire à résoudre.

Nous avons choisi une méthode de décomposition de domaine sans recouvrement, où les éléments sont répartis entre les sous-domaines. Chaque processeur traite un sous-domaine, et communique avec ses voisins à chaque pas de temps pour échanger les valeurs de degrés de libertés sur l'interface entre les sous-domaines. Si les sous-domaines sont suffisamment « gros » (si le rapport entre nombre de noeuds intérieurs et noeuds sur l'interface est assez grand), le rapport coût du calcul sur coût des communications sera grand, et la méthode assez efficace.

Cette méthode se programme de façon naturelle par échange de message, les communications sont fixes dans le temps, mais irrégulières en espace. Nous avons choisi la bibliothèque MPI [15] pour sa portabilité, sa simplicité d'utilisation et la bonne efficacité de ses implantations.

Par rapport à un code séquentiel, notre logiciel a besoin d'une entrée supplémentaire : une partition du maillage, décrivant à quel sous-domaine appartient chaque élément. Nous avons choisi d'utiliser le logiciel `Metis` [11, 12], mais de façon plus générale, nous ferons référence au « partitionneur » pour désigner tout logiciel qui réalise cette étape.

Une innovation du logiciel est de fonder les communications sur un ensemble de degrés de libertés appartenant aux mêmes processeurs, que nous appelons un *bloc*. Les blocs sont construits, après l'ajout des noeuds non-sommets, dans une phase d'initialisation. Ensuite, la communication à chaque pas de temps utilise ces blocs. Chaque noeud doit recevoir comme valeur la somme de toutes les valeurs de ses « homologues » dans les autres sous-domaines. Chaque communication correspond donc à plusieurs réductions généralisées. La difficulté est que chaque noeud peut participer à plusieurs opérations de réduction (il appartient à plusieurs blocs), et l'opération de réduction fournie par MPI est explicitement bloquante. Il a semblé intéressant de programmer une opération de réduction adaptée à notre problème, et non bloquante, ce qui permet de lancer plusieurs communications simultanément. La construction, et l'utilisation, des blocs sera décrite aux paragraphes 3.1 à 3.5.

La principale difficulté rencontrée dans la mise en oeuvre du code a été le traitement des noeuds nécessaires aux éléments d'ordre élevé. En effet, aussi bien les mailleurs, que les logiciels de partitionnement de maillage ne prennent en compte que les sommets *géométriques* du maillage. Or, les éléments finis décrits au paragraphe 2.1 nécessitent la présence de noeuds non-sommets, qu'il faudra donc ajouter avant de commencer le calcul proprement dit. Ceci est compliqué par le parallélisme : il est bien évident que les noeuds qui seront ajoutés sur une interface commune à deux sous-domaines

doivent l'être de façon cohérentes sur chacun de ces sous-domaines. Il importe, en particulier, qu'ils soient repérés comme « le même noeud » lors de la phase de communication. Nous décrirons comment atteindre ce but au paragraphe 3.5.

Un but plus mineur de ce code est de permettre de changer facilement d'élément fini. Pour assurer l'indépendance du code vis-à-vis de l'élément fini utilisé lors du calcul, nous avons choisi de mettre la description de l'élément fini dans un fichier, et le type souhaité est passé en argument (ceci impose l'utilisation d'un seul type d'élément pour tout le domaine).

Enfin, signalons que le logiciel est écrit en `C++`, et utilise massivement la bibliothèque standard, ce qui facilite grandement la manipulation de structures de données complexes.

3.1 Synchronisation : du modèle centralisé au distribué

Tout d'abord, notons que cette partie ne concerne que la construction des objets pour la phase de synchronisation, qui peut être plus ou moins centralisé mais que son application au niveau du solveur sera entièrement distribuée, détaillé au paragraphe 3.3.

Un modèle centralisé pour la distribution

Rappelons que l'objectif principal évoqué ici est la minimisation du nombre de communication par le regroupement de tous les noeuds se répartissant sur un même ensemble de processeurs. Nous appellerons ces regroupement des *blocs* (de communications). Le but de ces blocs est de minimiser le nombre de messages échangés, et donc l'influence de la latence. Cette phase de communication critique est celle de la synchronisation à chaque pas de temps de la solution obtenue sur les différents processeurs.

Il faut donc, pour construire les blocs, connaître pour un point donné, l'ensemble des domaines utilisant ce point. Pour cela plusieurs approche sont possibles

- Construire cette information sur un seul processeur, en analysant ainsi toute l'information, puis diffuser le résultat ;
- Construire cette information localement sur chaque processeur puis « recoller » le tout ;
- Construire cette information collectivement.

Une étude préliminaire offrait plusieurs choix au niveau de l'organisation des éléments.

1. L'organisation des éléments en y incorporant la connectivité des domaines. Il faut alors calculer cette connectivité des éléments à partir de la table des parties des éléments. Ce choix induisait alors un sur-coût (semblait-il à l'époque) non nécessaire et surtout restait moins ouvert car ne permettait pas de traiter les domaines « non-manifolds » [9] (le problème est qu'en 2D on peut alors rencontrer des domaines où des éléments ont plusieurs voisins pour une même arête !)
2. Une organisation simplement basée sur le hachage des parties (pour éviter des redondances) était aussi envisageable. Il offrait par ailleurs la possibilité de traiter le cas « non-manifold ».

C'est ce dernier choix que nous avons retenu, pour des raisons de sur-coût calculatoire et de généralité.

Pour donner simplement une idée de l'information à construire, donnons un algorithme qui pourrait être utilisé dans un modèle totalement centralisé.

1. Lecture du maillage, avec pour chaque sommet de chaque élément, association du sommet avec tous les domaines l'utilisant. Cette information provient du partitionneur et est associée à l'élément : chaque élément *sait* à quel domaine il appartient, ainsi un simple parcours des éléments suffit.

2. Inversion (par hachage) de la liste précédente ; le hachage permet d'obtenir simplement pour un ensemble de domaines tous les sommets correspondants.
3. Numérotation et communication de cette information aux différents processeurs avec en association la liste des points spécifiques.

A cela, on peut ajouter que le volume d'information à traiter est de l'ordre de la taille de l'ensemble des frontières et interfaces. Ceci constitue une première limite au modèle totalement centralisé sur un seul processeur pour des maillages important. Toutefois, notons que le calcul que l'on souhaite réaliser ne tient compte que des sommets du maillages et non des noeuds. En effet, l'insertion des noeuds se fera une fois cette phase de construction des blocs effectuée, par une structure totalement distribuée. Ainsi, nous devons impérativement utiliser un moyen qui minimise non pas le calcul en soi mais plutôt le volume de données maximum à la charge de chaque processeur, ce qui conduit à envisager un algorithme centralisé en gestion et distribué en stockage.

A présent, nous supposerons donc que l'objectif est de maximiser l'indépendance et la réduction des stockages locaux, sans pour autant éviter une phase centralisée à faible charge.

3.2 Synchronisation : vers un modèle distribué.

Le problème principal n'est pas vraiment qu'un maître fasse tout, en terme de calculs, mais qu'il ait à tout stocker seul. Une alternative simple est de distribuer l'information sur les différentes unités, maximisant ainsi la capacité de stockage.

Ainsi, une fois supposé que les éléments sont distribués sur les différents processeurs (on peut même être plus simple en signifiant que chaque processeur ne stocke que ses éléments), il faut recomposer l'information qui nous intéresse : la constitution des blocs.

3.2.1 Version bidirectionnelle

Cette appellation sera expliquée par la suite.

Chaque processeur peut déjà tenter de construire cette information comme s'il disposait de toute l'information (comme le faisait le maître dans le modèle 3.1 page précédente). Dès lors chaque processeur d'identifiant k dispose d'un ensemble d'information de type $(cpu_i)_i \rightarrow (p_j)_j$ représentant pour chaque ensemble de domaines (de processeurs) la liste des points distribués sur cet ensemble.

Le problème principal est la synthèse de cette information répartie, car alors le stockage ne devrait plus poser de problème ; ce sont alors des opérations de mise en conformité que nous avons à faire (portant principalement sur la discrimination de l'ensemble de processeurs : fusion ou partition).

Nous utilisons l'algorithme suivant :

Soit **2** processeurs souhaitant adapter la structuration en blocs en analysant conjointement **2** informations du type $L_1 \rightarrow A$ et $L_2 \rightarrow B$, où L_i représente un ensemble de processeurs et A, B les listes de points associées. De plus L' représente le résultat d'un nouvel ensemble de processeurs construit.

Il y aura

– **fusion** si $L_1 = L_2$

$$(L = L_1 = L_2) \rightarrow (A \bigcup B)$$

– **découpage** si $A \subset B$

$$\begin{cases} (L' = L_1 \bigcup L_2) \rightarrow A \\ L_2 \rightarrow (B \setminus A) \end{cases}$$

(et le symétrique pour $B \subset A$).

