SCHNAPS : une bibliothèque pour la résolution numérique des lois de conservation sur ordinateur massivement multicoeur

IRMA Strasbourg

29 septembre 2014

Table des matières

1 Méthode Galerkin Discontinu (GD)

1.1 Systèmes de lois de conservation

SCHNAPS est un acronyme de "Solveur Conservatif Hyperbolique Non-linéaire Appliqué aux PlasmaS". Il s'agit d'une bibliothèque de classes C pour résoudre numériquement des systèmes de lois de conservation sur des ordinateurs massivement parallèles, avec plusieurs niveaux de parallélisme, par exemple un gros cluster de GPU.

SCHNAPS se veut assez général. Par conséquent, nous considérons un domaine ouvert borné $\Omega \subset \mathbb{R}^3$ de frontière $\partial\Omega$. Sur ce domaine, nous considérons un système de lois de conservation de la forme

$$\partial_t W + \partial_i F^i(W) = S(W). \tag{1}$$

Dans cette équation, l'inconnue $W(x,t)$ est un vecteur de $\mathbb{R}^m$ qui dépend d'une variable d'espace $x \in \Omega$, $x = (x_1, x_2, x_3)$ et du temps $t \in [0,T]$. Le vecteur S représente les termes sources du système. Nous utilisons la convention de sommation sur les indices répétés. Soit un vecteur $n = (n_1, n_2, n_3) \in \mathbb{R}^3$, nous définissons le flux du système (1) par

$$F(W,n) = F^i(W)n_i. \tag{2}$$

Il faut adjoindre des conditions aux limites à (1). Formellement, ces conditions aux limites sont données par un flux frontière F_b et s'écrivent

$$F(W,n) = F_b(W,n), \quad x \in \partial\Omega, \tag{3}$$

où n désigne le vecteur normal sortant à Ω sur $\partial\Omega$. D'autre part, nous nous donnons aussi une condition initiale

$$W(x,0) = W_0(x), \quad x \in \Omega.$$

SCHNAPS permet de résoudre ce problème d'évolution par la méthode de Galerkin Discontinu (GD). Comme son nom l'indique, cette méthode consiste à construire une approximation de la solution au moyen d'éléments finis discontinus.

1.2 Flux numérique

La méthode GD est une généralisation de la méthode des éléments finis et de la méthode des volumes finis. Elle nécessite la définition d'un flux numérique sur les discontinuités de la solution discrète. Ce flux numérique est noté

$$F(W_L, W_R, n_{LR}), \tag{4}$$

où W_L et W_R représentent les valeurs de W de part et d'autre de la discontinuité, et n_{LR} un vecteur normal à la discontinuité orienté du côté L vers le côté R. Cette notation, bien que légèrement abusive, ne peut pas être confondue avec celle du flux (2), car le flux numérique dépend de deux états, W_L et W_R, au lieu d'un seul. Pour que l'approximation GD soit consistante avec le système (1), il faut que

$$\forall W, n, \quad F(W, W, n) = F(W, n). \tag{5}$$

Souvent, le flux numérique vérifie aussi une propriété de conservation

$$\forall W_L, W_R, n, \quad F(W_L, W_R, n) = -F(W_R, W_L, -n), \tag{6}$$

mais ce n'est pas toujours le cas, et ce n'est pas une nécessité dans SCHNAPS.

À partir de la fonction flux, nous pouvons retrouver les composantes du flux. Nous introduisons le symbole de Kronecker

$$\delta_i^j = \begin{cases} 1 & \text{if} \quad i = j, \\ 0 & \text{if} \quad i \neq j. \end{cases}$$

Le vecteur δ^j est un vecteur unitaire de $\mathbb{R}^3$ qui pointe dans la direction x_j. Alors

$$F^j(W) = F(W, \delta^j). \tag{7}$$

Par exemple, les équations de Maxwell entrent dans ce formalisme. Elles s'écrivent

$$\partial_t E - \nabla \times H = -J, \tag{8}$$
$$\partial_t H + \nabla \times E = 0, \tag{9}$$

où E est le champ électrique, H le champ magnétique et J le vecteur courant électrique. Nous posons

$$W = (E^T, H^T)^T, \quad S = (-J^T, (0,0,0)^T)^T,$$

et

$$F(W,n) = \begin{bmatrix} 0 & n\times \\ -n\times & 0 \end{bmatrix} W = A(n)W,$$

de telle façon que les équations de Maxwell sont un cas particulier de (1), avec $m = 6$.

Comme dans (7) nous pouvons définir les matrices symétriques 6×6

$$A^i = A(\delta^i), \quad i = 1 \cdots 3.$$

Alors les équations de Maxwell peuvent aussi être écrites sous la forme d'un système hyperbolique, aussi appelé système de Friedrichs

$$\partial_t W + A^i \partial_i W = S.$$

1.3 Formalisme GD général

Pour définir l'approximation GD, nous devons d'abord construire un maillage de Ω. Nous considérons un maillage constitué d'un nombre fini d'ensembles ouverts $L_k \subset \Omega$, $k = 1 \cdots N$, appelés "cellules" ou "éléments", et satisfaisant les deux conditions :

1. $\forall k, l \quad k \neq l \Rightarrow L_k \cap L_l = \emptyset$.
2. $\overline{\cup_k L_k} = \overline{\Omega}$.

Soient L et R deux cellules voisines. La face commune à L et R est notée

$$L/R = \overline{L} \cap \overline{R}.$$

Nous notons aussi n_{LR} le vecteur normal unitaire sur L/R orienté L vers R. Donc, $n_{LR} = -n_{RL}$.