– et plus généralement si $A \bigcap B \neq \oslash$:

$$\begin{cases} (L' = L_1 \bigcup L_2) \rightarrow (A \bigcap B) \\ L_1 \rightarrow (A \setminus B) \\ L_2 \rightarrow (B \setminus A) \end{cases}$$

Ainsi, pour plus de 2 processus, ceux-ci doivent effectuer 2 à 2 ce même schéma. Remarquons que le nombre de comparaisons peut être élevé mais que les ensembles manipulés sont très petits (de l'ordre du nombre de processeurs). En effet, en supposant que chaque processeur a été initialisé avec ses propres informations (éléments appartenant au domaine qu'il gère), seuls des mises à jour seront effectuées (communications de points), le reste n'étant que des décompositions locales de groupe de processeurs.

De plus, afin de minimiser les calculs d'inclusions et autres opérations ensemblistes, j'envisagerais de conserver les ensembles sont une forme ordonnée, plus agréable et rapide à manipuler.

Le point important ici est que chaque couple de processeurs analysant ensemble un potentiel groupe de communications a besoin de faire transiter l'information dans les 2 sens (d'où le nom de *bidirectionnelle*). Ces transferts de données sont de type Point-à-Point dans une terminologie MPI. Et globalement, il faudra N^2 communications de ce type (à une constante $1/2$ près si on est économe). Il est clair que ce nombre de communications est minimal au sens « point-à-point », mais des communications plus globales (de type *broadcast*) pourraient être plus intéressantes.

C'est à cause de la complexité du procédé et du coût de communications Point-à-Point que cette technique n'a pas été plus développée.

3.2.2 Version globalisante.

Cette autre approche ne construit pas de manière sophistiquée les blocs, mais les déduit d'une analyse particulière de toute l'information.

Le premier point important est qu'ici chaque analyse sera faite par le processeur concerné par ces blocs. Ce processeur doit tout d'abord connaître ses sommets. Ceci est simple après (voir même au cours de) la réception des éléments définis par leurs sommets dans une numérotation globale.

Ce qu'il faut pour obtenir la définition des blocs c'est de savoir quel sommet est en interaction avec quel autre sous-domaine. Pour cela, il suffit de « regarder » les sommets des autres éléments et, s'il existe des sommets locaux utilisés par un autre sous-domaine, alors ce sous-domaine extérieur doit entrer en jeu pour les blocs de communications des sommets communs. Dès que tous les autres éléments (des autres domaines) auront été parcourus, les blocs seront localement construits (hormis une numérotation globale cohérente).

Cette méthode ne nécessite alors que N communications globales (mono-directionelle), car tous les processeurs peuvent analyser les données d'un même autre processeur simultanément sans aucun risque de conflit de données. Ceci revient toujours à N^2 communications potentielles point-à-point, mais l'implémentation globale est généralement plus efficace dans le cadre mono-directionnel.

3.2.3 Synthèse et remarques générales

L'implémentation de cette dernière méthode est particulièrement simple, car il suffit *d'examiner* les données des autres processeurs pour en déduire ses propres blocs.

Cependant, la première méthode doit pouvoir s'adapter à un cadre de communications globales, qui permet les mêmes performances au niveau du nombre de communications mais dont l'implémentation est beaucoup plus technique pour des gains encore indéterminés.

Afin d'accélérer les 2 méthodes, il faut noter que seuls les éléments interfaces (et/ou frontière) sont utilisés lors des analyses. Il est donc capital de sélectionner ces sommets avant le début de l'analyse afin de minimiser alors le volume des données transférées (qui serait autrement de l'ordre du maillage complet). Pour cela, il suffit de parcourir les faces des éléments et de marquer celles étant utilisées 2 fois ; celles non marqués sont alors tout simplement externe au domaine. Cette phase est ici totalement parallèle car les domaines sont encore vus de manière indépendante.

D'autre part ceci est très utile lors de la gestion des blocs pour la création des nouveaux noeuds (*cf* 3.5 page 24) : la détermination de l'interface apparaissait alors comme extrêmement importante pour lever les ambiguïtés (surtout en 3D). Cependant, il faut signaler qu'alors la consommation de mémoire sera beaucoup plus importante dans le cas d'éléments d'ordre élevé car les structures utilisées pour la détermination de l'interface/frontière sont plus généralement utilisées pour intégration des nouveaux noeuds (hachage complet du sous-domaine et non seulement de l'interface).

Finalement, nous pouvons résumer l'algorithme utilisé comme suit :
Pour chaque domaine

- déterminer sa propre interface en tant qu'ensemble de faces
- construire une table associant ces faces à un ensemble de numéros de domaine

Pour chaque domaine i à tour de rôle faire
envoyer aux autres domaines $j \neq i$ la définition de son interface (faces).
Parallèlement
Pour chaque domaine $j \neq i$ faire
recevoir la définition de l'interface du domaine i
ajouter pour chaque face reçue (de l'interface de) i) l'indice i à la table des domaines de la face mais seulement si cette face est commune avec le domaine j (il suffit de regarder dans la liste)

Après toutes les communications précédentes
Chaque domaine peut alors inverser sa table face→liste_de_domaines pour obtenir une table du type liste_de_domaines→faces qu'il suffira de numéroter (globalement *cf 3.4 page 23*)

Une notion ne doit pas être oubliée : la structuration en blocs dont l'objectif est de savoir, pour tous les noeuds (construits un peu plus tard), à quel bloc de communication ils appartiennent.

C'est pourquoi il faut synchroniser (au sens de la méthode précédente) tous les parties utiles à l'insertion des futurs noeuds. Ceci implique donc de construire suivant le même processus les associations entre parties (au sens de sommet, arête, face) et groupe de processeurs utilisateurs de cette partie.

Une première idée fut de ne communiquer que les faces (ou chaque processeur cherche à reconnaître des parties communes), puis de reconstruire les autres dimensions utiles (sommet, arête) par une déduction issue de la décomposition de la partie (face) en sous-partie.

Avertissement : la synchronisation sur une information réduite (comme les sommets, arêtes, faces) suivie d'une « reconstruction » afin de construire l'ensemble des autres blocs, me paraît maintenant illusoire (j'ai personnellement recensé formellement toutes les possibilités, avec toujours un cas conflictuel possible !) dans le cas 3D (toutefois possible en 2D) et doit plutôt utiliser une synchronisation plus sûre par les faces (qui représentent bien l'ensemble des interfaces) non pas suivi d'une reconstruction globale, mais d'une synchronisation identique sur les sous-parties (avec laquelle nous obtiendrons la table parties(arêtes ou sommets)→liste_de_domaines).

Cependant, cette démarche peut être assouplie, en notant que cette seconde synchronisation peut s'effectuer sans communication supplémentaire. En effet, le principe de décomposition peut très bien être appliqué pour définir les parties à synchroniser sans fausser le système si cela est fait dans un but de définition de parties et non de reconstruction de blocs.

Jusqu'à présent, il n'a pas encore été question de centralisation. Mais, un point important est à prendre en compte : il faut que chaque processeur utilisateur d'un bloc puisse communiquer ses noeuds à ses voisins (défini à l'aide du groupe de processeur précédemment formé), avec le maximum de flexibilité autant temporelle (le moment ou l'envoyer) que descriptive (simplifier l'identification de la communication). C'est à ce niveau qu'intervient la petite partie centralisée. Son objectif est de construire une identification efficace et simple. Ceci tient en deux points :

1. Donner un identifiant globalement reconnu pour chaque bloc. (3.4 page 23)
   Ce point fut un choix, car une méthode plus *Distribuée* était aussi possible mais plus complexe, pour des gains négligeables.
2. Donner une structure de réduction commune (au sens de distribuée) et souple. (3.3)

3.3 Synchronisation en action

Mais il ne faut pas oublier la finalité de ces blocs : la synchronisation !

Dans tous les cas, il faudra que les contributions des blocs soient reparties sur tous les processeurs concernés (la répartition dans le cas présent est en fait une sommation des contributions)

Pour cela, en utilisant MPI, plusieurs solutions sont offertes : les fonctions MPI_Reduce, MPI_AllReduce, MPI_Gather, MPI_BroadCast.

Cet ensemble de fonctionnalités offre la possibilité de recueillir une information sur les différents processeurs puis d'en redistribuer une valeur résultant d'une opération (dans notre cas, une somme). De plus, ces communications peuvent être limitées à des groupes de processeurs, en définissant des communicateurs.

Où est donc le problème ?

Supposons que 2 domaines distincts veuillent effectuer une réduction sur 2 blocs distincts mais ayant des processeurs communs. Alors, de par le fait que toutes les fonctionnalités de MPI précédemment évoquées sont bloquantes (la fonction ne se termine que lorsque la communication a **complètement** été effectuée), il pourrait y avoir présence d'inter-blocage (deadlock), c'est à dire que chacun attendrait une réponse de l'autre sans issue possible. Il n'est pas certain que cela se produira, mais suivant l'implémentation de la bibliothèque et du phénomène bloquant cela pourrait entraîner d'éventuels problèmes. C'est pourquoi, une version *à la main* et spécifiquement non bloquante a été réalisée.

3.3.1 Représentation du modèle de réduction non bloquante

Le plus évident, et communément utilisé [2], consiste en une réduction de type arbre binaire ; cependant bien qu'évident, détaillons ce point afin d'en déterminer une implémentation personnelle efficace.

La représentation précédente, montre sur la partie supérieure les différents processeurs voulant effectuer une réduction. Descendre dans l'arbre signifie simplement que les 2 processeurs issus de 2 branches se rejoignant, réduisent partiellement leurs informations de telle manière que ce soit celui dont le processeur de numéro figurant à l'intersection qui obtienne la valeur réduite partielle. La réduction est bien entendue totalement effectuée en atteignant la racine de l'arbre. Les noeuds de l'arbre représentent les communications. Par exemple, les 2 processeurs 2 et 4 se réduisent en 4, seulement lorsque à la fois 2 et 4 sont déjà les résultats de réductions partielles.

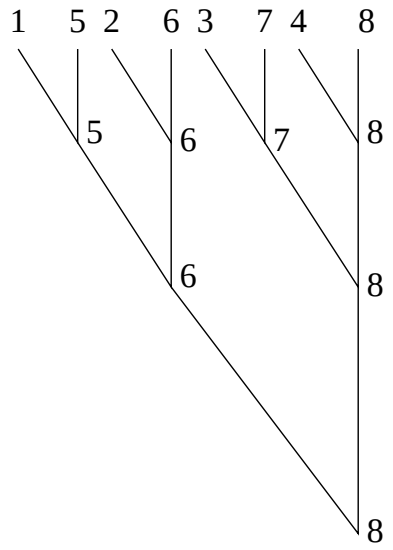

FIG. 5: Arbre binaire de réduction

En supprimant le temps dans le schéma précédent, nous pouvons obtenir le schéma ci-dessous :

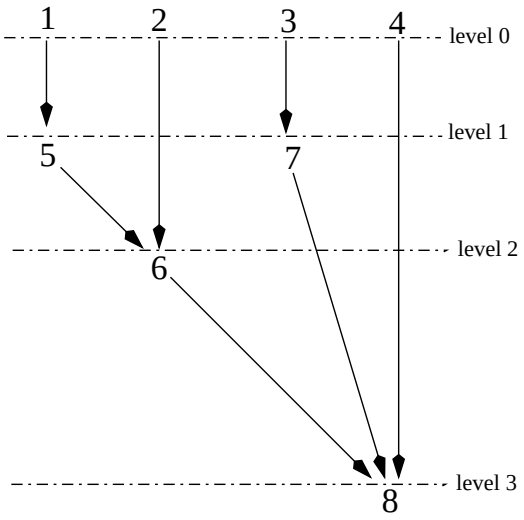

FIG. 6: Arbre binaire asynchrone

Ici, chaque noeud correspond à un processeur, et cet arbre (non binaire), permet de gagner en asynchronicité, par le fait que l'arité d'un noeud (nombre de parents, que nous appellerons *niveau* par la suite) est en relation avec le niveau de réduction nécessaire pour l'atteindre.

Ex : 1 et 5 se réduisent en 5, 2 et 6 en 6, puis 5 et 6 en 6, mais en tenant compte du temps (cas synchrone) les communications $2 \rightarrow 6$ et $5 \rightarrow 6$ s'effectueraient à la suite (en effet, le nombre de réductions pour atteindre ce point est différent suivant le chemin parcouru), et donc avec peu de concurrence d'accès ($5 \rightarrow 6$ se produira après la réduction $1 \rightarrow 5$) ; il faut mettre ceci en relation avec le fait dans le schéma précédent que $5 \rightarrow 6$ attend $2 \rightarrow 6$ alors qu'ils ont le même but : se réduire en 6.

On voit que la notion de niveau signifie a peu près le nombre de réductions à effectuer avant d'avoir réduit la position. (exact pour 2^n processeurs).

Dans cette approche chaque processeur n'a besoin de connaître que son suivant et son niveau (le niveau peut être alors vu comme le nombre de requêtes attendues avant la poursuite du schéma)

3.3.2 Algorithme de réduction non bloquante

Il reste une difficulté : le niveau n'est pas évident à déterminer pour un nombre de processeurs quelconque. C'est pourquoi une méthode incrémentale sera appliqué pour le déterminer (si $i \rightarrow j$ alors le niveau de j augmente de 1).

Réduction du groupe de processeurs nommés cpu_i pour $i = 0..order - 1$ par la détermination de $next_i$ et de $level_i$ (suivant et niveau du cpu_i)

```
r = order
i = 0
while (r > 1) do
  m = r mod 2;
  r = r div 2;
  for j=0..r-1 do
    next_i = cpu_{i+r}
    level_{i+r} = level_{i+r} + 1
    i = i + 1
  enddo
  r = r + m
enddo
next_i = undef (la fin)
```

La figure 7 montre des exemples, dans des cas non triviaux, pour 7 et 9 processeurs.

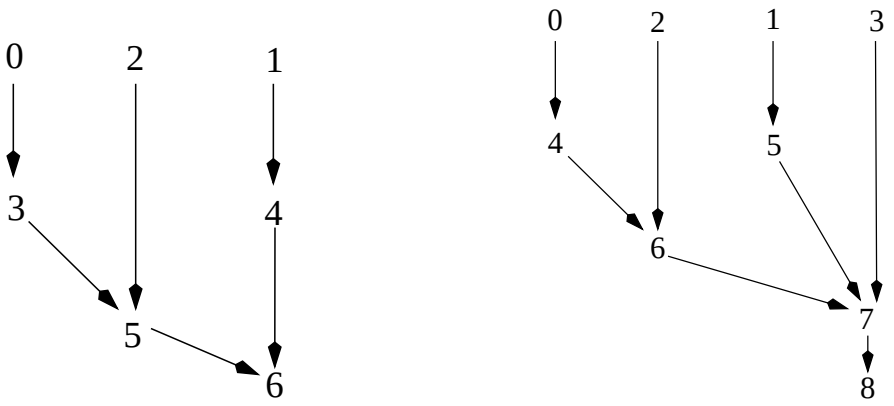

FIG. 7: Réduction non-bloquante, exemples avec 7 et 9 processeurs

Ce dernier cas peut faire apparaître un problème sur $7 \rightarrow 8$. En fait, il doit être possible, de trouver un algorithme qui équilibre plus les arbres, mais ce point étant peu limitatif (car 8 n'attend qu'une seule fois) n'a donc pas été plus développé.

L'utilisation est alors simple : chaque cpu_i attend $level_i$ requêtes avant de continuer. Une fois ceci fait, il effectue une réduction partielle avec $next_i$ sauf si celui-ci vaut $undef$ auquel cas la réduction est terminée.

Le caractère non bloquant est lié au fait que toutes les réceptions de communications peuvent être postées (i.e émises) avant l'attente de leur arrivée (à chaque réception, le processeur effectue la réduction partielle adaptée).

Il est à noter que lors des réductions, nous n'aurons besoin que d'un seul buffer de taille correspondant au nombre de noeuds interfaces par processeur (les noeuds internes ne subissant pas de synchronisation !). Chaque bloc n'attend qu'une seule requête à la fois (en mode synchrone non bloquant : Issend-Irecv de MPI) ce qui limite ainsi la surcharge réseau (pas d'effet de bufferisation supplémentaire).

3.3.3 Diffusion de la réduction à tout le groupe

Ce point est simplement la transmission de la valeur réduite servant à la synchronisation à tous les individus du groupe de processeurs. Pour cela une méthode standard de diffusion exponentielle a été utilisée : chaque processeur recevant la valeur réduite, essaie de la propager à 2 de ses voisins définis suivant un arbre de diffusion partant de la racine de la réduction. Le terme « essaie » signifie en fait que cette diffusion est très rapide en nombre d'étapes et que beaucoup n'auront pas à effectuer cette réduction.

Symboliquement, cet arbre de diffusion est « l'inverse » de l'arbre de réduction.

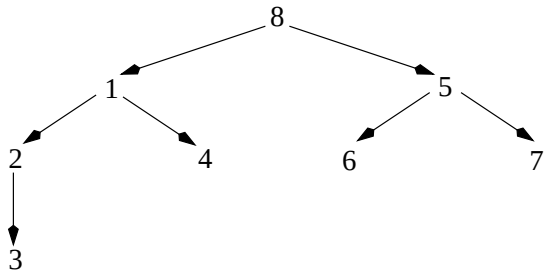

FIG. 8: Arbre de diffusion

3.3.4 Avantage en terme de quantité de communications

Pour voir l'asynchronicité et l'efficacité potentielle de la méthode, on remarquera que pour la synchronisation de *n* blocs un processeur a toujours au plus *n* communications en attente pour *n* blocs différents, ce qui nous permet d'allouer les zones de réceptions et de les utiliser directement « *en place* » lors des réceptions des diffusions. (Communications synchrones sans aucune zone temporaire utilisée pour recopie).