Dans chaque cellule L, nous construisons une base de fonctions scalaires φ_i^L, $i = 1 \cdots p_L$ dont le support est dans L. Il est possible d'avoir des niveaux d'approximation différents p_L sur des cellules différentes. Dans la cellule L, la solution de (1) est approchée par

$$W(X,t) = W_L(X,t) = W_L^j(t)\varphi_j^L(X) \quad X \in L. \tag{10}$$

La solution numérique satisfait le schéma d'approximation GD

$$\forall L, \forall i \quad \int_L \partial_t W \varphi_i^L - \int_L F(W, \nabla \varphi_i^L) + \int_{\partial L} F(W_L, W_R, n_{LR}) \varphi_i^L = \int_L S \varphi_i^L. \tag{11}$$

Pour plus de clarté, nous pouvons faire les remarques suivantes sur le schéma GD (11) :

1. La formulation (11) est obtenue formellement en multipliant (1) par une fonction de base φ_i^L et en intégrant par parties sur la cellule L.
2. Nous utilisons (de façon abusive) la même notation la solution exacte et la solution approchée. En général, à partir de maintenant W désignera l'approximation GD de la solution exacte.
3. Par R nous désignons une cellule générique qui touche la cellule L le long de sa frontière ∂L. Cette notation est justifiée par le fait que, à une rotation près, on peut toujours supposer que le vecteur normal n_{LR} est orienté de la cellule L à gauche (Left), vers la cellule R à droite (Right).
4. Comme W est discontinu sur le bord de la cellule, il n'est pas possible de définir $F(W, n_{LR})$ sur ∂L. Par conséquent, comme dans la méthode des volumes finis, nous devons introduire un flux numérique $F(W_L, W_R, n_{LR})$. Généralement, le flux numérique satisfait deux conditions
   (a) Consistance : $F(W, W, n) = F(W, n)$.
   (b) Conservation : $F(W_L, W_R, n_{LR}) = -F(W_R, W_L, n_{RL})$.
5. Dans le cas des équations de Maxwell, nous utilisons le flux décentré standard

$$F(W_L, W_R, n) = A(n)^+ W_L + A(n)^- W_R.$$

6. Finalement, dans (11) nous devons être plus précis lorsque la cellule L touche le bord du domaine de calcul. En effet, sur $\partial L \cap \Omega$ le champ W_R n'est pas disponible. Nous remplaçons donc sur ces interfaces le flux numérique $F(W_L, W_R, n_{LR})$ par un flux frontière

$$F_b(W_L, n_{LR}).$$

Nous pouvons alors introduire le développement (10) dans (11). L'approximation GD devient alors un système d'équations différentielles ordinaires satisfaites par les coefficients $W_L^j(t)$ sur les bases locales. Nous résolvons ce système d'équations différentielles par un intégrateur Runge-Kutta du second ordre.

2 SCHNAPS : philosophie

Dans cette section nous tentons de donner une vue d'ensemble synthétique de la conception du logiciel. Pour une vue détaillée de la hiérarchie des classes nous renvoyons à la documentation doxygen de SCHNAPS (voir section 3.5).

2.1 OpenCL

SCHNAPS est une bibliothèque, écrite en C, qui permet de résoudre numériquement un système de lois de conservation générique par la méthode GD sur un calculateur disposant d'un ou plusieurs accélérateurs supportant l'environnement OpenCL. OpenCL permet d'écrire et d'exécuter en parallèle des programmes (appelés *kernels*) sur tous les processeurs d'un accélérateur. Lorsque plusieurs accélérateurs sont disponibles, SCHNAPS utilise l'environnement MPI pour les communications entre les accélérateurs. Cette approche très générale permet d'utiliser SCHNAPS sur une grande variété de calculateurs. SCHNAPS pourra aussi bien tourner sur un PC disposant d'un seul CPU classique que sur un supercalculateur avec des centaines de GPU.

La classe `CLInfo` contient des informations et des méthodes pour initialiser l'environnement OpenCL.

Une particularité d'OpenCL est qu'une partie du code est compilée à l'exécution. Cette particularité s'explique par la nécessité d'être compatible avec des accélérateurs d'architectures matérielles différentes. SCHNAPS intègre donc une notion de *fonction* enrichie (`Function`). Les fonctions SCHNAPS se comportent comme des fonctions C traditionnelles, mais contiennent également leur propre header et code source. Elles peuvent être appelées depuis le programme hôte ou depuis un kernel OpenCL.

Par ailleurs, pour calculer sur des accélérateurs de type GPU, il est nécessaire de dupliquer des données entre le CPU et le GPU. Pour cela, nous utilisons une classe de *buffers OpenCL* (`CLBuffer`). Cette classe permet d'allouer un vecteur sur le CPU et le GPU. Elle s'occupe aussi des synchronisations entre le CPU et le GPU.

2.2 Modèles physiques

Dans SCHNAPS, nous utilisons une notion de *modèle physique* générique. Un modèle dans SCHNAPS (`Model`) contient les informations suivantes :

- dimension m du vecteur W des variables conservatives du système (1),
- flux numérique,
- condition initiale,
- flux ou conditions aux limites,
- termes sources,
- et éventuellement : solution de référence (si elle existe), caractéristiques des matériaux, *etc.*

Toutes ces informations sont données dans des fonctions SCHNAPS, car elles doivent pouvoir être appelées depuis les kernels OpenCL.