3.3.5 Inconvénients de répartition non uniforme

Un point peut être soulevé quant aux performances théoriques de ce modèle. En effet, dans les schémas précédents, les processeurs d'ordre faible émettait plus de blocs qu'ils n'en attendaient, et ceux d'ordre élevé en attendent plus qu'ils n'en émettent. Et en poussant à l'extrême, on voit que le premier processeur 0 n'attend jamais, et ne participe pas vraiment à la réduction en tant que calcul (il ne calcule jamais de somme partielle).

Ce déséquilibre peut être résolu soit en adaptant les charges des différents processeurs aux nombres de sommes partielles effectuées, soit en équilibrant aléatoirement par une permutation sur l'ensemble des indices. C'est ce dernier choix qui a été retenu, pour sa simplicité.

Des tests sur réseau de stations DEC-Alpha ont permis de valider cette démarche (des gains de 20% sont constatés). Mais le même test sur machine parallèle HP 9000/800 HP-UX 11.0 à mémoire partagée, a donné un résultat nettement moins significatif, ce qui conduit à penser que l'architecture du système de communication a une influence sur ce point (réseau Ethernet contre mémoire partagée). Ce mode peut donc activé ou non à la compilation suivant le cas que rencontrera l'utilisateur.

3.4 Choix et *implémentation*

Ainsi, la gestion de la création des blocs est toujours centralisé autour d'un Maître de gestion (afin d'assurer un identificateur global non conflictuel) mais ne veut pas dire que les esclaves doivent en abuser, en appelant systématiquement le maître pour le moindre problème de construction de blocs triviaux. Ainsi, un modèle plus élaboré nécessite que chaque esclave ait une mémoire additionnelle (cache) évitant ainsi de *poser plusieurs fois la même question pour une même réponse*.

Le choix retenu sera donc :

- La méthode globalisante
- Communications de type réseau
- Analyse de l'interface préalable.

L'implémentation est alors fort simple, mis à part la numérotation globale des blocs.

A ce niveau, l'idée de globale fait toujours intervenir implicitement l'idée de Maître. Cependant, ce maître est très peu actif et son assimilation à un processus très léger est possible mais se révèle désastreux au niveau des performances. L'implémentation sera donc toujours avec un processus hébergeant le Maître (vrai et unique), se chargeant tout d'abord des besoins locaux de l'hôte, puis servant à la demande ses homologues. Cette implémentation tient désormais compte de la nouvelle « intelligence » des esclaves permettant d'éviter des requêtes excessive au Maître (qui, vu localement, ressemble à un serveur local qui regarde s'il connaît la réponse et le cas échéant en fait une requête au vrai Maître de gestion).

Ainsi, on a la structuration multi-niveaux suivante qui idéalise le procédé avec un serveur local

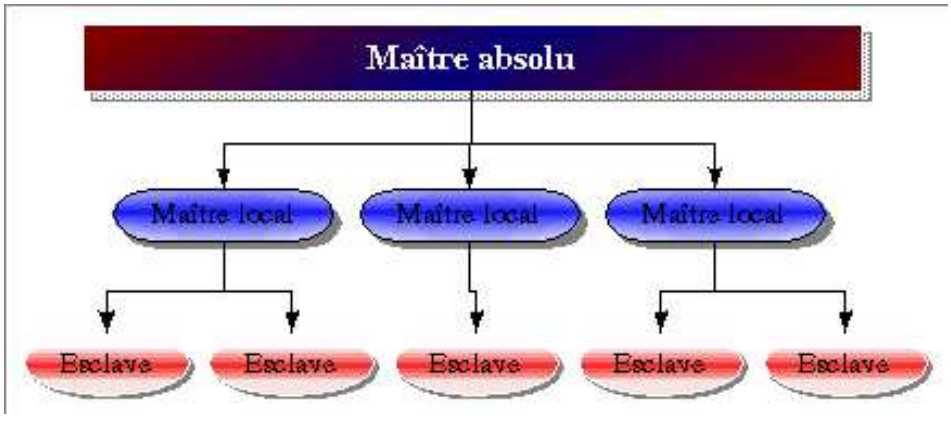

FIG. 9: Structure avec serveur local

par complexité de communication. Sur un réseau de machine multiprocesseurs à mémoire partagée, un serveur local s'occuperait de tous les esclaves représentés par les processeurs locaux et le maître absolu s'occuperait de l'ensemble des serveurs locaux.

Ceci peut être représenté plus simplement dans notre cas en prenant comme serveur principal (maître absolu) un des serveurs locaux eux-mêmes en charge d'un seul processus esclave.

Voici donc le schéma implémenté reprenant les mêmes représentations que la figure précédente.

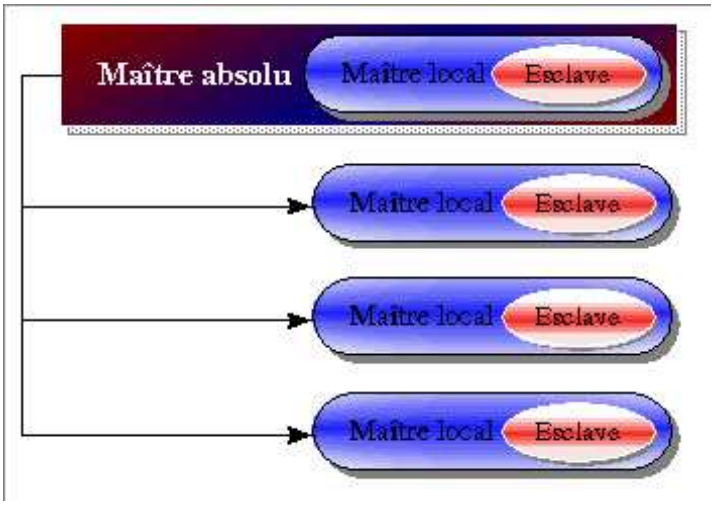

FIG. 10: Serveur « absolu »

Ce qu'il faut y voir, c'est la hiérarchisation des requêtes suivant la complexité à les traiter, en particulier si elles sont issues de la création de nouveaux blocs. Ceci permet donc de fournir une sorte de mémoire cache (rapide d'accès) et de rendre plus intelligent les couples esclaves/serveurs locaux afin de ne plus surcharger inutilement le serveur principal.

3.5 Ajout « synchrone » de nouveaux points.

Nous allons à présent analyser l'étape d'ajout de points ou de noeuds au sein des domaines locaux de manière à conserver à l'ensemble une cohérence nécessaire. C'est ce point qui, allié au pré-maillage et à la structure par blocs des communications, permet des gains d'espace (disque et mémoire) ainsi qu'un temps d'analyse des blocs réduit. C'est tout à fait ce que nous avons dans le cas des éléments d'ordre élevés. Ce principe vise donc à ajouter de manière cohérente (principalement sur les interfaces) les nouveaux noeuds.

3.5.1 Structuration de l'information

Ainsi, il faut pouvoir résoudre notre problème complet en reconstruisant les informations manquantes dues aux abstractions faites sur le pré-maillage et la généricité des éléments utilisables tout en conservant notre synchronisation par blocs.

Pour bien voir le problème à résoudre, détaillons la future phase de synchronisation des calculs au niveau du solveur parallèle : dans cette étape, chaque processeur devra se synchroniser avec ses voisins ; une telle synchronisation est en faite la réduction suivant la méthode précédente (*cf* 3.3 page 19) non pas de simples valeurs, mais de vecteurs de données tout entier dont la valeur réduite devra bien entendue être la somme vectorielle résultante. Ceci dit, il faut que chaque composante des différents vecteurs coïncide sur les différents processeurs en terme d'association de la valeur à un même noeud. Alors, une idée simple, est de les ranger dans un ordre quelconque mais fixé. Cependant, il ne faut pas oublier que ces blocs ne sont pas composés des noeuds du maillage, mais de sommets

grossiers qui dans notre modèle général, ne seront peut être pas des noeuds du maillage définitif, et qui ne tiennent pas non plus compte d'un éventuel remaillage interne à chaque domaine !

Pour préciser ce point, nous pouvons déjà voir que l'ajout de noeuds ou des points (resp. pour construire les éléments et remailler localement les domaines) peut être traité d'une seule manière (au sens du respect des blocs).

L'idée est que le positionnement de points sur le bord d'un domaine (aux interfaces) doit rester synchrone avec ses domaines voisins, dans la mesure où les blocs doivent toujours être composés des mêmes noeuds et dans le même ordre, tout du moins, un ordre que l'on puisse retrouver ; c'est ici que les méthodes directes nécessiteraient des indirections pour conserver les blocs et donner un profil optimal. Cependant, dans le cas explicite qui nous occupe ici, nous éviterons les indirections en utilisant l'ordre pour former des vecteurs de synchronisation parfaits. Pour cela, il faut voir que lorsque l'on doit ajouter un point, c'est toujours par rapport à une position relative sur un segment, une face, un volume (le cas « un point » étant anecdotique). Nous qualifierons de parents les points servant à définir cette position relative ; de plus nous les considérerons toujours en nombre minimum. Ainsi, un point d'un segment n'a besoin que de 2 parents pour être défini sans ambiguïté. Dès lors, la position peut s'exprimer par des coordonnées barycentriques associées aux parents du point désiré.

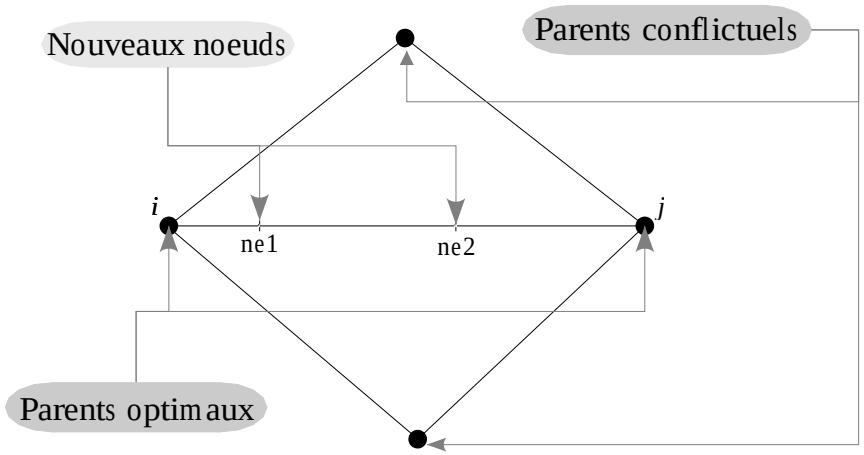

FIG. 11: Création des nouveaux noeuds

Cette figure exprime la notion de parents optimaux, et de parents conflictuels ; les parents conflictuels peuvent être vus comme des parents potentiels auxquels il serait associé des coordonnées barycentriques de valeurs 0, donc inutiles. Soient ne1 et ne2, 2 nouveaux noeuds sur le segment (i, j) avec respectivement les coordonnées barycentriques (α_1, β_1) et (α_2, β_2). L'idée de la synchronisation est que chaque domaine autour du segment (i, j) doit ranger les nouveaux noeuds exactement à la même position dans le vecteur de final, et ce, avec le minimum de communication (voir même sans aucune communication). Cette approche sera totalement extensible aux dimensions supérieurs ; ainsi en 3D, une arête doit pouvoir être partagée par plus de 2 domaines, et sur une face, l'ordre ne doit pas être perturbé par le fait qu'il n'y pas d'ordre standard pour ranger les nouveaux noeuds toujours dans le même ordre sur tous les domaines. Le problème de l'ordre peut être résolu par l'ordre lexicographique, mais la difficulté est qu'il est peu stable sur les réels. Pour lever partiellement ce problème, il faut ajouter une base servant à stabiliser au mieux les opérations sur les coordonnées barycentriques : un exemple pratique serait, dans un cas de remaillage, l'ajout de point suivant une progression géométrique ; le problème des perturbations pourrait donner un résultat faux juste par le fait que les parents

ont été pris dans un ordre différent ! !. C'est pourquoi, un ordre sur les parents doit aussi être donné : Ordre arbitraire mais constant pour un même ensemble de parents. Or la seule information stable actuellement partagée par tous les domaines a propos de leurs sommets est leurs numéros globaux dans le pré-maillage : c'est donc sur cette valeur que sera effectué le rangement des parents. Alors, même sans aucune communication, chaque processeur, peut ranger ces nouveaux points correctement et de manière commune à tous les processeurs utilisant les mêmes blocs. De plus, le rangement des parents (accompagnés de leur progéniture) suit encore l'ordre lexicographique pour sa stabilité et simplicité pour des valeurs entières.

Pour résumer :

1. rangement des parents optimaux suivant leur index global dans le pré-maillage (pour chaque partie : point, arête, face, volume)
2. rangement des noeuds enfants (issus d'un ensemble de parents) suivant l'ordre lexicographique (réel).
3. rangement de l'ensemble (*parents, liste ordonnée des coordonnées barycentriques*) dans une table lexicographiquement ordonnée (entier) suivant l'ensemble des parents.

Il faut préciser que la notion de parents optimaux est fondamentale ; si des parents conflictuels intervenaient dans l'ensemble des parents, bien que cela ne change rien sur le positionnement dans l'espace car étant associés à une composante barycentrique nulle, cela perturberait l'ordre de la table du *3.* du résumé d'analyse précédent. Une généralisation de ce principe est que si un sommet grossier est aussi un noeud (ce qui n'est pas obligatoire dans notre conception), ce noeud aura comme unique parent son sommet, avec une valeur de coordonnée barycentrique égale à 1.

4 Résultats numériques – performances

Nous présentons maintenant un ensemble de résultats numériques pour valider et illustrer les performances de la méthode. Nous renvoyons également aux précédents travaux[17, 7] pour des comparaisons plus détaillées entre les différents éléments en 2D.

4.1 Exemples de validation

4.1.1 Milieu homogène, dimension 2

Nous nous plaçons dans le carré $[-5,5]\times[-5,5]$ supposé homogène. Nous prenons une source de la forme : $S(x,t) = f(x)g(t)$ avec

$$f(x) = 1000\exp(-5(x^2+y^2)) \tag{29}$$

et

$$g(t) = 2\left(\frac{\pi}{t_0}\right)^2\left(2\left(\frac{\pi}{t_0}\right)^2(t-t_0)^2 - 1\right)e^{-\left(\frac{\pi}{t_0}\right)^2(t-t_0)^2} \tag{30}$$

avec $t_0 = 1.35$ s (une telle source est couramment utilisée dans la littérature géophysique, ou elle s'appelle une ondelette de Ricker. IL s'agit en fait de la dérivée seconde d'une gaussienne). Elle est représentée sur la figure 12.

La figure 13 montre un instantané à $t = 3$ s (calculé en P^1), qui montre que l'on conserve bien le caractère symétrique de la solution.

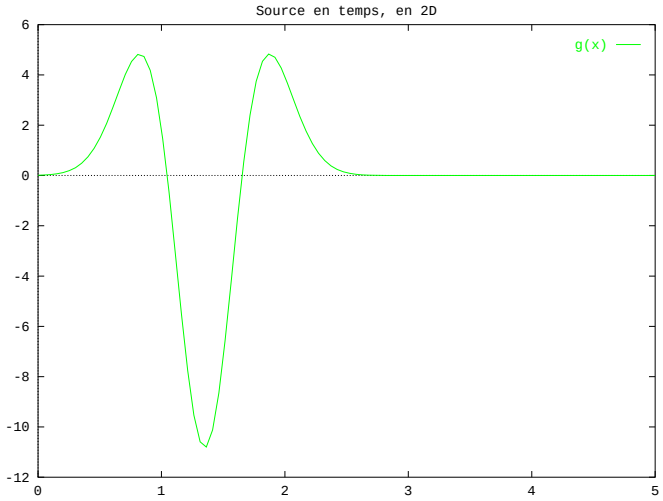

FIG. 12: L'ondelette de Ricker

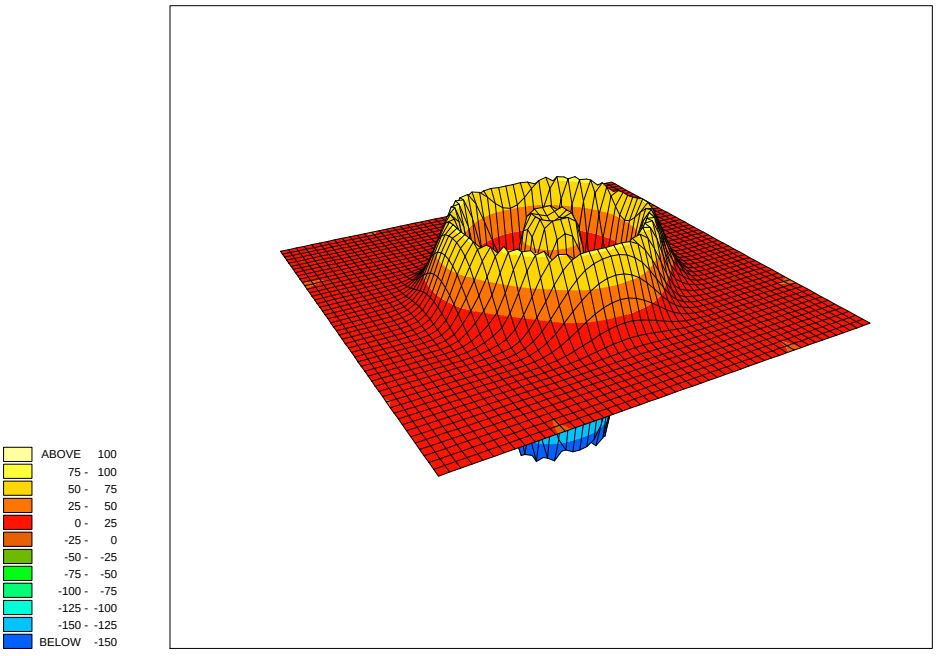

FIG. 13: Instantané à $= 3s$

Pour cet exemple, il n'existe pas de solution analytique (simple). Nous avons donc comparé nos résultats à ceux d'un code réalisé par S. Fauqueux, qui utilise des éléments finis spectraux sur un maillage quadrangulaire [6]. Nous avons comparé les éléments du second et du quatrième ordre avec un élément essentiellement d'ordre 10. Les résultats sont présentés sur la figure 14.