2.3 Sous-domaines

Afin d'atteindre de bonnes performances en calcul parallèle, SCHNAPS repose sur plusieurs niveaux de parallélisme. Le premier niveau est un parallélisme à gros grain, géré par la bibliothèque MPI. Un autre niveau de parallélisme à grain plus fin est géré à partir de l'environnement OpenCL, qui permet d'exploiter la puissance de calcul des processeurs multicoeurs ou d'accélérateurs, comme les GPUs.

Les maillages de SCHNAPS sont organisés pour suivre cette hiérarchie. D'abord, le domaine Ω est découpé en *sous-domaines*. Chaque sous-domaine est associé à un noeud MPI. Un noeud MPI ne connaît que son sous-domaine.

S'il n'y a qu'un seul sous-domaine, SCHNAPS peut fonctionner sans la bibliothèque MPI.

2.4 Zones

Par ailleurs, chaque sous-domaine est lui-même découpé en *zones*. Les zones sont constituées d'éléments partageant les mêmes caractéristiques (modélisation physique, interpolation, forme, *etc.*) Ce regroupement assure une meilleure parallélisation, car les calculs des éléments d'une même zone sont similaires. Il existe des *zones volumiques* pour les calculs dans le volume. Il existe aussi des *zones d'interface* (ou *zones surfaciques*) constituées d'éléments surfaciques pour la modélisation de termes d'interface.

Les échanges de données entre les zones volumiques sont assurés par les zones surfaciques. Les zones surfaciques au bord des sous-domaines sont également utilisées pour les communications MPI.

Nous réalisons les calculs à l'intérieur des zones à partir de kernels OpenCL. Ces kernels calculent les flux, les termes sources dans les zones volumiques. Les kernels calculent également les flux provenant des zones surfaciques. Ils réalisent enfin les extractions de données des zones volumiques vers les zones d'interfaces.

Cette notion de zones existe dans plusieurs classes :

— Le *maillage de zone* (`ZoneMesh`) permet de décrire le maillage géométrique, éventuellement déstructuré, mais conforme, d'une zone. Cette classe permet de décrire aussi bien un maillage volumique que surfacique. Pour créer un maillage non conforme, il est nécessaire d'utiliser plusieurs maillages volumiques reliés par un maillage surfacique. Un `ZoneMesh` contient une structure classique de maillage : noeuds géométriques, éléments, tableaux de connectivité éléments vers noeuds et éléments vers éléments. Si le `ZoneMesh` est surfacique, il contient en plus deux pointeurs vers les `ZoneMesh` volumiques voisins de gauche et de droite. Dans ce cas, la connectivité éléments vers éléments permet de retrouver les voisins volumiques d'un élément surfacique.

— Le *champ de zone* (`ZoneField`) permet de décrire un champ complet de variables conservatives sur un `ZoneMesh`. Le `ZoneField` pointe vers un `Model`. Il contient également une fonction SCHNAPS très importante pour les performances : la fonction `varindex`. Cette fonction permet de retrouver en mémoire une donnée conservative, connaissant son indice, son élément et son numéro de point Gauss. Le rangement effectif en mémoire de la donnée est réalisé dans cette fonction. Il est important de faire attention à ce rangement afin de maximiser la localité des accès mémoires. La réécriture de cette fonction permet de changer l'organisation des données en mémoire sans changer la plupart des algorithmes GD.

— Les classes `ZoneSimulation` et `ZoneInterface` contiennent un maillage et des champs de zone. Elles contiennent également les algorithmes nécessaires à l'implémentation de la méthode GD. La classe `ZoneSimulation` est plus particulièrement dédiée aux algorithmes sur les zones volumiques, tandis que la classe `ZoneInterface` s'occupe plutôt des tâches surfaciques. Par exemple nous trouvons dans `ZoneSimulation` l'algorithme qui permet d'assembler les flux des faces internes d'une zone. Les flux d'interface sont calculés par `ZoneInterface`. Ce découpage assure un recouvrement des communications par les calculs : le calcul des faces internes peut en effet commencer même si

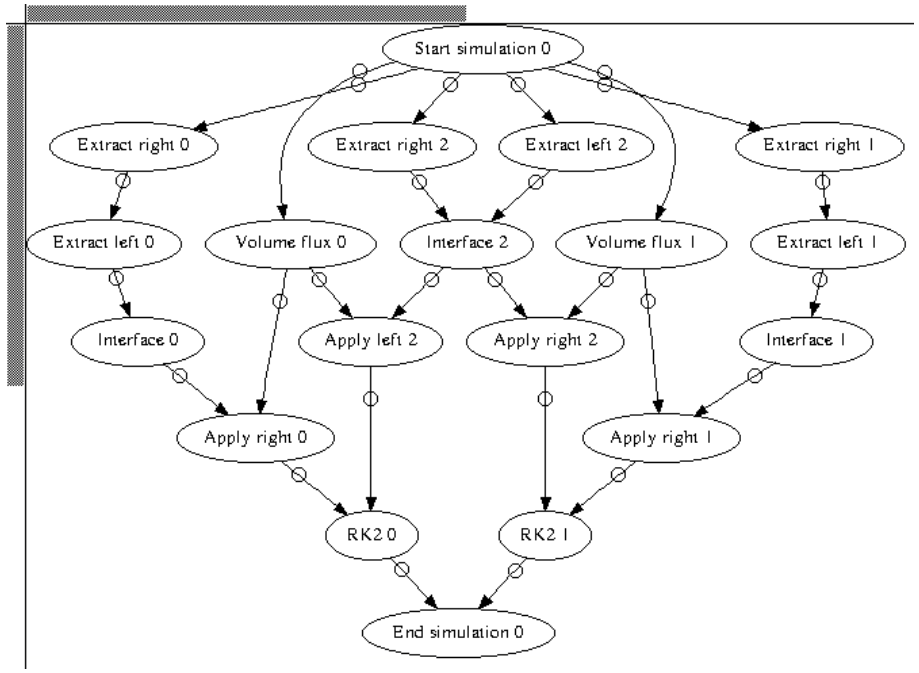

FIGURE 1 – Graphe des tâches pour un calcul dans un sous-domaine contenant 3 zones. Les tâches sont représentées par les noeuds du graphe et les flèches représentent les dépendances entre les tâches (une tâche doit attendre la fin des tâches dont elle dépend avant de démarrer).