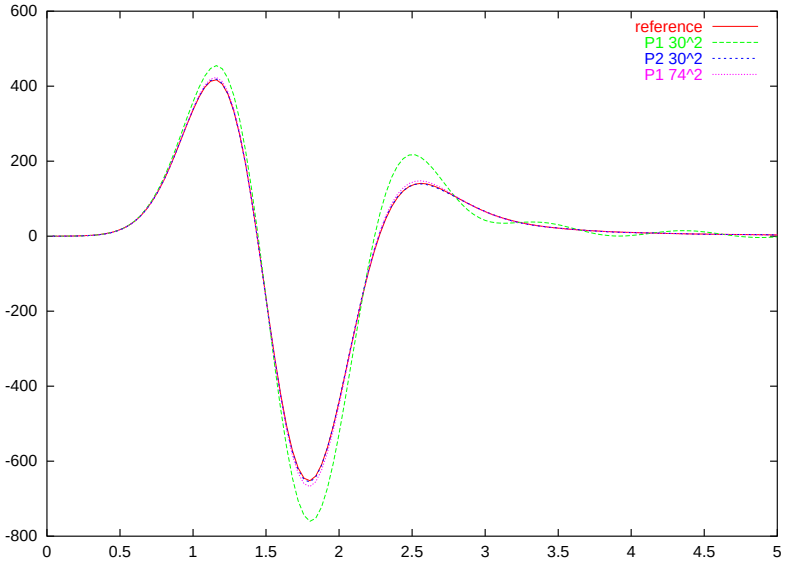

FIG. 14: Comparaison des éléments d'ordre 2 et 4 avec un code spectral

Cette figure représente une coupe de la solution selon l'axe des abcisses, à un temps fixé.

La ligne marquée « référence » correspond au code de S. Fauqueux (la précision est telle que son résultat peut être pris comme la solution exacte). La ligne marquée $P130\hat{2}$ (qui se distingue des autres) correspond à l'élément fini usuel sur un maillage avec 30 points par coté, et de même la ligne marquée $P230\hat{2}$ correspond au nouvel élément d'ordre 4. Enfin la ligne marquée $P174\hat{2}$ correspond à l'élément fini usuel avec c74 points par coté, de façon à ce que le nombre de degrés de libertés soit égal à celui pour l'élément d'ordre 4 avec 30 points par cotés.

Il est clair, et bien connu, que l'élément P^1 usuel demande un grand nombre de points pour obtenir une précision satisfaisante. En fait, même avec le même nombre total de degrés de libertés, le nouvel élément fini donne une meilleure précision. Comme son coût n'est pas beaucoup plus élevé, cet élément est très compétitif, et cet avantage s'accentue aux temps longs, comme cela a été montré dans [7].

4.1.2 Conditions absorbantes

Le but de cet exemple est présenté les possibilités de conditions aux limites que propose le code. Celles-ci peuvent être de 2 type.

- Conditions absorbantes du premier ordre
- Conditions de Dirichlet ($u = 0$)

Rappelons que les conditions absorbantes du premier ordre se traduisent par la formulation

$$\frac{\partial u}{\partial \nu}(x,t) + \alpha(x)\frac{\partial u}{\partial t}(x,t) + \beta(x)u(x,t) = h \otimes g_2(x,t), \forall x \in \partial\Omega, \quad \forall t \in]0,T[$$

Dans notre approche, nous prendrons α et β constants par face.

Notons que en tant que conditions absorbantes, le premier ordre est à peu près tout ce que l'on puisse faire pour des éléments finis sans pour autant passer aux PML qui seront envisagées dans un futur proche ; ceci expliquera le relative mauvaise qualité de l'absorption.

Notre approche actuelle du cas Dirichlet n'est peut être pas optimale au sens où la formulation mathématique l'aborde comme une absence de de degré de liberté alors que nous dans notre approche nous l'envisageons comme un forçage à zéro. Cette approche est en étude pour se rapprocher de l'approche mathématique qui est de surcroît plus économique en terme de calcul.

L'exemple présenté ici propose de comparer l'efficacité relative des conditions absorbantes du premier ordre par rapport à celles de Neumann.

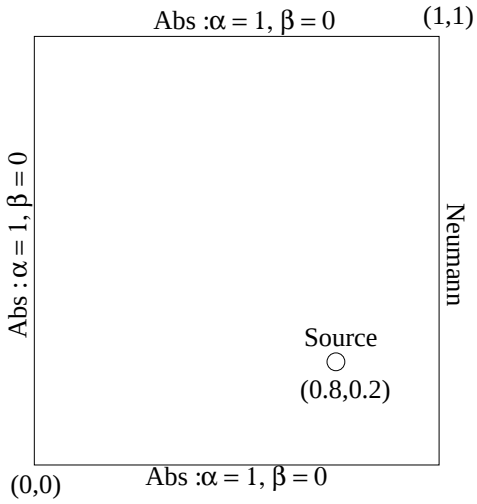

FIG. 15: Expérience avec conditions limites absorbantes

- Le maillage est régulier, de dimension 200×200 formé de triangles rectangles.
- La source spatiale est de type Dirac en $(0.8, 0.2)$ pondéré par l'inverse du carré du pas d'espace.
- La variation en temps de la source spatiale est de défini par la dérivée d'une gaussienne avec une fréquence de 30 Hz.

La solution à l'instant $t = 0.5$s est représentée, sous deux vues différentes, sur les figures 16et 17. On constate que le bord absorbant joue son rôle, quoique la condition du premier ordre n'offre qu'une absorption limitée.

4.1.3 Milieu homogène, dimension 3

Nous réalisons la même expérience en dimension 3, en prenant cette fois un cube de demi-coté 5. La source est la même qu'au paragraphe précédent (la fonction f dépend également de z). Cette fois, il est facile de comparer les méthodes numériques à une solution exacte. Étant donné une fonction $u_0(r)$, la fonction

$$u(r,t) = \frac{1}{r} u_0(r-t)$$

est solution de l'équation des ondes avec la vitesse 1, au moins tant que cette solution n'a pas atteint le bord, c'est-à-dire ici pour $t \leq 5$.

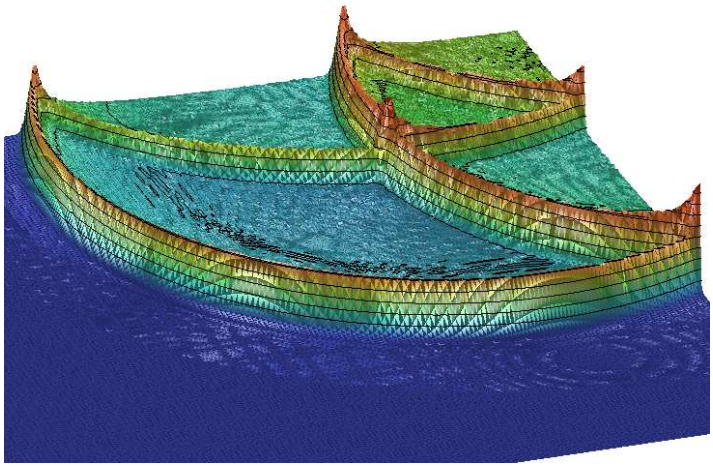

FIG. 16: Éléments P2 Visualisés P1 : $t = 0.5s$ avec 240000 Degrés de liberté

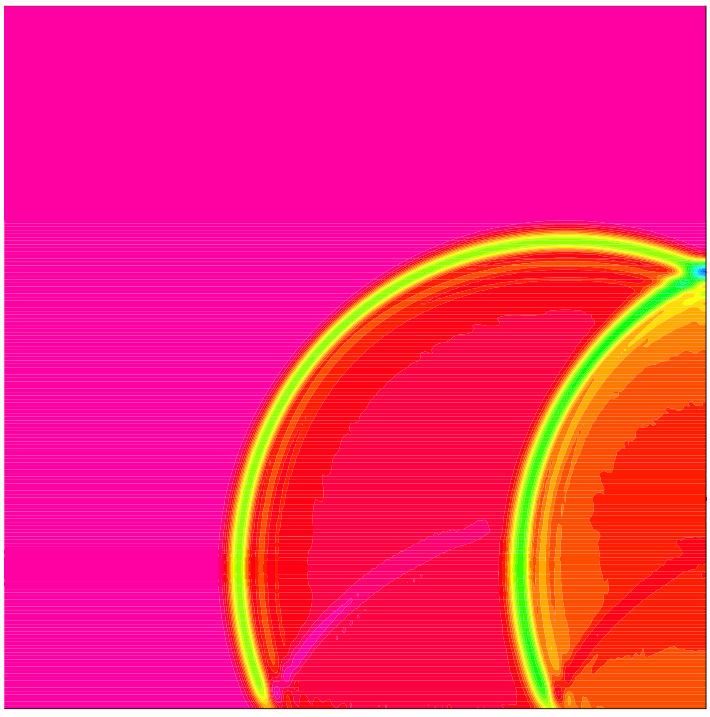

FIG. 17: Source normalisée et maillage 300×300 avec variation en temps adaptée montrant un « bon » cas pour les conditions absorbantes du premier ordre [$t = 0.5s$]

Nous avons choisi la fonction

$$u_0(r) = \begin{cases} (2r_1 - |r|)(2r_0 - |r|), & \text{si } 2r_0 \leq r \leq 2r_1 \\ 0 & \text{sinon.} \end{cases}$$

Nous comparons, sur la figure 18, la solution analytique à des simulations réalisées avec les éléments d'ordre 2 et 4. Le maillage P^1 comporte 110 noeuds sur chaque face du cube, le maillage P^2 en comporte 30.

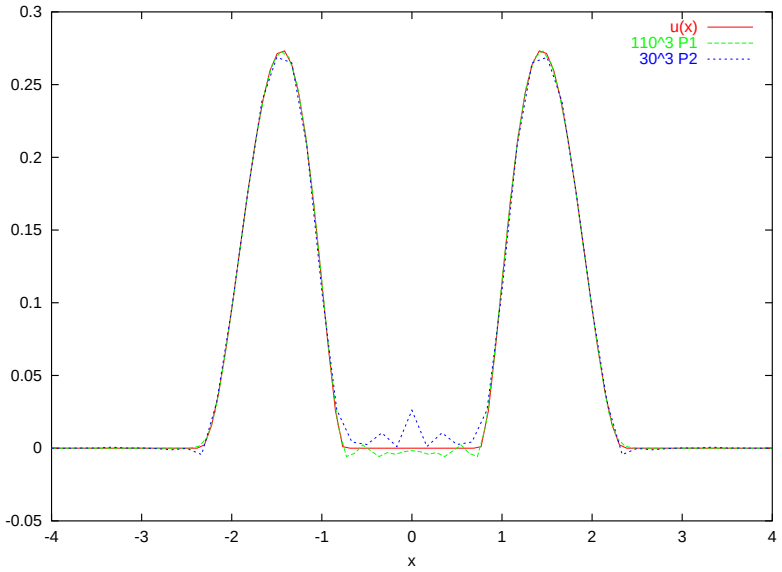

FIG. 18: Comparaison d'une solution analytique avec des simulations – coupe sur l'axe des abcisses

La situation est ici moins satisfaisante qu'en 2 dimensions. Si l'élément usuel donne un résultat satisfaisant, ce n'est pas encore le cas pour le nouvel élément avec le maillage utilisé. Or il faut savoir que ce calcul comporte près de 3 millions de degrés de libertés ! Un autre facteur limitant est que la condition de stabilité du nouvel élément semble être très petite. Nous avons constaté expérimentalement sur cet exemple une CFL de 0.018. La combinaison du grand nombre de degrés de libertés avec une faible CFL rend ce nouvel élément beaucoup moins compétitif en 3D qu'en 2D.

4.2 Exemples en milieu hétérogène

4.2.1 Milieu à deux couches

L'exemple suivant est inspiré d'un modèle utilisé par Bangerth et Rannacher [3] pour évaluer un algorithme de raffinement de maillage. Il s'agit d'un modèle 2D à deux couches. La condition initiale est concentrée dans un carré de coté 0.1 (le domaine est le carré unité, la vitesse vaut 1 pour $y < 0.2$ et 3 pour $y > 0.2$), ce qui force à utiliser un maillage très fin.

La condition initiale est donnée par :

$$u_0(x,y) = \begin{cases} \exp(-100(x^2+y^2))(1-100(x^2+y^2)) & \text{pour } x^2+y^2 \leq 0.01 \\ 0 & \text{sinon.} \end{cases} \tag{31}$$

Nous présentons sur la figure 19 des instantanés calculés avec les éléments du second et du quatrième ordre, sur un même maillage (donc avec un nombre différent de degrés de liberté). La discontinuité de vitesse est bien représentée. L'amélioration due à l'élément du quatrième ordre saute aux yeux.

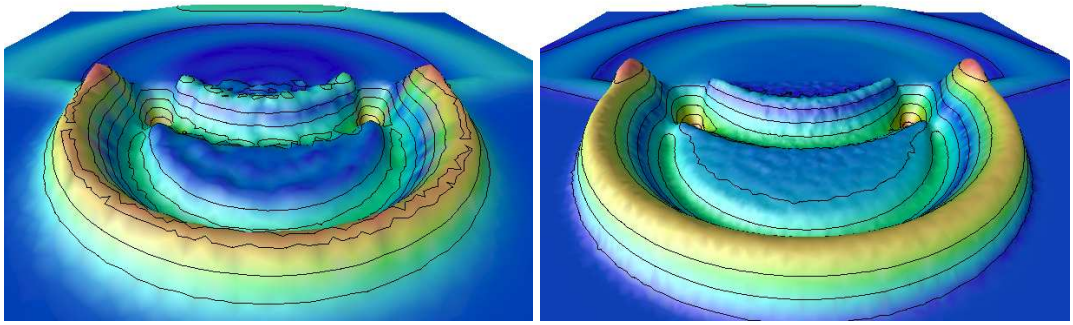

FIG. 19: À gauche : élément du second ordre, à droite : élément du quatrième ordre

Le calcul en P^1 a demandé 4417 degrés de libertés, et 2600 avec $\tilde{P}^2$.

4.2.2 Milieu complexe en 3D

Cet exemple est cette fois inspiré d'un modèle fourni par l' IFP que nous avons simplifié. Le modèle original est un maillage en différences finies de taille $476 \times 476 \times 326$ (avec un pas de grille de 10 m). Nous avons réduit les dimension latérales et le nombre de couches, pour obtenir le modèle présenté sur la figure 20. Les dimensions du modèle sont de 3 km sur 2 km (de profondeur), et la vitesse varie de $1500\mathrm{m/s}$ (vitesse du son dans l'eau) dans la couche supérieure à $13000\mathrm{m/s}$ dans la couche la plus profonde.

Le maillage a été obtenu en combinant les outils Modulef avec le mailleur GHS3D développé dans le projet Gamma de l'INRIA. Le modèle‘le contient 500000 noeuds et 3 millions de tétraèdres. Les calculs ont été réalisés sur 20 noeuds de la grappe de PC du laboratoire ID de l'IMAG.

Deux instantanés sont représentés sur les figures 21 et 22.

4.3 Performance en parallèle

Nous avons mené plusieurs expériences pour évaluer l'efficacité parallèle du code.

4.4 Mémoire partagée

Nous avons tout d'abord eu accès à un calculateur de 16 processeurs HP à mémoire partagée. Il est toujours possible d'utiliser MPI sur ce type de machine, et de laisser le système d'exploitation utiliser au mieux le matériel.

Le premier exemple est simplement un cube discrétisé par un maillage régulier avec 110 points sur chaque coté (chaque cube est subdivisé en 6 tétraèdres). Cela correspond à un total de 1 367 631 sommets. La table 1 présente les résultats obtenus pour 200 pas de temps.

Ces résultats sont assez décevants. On peut avancer deux raisons à cela : la première est que le maillage est en fait trop petit pour donner une bonne efficacité, et la seconde (qui est liée) est que les processeurs individuels sont trop puissants.