les communications des interfaces ne sont pas terminées.

2.5 Simulation

Une *simulation* (`Simulation`) contient toutes les zones d'un sous-domaine. Cette classe est conçue pour contenir les algorithmes à appliquer sur les zones. En particulier, c'est cette classe qui construit un graphe des tâches pour lancer les calculs OpenCL et les communications MPI de façon asynchrone. Un exemple de graphe des tâches pour un calcul sur un sous-domaine avec trois zones est représenté Figure 1. Le graphe est exécuté de façon totalement asynchrone. SCHNAPS utilise le mécanisme des événements OpenCL pour gérer les dépendances entre les tâches. Les communications MPI sont lancées dans des threads indépendants afin de ne pas bloquer le déroulement du calcul OpenCL.

2.6 Éléments géométriques

Dans cette section, nous nous plaçons au niveau d'un élément L. Suivant la zone dans laquelle il se trouve, cet élément peut avoir des caractéristiques géométriques

différentes. Nous utilisons le formalisme classique des éléments finis. À chaque zone nous attribuons un élément de référence $\hat{L}$. Cet élément s'appuie sur des *noeuds géométriques*, notés $\hat{X}_i$ et des *fonctions de base géométriques*, notées $\hat{\psi}_i$ telles que

$$\hat{\psi}_i(\hat{X}_j) = \delta_{ij}.$$

Dans SCHNAPS, dans une zone donnée, le nombre de noeuds et de fonctions géométriques est noté `nb_nodes`. Pour un hexaèdre à huit noeuds (élément de type H8), `nb_nodes`=8. Les noeuds sont donnés dans le tableau `ref_node`. L'élément de référence possède également un certain nombre de faces, noté `nb_faces`. Le tableau `face2node` permet de retrouver les noeuds d'une face donnée. Le numéro du j-ième noeud de la face jf est donné par `face2node[nb_face_nodes*jf+j]`. L'entier `nb_face_nodes` est le nombre maximal de noeuds par face (par exemple `nb_face_nodes`=4 pour un hexaèdre à 8 noeuds).

Nous utiliserons ce formalisme pour toutes les familles d'éléments finis de SCHNAPS, qu'il s'agisse d'hexaèdres, de quadrangles, de triangles, de tétraèdres, d'éléments courbes. Les éléments de type surfacique (qui se trouvent dans les zones d'interface) sont décrits avec ce formalisme. Pour ces éléments, nous utilisons de plus un paramétrage volumique au voisinage de la surface dans la direction du vecteur normal. Par exemple, le quadrangle à 4 noeuds (élément de type Q4) est "épaissi" dans la direction de son vecteur normal en un élément de type H8. Les quatre noeuds supplémentaires sont déduits des 4 noeuds initiaux.

À titre d'exemple, nous montrons maintenant précisément comment nous pouvons définir l'interpolation pour des hexaèdres à 8 noeuds (éléments de type H8 dans la terminologie classique des éléments finis). Dans ce cas, l'élément de référence $\hat{L}$ est le cube unité

$$\hat{L} = [0,1]^3.$$

Soient $\hat{X} = (\hat{x}, \hat{y}, \hat{z})$ les coordonnées dans l'élément de référence. Les noeuds de référence $\hat{X}^i$, $i = 1 \cdots 8$ et les fonctions géométriques $\hat{\psi}_i$ de l'élément de référence

sont données par

i	$\hat{X}^i$	$\hat{\psi}_i$
1	$(0,0,0)$	$(1-\hat{x})(1-\hat{y})(1-\hat{z})$
2	$(1,0,0)$	$\hat{x}(1-\hat{y})(1-\hat{z})$
3	$(1,1,0)$	$\hat{x}\hat{y}(1-\hat{z})$
4	$(0,1,0)$	$(1-\hat{x})\hat{y}(1-\hat{z})$
5	$(0,0,1)$	$(1-\hat{x})(1-\hat{y})\hat{z}$
6	$(1,0,1)$	$\hat{x}(1-\hat{y})\hat{z}$
7	$(1,1,1)$	$\hat{x}\hat{y}\hat{z}$
8	$(0,1,1)$	$(1-\hat{x})\hat{y}\hat{z}$

Un hexaèdre H8 arbitraire est alors défini par huit noeuds X_L^i. La transformation géométrique qui envoie $\hat{L}$ sur L est définie par

$$\tau_L(\hat{X}) = \hat{\psi}_i(\hat{X}) X_L^i.$$

Nous faisons l'hypothèse que les noeuds X_L^i sont choisis de telle sorte que τ_L est une transformation directe et inversible. En pratique, il faut s'assurer que le choix des noeuds ne conduit pas à des éléments mal orientés ou trop déformés. Comme les fonctions de base géométriques satisfont

$$\hat{\psi}_i(\hat{X}^j) = \delta_{ij}$$

nous déduisons que la transformation géométrique envoie les noeuds de référence sur les noeuds de l'élément L

$$\tau_L(\hat{X}^i) = X_L^i.$$

Pour la numérotation des faces, nous utilisons la convention suivante. Tout d'abord, pour le cube de référence, les faces sont numérotées de 0 à 5 selon le modèle de la figure 2 :