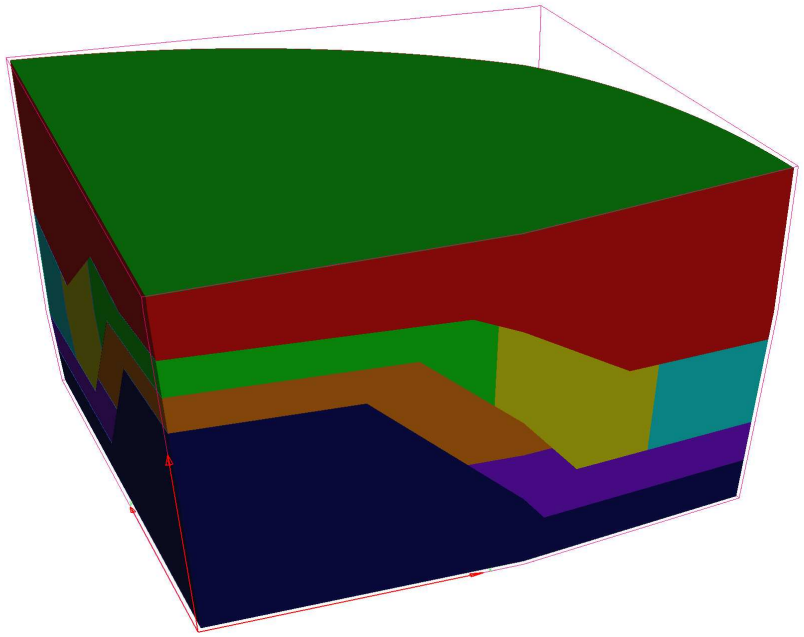

FIG. 20: Modèle hétérogène

Nbre proc.	4	8	16
DDLs locaux (x1000)	353	180	90
Initialisation (s)	766	535	408
Temps calcul (s)	243	150	122
Temps total (s)	1009	685	532
Accélération	1	1.47	1.89
Efficacité	1	0.74	0.48

TAB. 1: Performance en P^1 sur HP

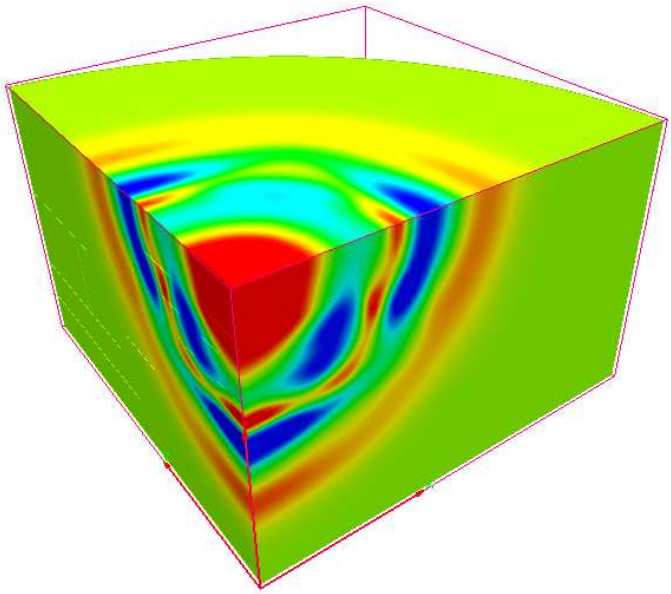

FIG. 21: Instantané à $t = 0.5s$

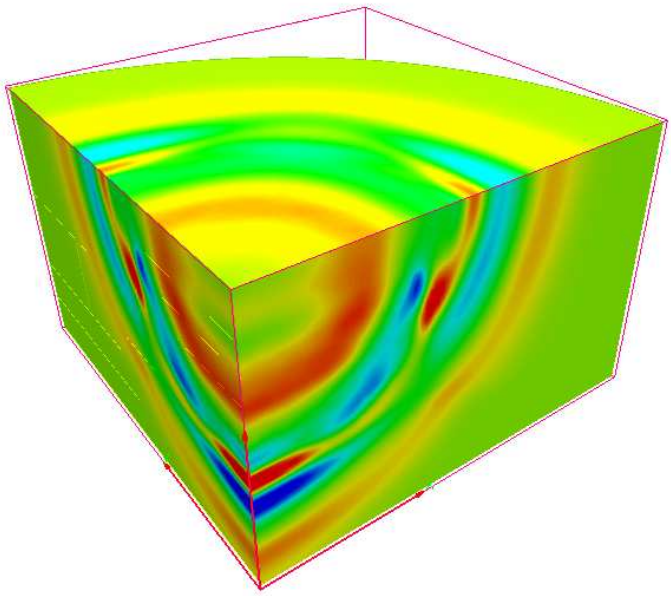

FIG. 22: Instantané à $t = 0.7s$

Nous avons ensuite utilisé 40 noeuds par coté, mais avec l'élément $\tilde{P}^2$, ce qui correspond à plus de 3 millions de degrés de liberté en tout (pour 68 000 noeuds).

Nbre proc.	8	12	16
DDLs locaux (x1000)	414	275	210
Initialisation (s)	103	74	62
Temps calcul (s)	528	369	282
Temps total (s)	631	443	344
Accélération	1	1.42	1.83
Efficacité	1	0.94	0.92

TAB. 2: Performance en $\tilde{P}^2$ sur HP

Les résultats sont nettement meilleurs, comme pouvait le laisser espérer le plus grand nombre de degrés de libertés.

4.5 Expérience sur T3E

Nous avons ensuite réalisé la même expérience (avec l'élément $\tilde{P}^2$) sur le Cray T3E de l'IDRIS. Il s'agit cette fois d'une « vraie » machine à mémoire partagée, avec un maximum de 256 processeurs.

Nbre proc.	Temps (s)	Accélération	Efficacité
64	645	64.0	1.00
80	611	67.6	0.84
96	501	82.4	0.85
112	438	94.2	0.84
128	407	101.4	0.79
144	363	113.7	0.79
160	354	116.6	0.73
176	347	119.0	0.68
192	314	132.5	0.68

TAB. 3: Performance parallèle sur le Cray T3E

Pour des raisons de place mémoire il n'est pas possible de faire tourner cet exemple sur moins que 64 processeurs. On constate que l'efficacité décroît régulièrement, ce qui est normal pour un problème de taille fixée. Elle reste assez bonne jusqu'à 144 processeurs.

Remerciements

Une partie des calculs a été réalisée à l'IDRIS.

Nous remercions le laboratoire ID pour l'accès à leur centre de ressources (`http://www-id.imag.fr/grappes.html`). Ce travail a utilisé la grappe de PC i-cluster de ID/HP.

Références

[1] R. M. Alford, R. Kelly, and D. M. Boore. Accuracy of the finite difference modelling of the acoustic wave equation. *Geophysics*, 39 :834–842, 1974.

[2] G.S. Almasi and A. Gottlieb. *Highly parallel computing*. The Benjamin/Cummings series in computer science and engineering. Benjamin/Cummings, 2nd edition, 1994.

[3] W. Bangerth and R. Rannacher. Finite element approximation of the acoustic wave equation : error control and mesh adaptation. *East-West J. Numer. Math.*, 7(4) :263–282, 1999.

[4] M. J. S. Chin-Joe-Kong, W. A. Mulder, and M. van Veldhuizen. Higher order triangular and tetrahedral finite elements with mass lumping for solving the wave equation. *Journal of Engineering Mechanics*, 1999. to appear.

[5] Philippe G. Ciarlet. *The Finite Element Method for Elliptic Problems*. North-Holland, 1976.

[6] G. Cohen and S. Fauqueux. Mixed finite elements with mass-lumping for the transient wave equation. *J. of Computational Acoustics*, 8(1) :171–188, 2000.

[7] Gary Cohen, Patrick Joly, Nathalie Tordjman, and Jean Roberts. Higher order triangular finite elements with mass lumping for the wave equation. *SIAM J. Num. Anal.*, 38(6) :2047–2078, 2000.

[8] R. Dautray and J.L. Lions. *Analyse Mathématique et Calcul Numerique pour les Sciences et les techniques*, volume 3. Masson, 1988.

[9] Pascal Frey and Paul-Louis George. *Maillage*. Hermes, 1999.

[10] Frédéric Hannoyer. Tetrahedral finite elements with mass lumping for solving the wave equation. Rapport de Stage, Ecole Polytechnique, 1996.

[11] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48 :96–129, 1998.

[12] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1) :359–392 (electronic), 1999.

[13] Christophe Lemuet. Développement d'un logiciel de résolution de l'équation des ondes par éléments finis d'ordre élevé. Rapport de stage de DEA, Université Paris VI, 1999.

[14] M. J. Lighthill. On sound generated aerodynamically : I. general theory. *Proc. Roy. Soc. London*, 1952.

[15] MPI Forum. MPI : A message passing interface standard. *Int. J. of Supercomputer Applications*, 8, 1994.

[16] W. A. Mulder. A comparison between higher-order finite elements and finite diffrences for solving the wave equation. In J.-A. Désidéri, P. Le Tallec, E. O nate, J. Périaux, and E. Stein, editors, *Proceedings of the Second ECCOMAS Conference on Numerical Methods in Engineering*, pages 344–350. Wiley, 1996.

[17] Natahlie Tordjman. *Eléments finis d'ordre élevé avec condensation de masse pour l'équation des ondes*. Thèse de doctorat, Université Paris IX Dauphine, 1995.

[18] R. Vichnevetsky and J. B. Bowles. *Fourier Analysis of Numerical Approximations of Hyperbolic Equations*. SIAM Stud. Appl. Math. SIAM, 1982.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifi que
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399