Nous définissons ensuite un tableau de permutation des axes des faces

$$\text{axis\_permut} = \begin{pmatrix} 0 & 2 & 1 & 0 \\ 1 & 2 & 0 & 1 \\ 2 & 0 & 1 & 1 \\ 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 \\ 1 & 0 & 2 & 0 \end{pmatrix}.$$

Chaque ligne correspond à une face du cube. Les trois premières colonnes code une

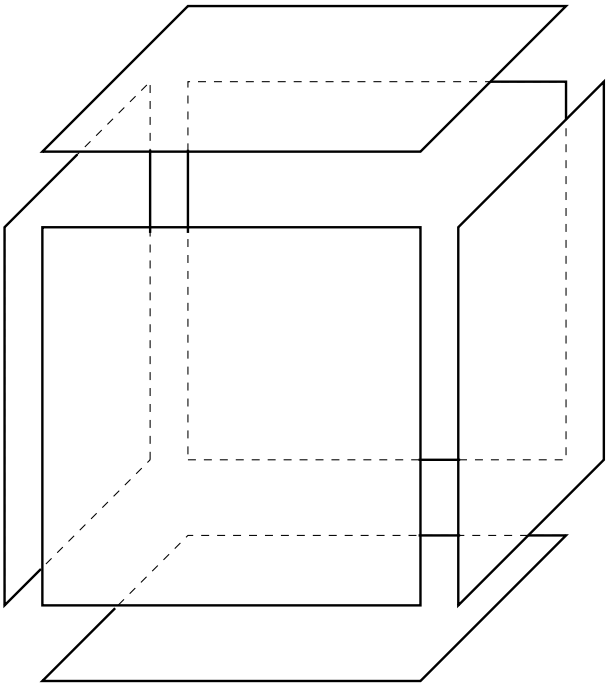

FIGURE 2 – Cube de référence.

permutation des axes selon la convention "0" pour x, "1" pour y et "2" pour z, les deux premiers axes étant dans le plan de la face et le troisième orienté suivant la normale sortante. La dernière colonne donne la valeur de la troisième coordonnée qui est constante sur la face. Par exemple, la troisième ligne correspond à la face 2 qui est dans le plan ("2","0") $= (z, x)$. La direction normale à la face est la direction "1" $= y$. Enfin sur cette face, on a bien $y = 1$.

Pour un numéro de face `ifa`, nous notons opposite_face(`ifa`) le numéro de face opposée :

$$\text{opposite\_face} = \begin{pmatrix} 2 \\ 3 \\ 0 \\ 1 \\ 5 \\ 4 \end{pmatrix}.$$

La géométrie des éléments est décrite dans la classe `ElementGeometry`. Cette classe contient des fonctions SCHNAPS (qui pourront donc être appelées depuis les kernel OpenCL) pour calculer la transformation géométrique τ en un point de référence, son gradient, son jacobien, son inverse, le vecteur normal dans le cas d'un élément surfacique, *etc.*

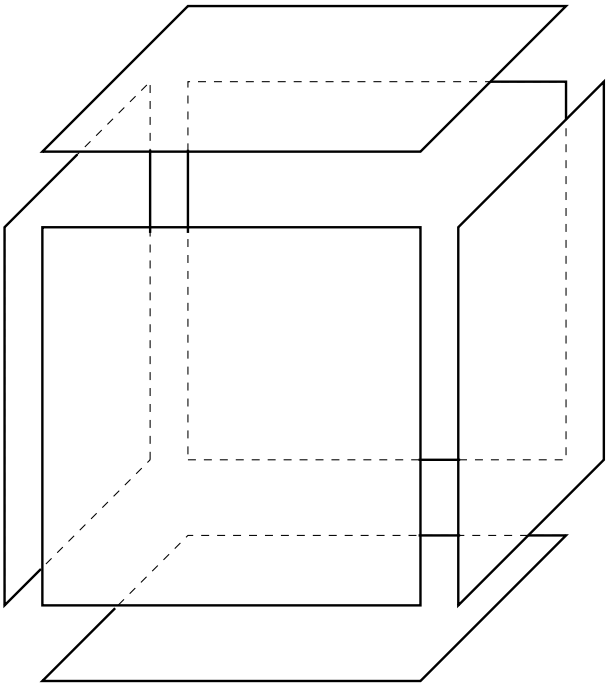

FIGURE 3 – Cube de référence.

2.7 Interpolation

Une fois définie la géométrie d'un élément, on peut définir l'interpolation d'un champ scalaire. Cette information est décrite dans la classe `ElementInterpolation`.

Pour l'instant dans SCHNAPS, pour des raisons d'efficacité, nous utilisons une approximation nodale adaptée à la quadrature numérique (voir [1, 2]) : les champs sont définis aux points d'intégration de Gauss des éléments. Ce choix permet d'accéder directement aux champs pour l'intégration numérique dans le volume. Ce choix assure aussi que les matrices masses locales sont diagonales.

Nous décrivons par exemple comment nous interpolons les champs sur un élément de type H8 avec des polynômes de degré d.

Pour l'interpolation nous fixons d'abord dans la cellule L un degré d. Nous considérons les $(d+1)$ zéros $(\xi_i)_{i=0\cdots d}$ du $(d+1)^{\text{ième}}$ polynôme de Legendre sur $[0,1]$, e t les poids d'intégration correspondants ω_i. Nous notons aussi I_k le $k^{\text{ième}}$ polynôme de Lagrange associé aux points ξ_i. Rappelons que I_k est un polynôme de degré d et que

$$I_j(\xi_i) = \delta_{ij}.$$

Nous construisons alors, sur l'élément de référence $\hat{L}$, les points de Gauss $\hat{Y}_q$, les poids $\hat{\lambda}_q$ et les fonctions d'interpolation $\hat{\varphi}^q(\hat{X})$ à partir de produits tensoriels de quantités monodimensionnelles. Plus précisément, soient i, j et k trois entiers dans

$\{0 \cdots d\}$ et soit $q = (d+1)^2 k + (d+1)j + i$ alors

$$\hat{Y}_q = (\xi_i, \xi_j, \xi_k), \quad \hat{\lambda}_q = \omega_i \omega_j \omega_k, \quad \hat{\varphi}^q(\hat{X}) = I_i(\hat{x}) I_j(\hat{y}) I_k(\hat{z}).$$

Finalement, nous obtenons les fonctions de base sur l'élément L en transportant les fonctions d'interpolation de référence avec la transformation géométrique.

$$\varphi_L^i(X) = \hat{\varphi}^i(\hat{X}) \text{ with } X = \tau_L(\hat{X}).$$

La classe `ElementInterpolation` contient des fonctions permettant de calculer : la position des points de Gauss du volume ou des faces, les poids de Gauss, les valeurs des fonctions de base et de leurs gradients.

2.8 Interpolation de Gauss-Lobatto avec des sous-cellules

Pour l'interpolation nous fixons d'abord dans la cellule L un degré $d(\ell)$ pour chaque direction $\ell = 0, 1, 2$. Dans chaque direction, nous considérons les $(d(\ell)+1)$ points de Gauss-Lobatto $(\xi_i)_{i=0\cdots d(\ell)}$ sur $[0,1]$, et les poids d'intégration correspondants ω_i. Nous considérons de plus un raffinement $\text{nraf}(\ell)$ pour $\ell = 0, 1, 2$. Nous notons aussi I_k le $k^{\text{ième}}$ polynôme de Lagrange associé aux points ξ_i. Rappelons que I_k est un polynôme de degré d et que

$$I_j(\xi_i) = \delta_{ij}.$$

Sur l'élément de référence $\hat{L}$, nous construisons alors les points de Gauss-Lobatto $\hat{Y}_q$, les poids $\hat{\lambda}_q$ et les fonctions d'interpolation $\hat{\varphi}^q(\hat{X})$ à partir de produits tensoriels de quantités monodimensionnelles. Plus précisément, soient $i(\ell)_{\ell=0,1,2}$ trois entiers dans $\{0 \cdots d(0)\} \times \{0 \cdots d(1)\} \times \{0 \cdots d(2)\}$ et $r(\ell)_{\ell=0,1,2}$ trois entiers dans $\{0 \cdots \text{nraf}(0)\} \times \{0 \cdots \text{nraf}(1)\} \times \{0 \cdots \text{nraf}(2)\}$ et soit

$$\begin{aligned} q &= i(0) \\ &+ i(1)(d(0)+1) \\ &+ i(2)(d(0)+1)(d(1)+1) \\ &+ r(0)(d(0)+1)(d(1)+1)(d(2)+1) \\ &+ r(1)(d(0)+1)(d(1)+1)(d(2)+1)\text{nraf}(0) \\ &+ r(2)(d(0)+1)(d(1)+1)(d(2)+1)\text{nraf}(0)\text{nraf}(1), \end{aligned}$$

alors

$$\begin{aligned}
\hat{Y}_q &= (h(0)(r(0)+\xi_{i(0)}), h(1)(r(1)+\xi_{i(1)}), h(2)(r(2)+\xi_{i(2)})),\\
\hat{\lambda}_q &= h(0)\omega_{i(0)}h(1)\omega_{i(1)}h(2)\omega_{i(2)},\\
\hat{\varphi}^q(\hat{X}) &= I_{i(0)}(\frac{\hat{x}}{h(0)} - r(0))I_{i(1)}(\frac{\hat{y}}{h(1)} - r(1))I_{i(2)}(\frac{\hat{z}}{h(2)} - r(2))\,\chi^r(\hat{X}),
\end{aligned}$$

où $h(\ell) = 1/\mathrm{nraf}(\ell)$ et

$$\chi^r(\hat{X}) = \mathbb{1}_{[h(0)r(0),h(0)(r(0)+1)]}(\hat{x})\mathbb{1}_{[h(1)r(1),h(1)(r(1)+1)]}(\hat{y})\mathbb{1}_{[h(2)r(2),h(2)(r(2)+1)]}(\hat{z}).$$

En pratique, on calcule q grâce à la formule de Hörner

$$q = i(0)+(d(0)+1)(i(1)+(d(1)+1)(i(2)+(d(2)+1)(r(0)+\mathrm{nraf}(0)(r(1)+\mathrm{nraf}(1)r(2))))).$$

Dans la suite nous utiliserons la notation q aussi pour le multi-indice $q = (i(0), i(1), i(2), r(0), r(1), r(2)$
(i, r).

Finalement, nous obtenons les fonctions de base sur l'élément L en transportant les fonctions d'interpolation de référence avec la transformation géométrique.

$$\varphi^i_L(X) = \hat{\varphi}^i(\hat{X}) \text{ with } X = \tau_L(\hat{X}).$$

Nous définissons à présent l'indexation des points de Gauss-Lobatto sur les faces. Ces points sont repérés par un numéro de face $\texttt{ifa} \in \{0 \cdots 5\}$, deux indices de points $(i'(0), i'(1))$ et deux indices de sous-face $(r'(0), r'(1))$. Nous utilisons le tableau axis_permut introduit à la section 2.6 pour retrouver la position du point de Gauss-Lobatto dans le volume à l'aide des formules suivantes

$$\left\{\begin{array}{rcl}
i(\mathrm{axis_permut}(\texttt{ifa}, 0)) &=& i'(0),\\
i(\mathrm{axis_permut}(\texttt{ifa}, 1)) &=& i'(1),\\
i(\mathrm{axis_permut}(\texttt{ifa}, 2)) &=& \mathrm{axis_permut}(\texttt{ifa}, 3)\, d(\mathrm{axis_permut}(\texttt{ifa}, 3)).
\end{array}\right.$$

Les indices de la sous-cellule sont donnés par

$$\left\{\begin{array}{rcl}
r(\mathrm{axis_permut}(\texttt{ifa}, 0)) &=& r'(0),\\
r(\mathrm{axis_permut}(\texttt{ifa}, 1)) &=& r'(1),\\
r(\mathrm{axis_permut}(\texttt{ifa}, 2)) &=& \mathrm{axis_permut}(\texttt{ifa}, 3)\,(\mathrm{nraf}(\mathrm{axis_permut}(\texttt{ifa}, 3)) - 1).
\end{array}\right.$$

Les degrés d'approximation permutés sont donnés par $d'(\mathrm{axis\_permut}(\texttt{ifa}, \ell)) =$

$d(\ell)$ et les raffinements permutés sont donnés par $\mathrm{nraf}'(\mathrm{axis\_permut}(\mathtt{ifa}, \ell)) = \mathrm{nraf}(\ell)$.

Dans la suite, il est nécessaire de pouvoir passer de l'indexation des points de Gauss-Lobatto sur les faces à leur indexation dans les volumes. Avec la convention choisie, le numéro $q' = (i', r')$ du point de Gauss-Lobatto sur la face est donné par

$$\begin{aligned} q' &= i'(0) \\ &+ i'(1)(d'(0)+1) \\ &+ r'(0)(d'(0)+1)(d'(1)+1) \\ &+ r'(1)(d'(0)+1)(d'(1)+1)\mathrm{nraf}'(0). \end{aligned}$$

Nous notons Π la transformation qui fait passer de la numérotation q' sur le bord ∂L à la numérotation q à l'intérieur de la cellule

$$q = \Pi(\mathtt{ifa}, q').$$

Enfin, nous avons aussi besoin d'une numérotation des points de Gauss des faces des sous-cellules. Soit un numéro de sous-cellule r, un numéro de face `ifa` et un numéro de point de Gauss de sous-cellule i'. L'indice correspondant à q dans la cellule L est donné par la transformation

$$q = \pi(r, \mathtt{ifa}, i').$$

Dans le détail, on a $q = (i(0), i(1), i(2), r(0), r(1), r(2))$ avec

$$\left\{ \begin{array}{lcl} i(\mathrm{axis\_permut}(\mathtt{ifa}, 0)) &=& i'(0), \\ i(\mathrm{axis\_permut}(\mathtt{ifa}, 1)) &=& i'(1), \\ i(\mathrm{axis\_permut}(\mathtt{ifa}, 2)) &=& \mathrm{axis\_permut}(\mathtt{ifa}, 3)\, d(\mathrm{axis\_permut}(\mathtt{ifa}, 3)). \end{array} \right.$$

2.9 Schéma GD pour l'interpolation de Gauss-Lobatto sur des sous-cellules

Dans le cas de l'interpolation de Gauss-Lobatto sur des sous cellules, nous pouvons exprimer l'algorithme de calcul GD sous une forme plus efficace, direction par direction. Nous commençons par remplacer les intégrales de volumes et de bord par leurs quadratures de Gauss-Lobatto. Pour une cellule L et une composante $q = (i, r)$ la

quadrature Gauss-Lobatto s'écrit

$$\omega_q^L \frac{d}{dt} W_L^q - \mathcal{V} + \mathcal{D} + \mathcal{B} = \omega_q^L S(Y_L^q),$$

où $\mathcal{V}$, $\mathcal{D}$ et $\mathcal{B}$ désignent respectivement les termes volumiques, les termes aux interfaces des sous-cellules et les termes de bord de la cellule L. Pour l'integrale de volume, il faut considérer les contributions sur tous les points de Gauss-Lobatto Y_L^p avec $p = (j(0), j(1), j(2), s(0), s(1), s(2)) = (j, s)$, soit

$$\mathcal{V} = \sum_p \omega_p^L F(W_L^p) \cdot \nabla \varphi_q^L(Y_L^p).$$

D'autre part, si q correspond à un point de Gauss interne à la cellule L on a

$$\mathcal{B} = 0.$$

Si q correspond à un point du bord ∂L, il existe au moins un numéro de face ifa et un indice de bord q' tel que

$$q = \Pi(\text{ifa}, q')$$

et

$$\mathcal{B} = \sum_{q=\Pi(\text{ifa},q')} \omega_{q'}^{\partial L} F(W_L^q, W_R(Y_L^q), n_{LR}(Y_L^q)).$$

Il reste à expliciter les termes de saut aux interfaces des sous-cellules.

Si q correspond à un point de Gauss interne à une sous-cellule alors

$$\mathcal{D} = 0.$$

Si q correspond à un point de bord de la sous-cellule r alors il existe une ou plusieurs sous-faces `ifa` et un numéro i' de point de Gauss dans la sous-face tels que

$$q = \pi(r, \texttt{ifa}, i').$$

Le point q admet un unique vis-à-vis $\tilde{q}$ sur la face `ifa` dans une sous-cellule voisine $\tilde{r}$

$$\tilde{q} = \pi(\tilde{r}, \text{opposite\_face}(\texttt{ifa}), \tilde{i'}),$$

et les points de Gauss-Lobatto correspondants sont confondus

$$Y_L^{\tilde{q}} = Y_L^q.$$

Avec ces notations, les flux d'interfaces internes s'écrivent

$$\mathcal{D} = \sum_{q=\pi(r,\text{ifa},i')} \omega_{i'}^{\partial r} F(W_L^q, W_L^{\tilde{q}}, n_{r\tilde{r}}(Y_L^q)),$$

où $\omega_{i'}^{\partial r}$ désigne les poids de Gauss des point i' sur le bord de la sous-cellule r. La normale à l'interface entre les sous-cellules r et $\tilde{r}$, orientée de r vers $\tilde{r}$ est notée $n_{r\tilde{r}}$.

2.10 Algorithme GD avec optimisation des accès mémoire sur une macro-cellule

to do

2.11 Autres classes

Pour des raisons pratiques, nous avons dû développer d'autres utilitaires :

- La classe `Mesher` permet de réaliser rapidement des maillages constitués de blocs structurés curvilignes. Ce mailleur, indépendant de SCHNAPS, permet d'alimenter rapidement des tests unitaires ou de réaliser des calculs sur des géométries simples.
- La classe `GmshConverter` contient des outils pour convertir des maillages issus de gmsh ou de `Mesher` dans le format SCHNAPS.

3 Installation

3.1 Prérequis

L'installation de SCHNAPS repose sur :

- Système Linux ou Mac récent. Windows est prévu, mais pas encore opérationnel.
- Un compilateur C récent supportant la norme C99. Testé avec gcc et clang.
- git pour le gestionnaire de version.
- scons comme système de compilation.
- doxygen pour construire la documentation.
- au moins un pilote OpenCL. Testé avec les pilotes AMD (CPU ou GPU), Intel (CPU ou GPU), NVIDIA.

— boost. Nous utilisons boost-graph pour implémenter le graphe des tâches et boost-test pour les tests unitaires.
— MPI pour le parallélisme multi-gpu. MPI est facultatif et peut-être désactivé.
— gmsh pour la visualisation de certains résultats et pour construire les maillages de certains tests unitaires.

SCHNAPS utilise la bibliothèque LGPL ANN `http://www.cs.umd.edu/~mount/ANN/`. Les sources de cette bibliothèque sont incluses dans la distribution.

3.2 Téléchargement

SCHNAPS est hébergé sur github.

Pour pouvoir modifer SCHNAPS, il faut d'abord ouvrir un compte sur github et demander à être membre du projet SCHNAPS.

Pour télécharger la dernière version

```
git clone https://github.com/phelluy/SCHNAPS.git
```

Il est aussi possible de télécharger une archive zip à partir de l'interface web de github.

3.3 Compilation

Se placer dans le dossier SCHNAPS :

```
cd SCHNAPS
```

Il faut choisir où placer l'exécutable. On peut par exemple créer un dossier bin dans le dossier SCHNAPS :

```
mkdir bin
```

Il faut ensuite créer une variable d'environnement pour que scons retrouve ce dossier :

```
export SCHNAPS_INSTALL_DIR=$(PWD)/bin
```

Pour la compilation, il suffit de taper :

```
scons
```

Pour compiler la version MPI

```
scons --mpi
```

3.4 Tests

Pour l'instant, SCHNAPS est en cours de développement. Nous avons implémenté des tests unitaires ainsi que quelques tests de calculs GD pour des modèles et des géométries simples.

Pour lancer tous les tests en mode debug simple précision :

```
./bin/SCHNAPS_utest_dbg
```

Pour lancer un test particulier :

```
./bin/SCHNAPS_utest_dbg --run-test=NOM_DU_TEST
```

Par exemple :

```
./bin/SCHNAPS_utest_dbg --run_test=test_transport --report_level=detailed
```

lance un test de résolution de l'équation de transport sur un cube. La solution exacte est appliquée sur les frontières du cube. Ce test génère plusieurs fichiers de visualisation. Pour voir la solution à l'instant final, on peut par exemple faire :

```
gmsh test_transport_fin.msh
```

Autre exemple :

```
./bin/SCHNAPS_utest_dbg --run_test=test_interface_raf_nonraf \
--report_level=detailed
```

lance un test de résolution de l'équation de transport sur un maillage constitué de 3 zones. Une des zones est maillée plus grossièrement que les deux autres. Les raccords entre les trois zones sont donc conformes ou non conformes. De plus, deux des zones sont constituées de cellules courbes.

Affichage du résultat à l'instant final :

```
gmsh end_test_interface_raf_0.msh -open \
end_test_interface_raf_1.msh -open end_test_interface_raf_2.msh
```

(il faut ajuster les échelles dans les menus gmsh pour que le post-traitement des trois zones soit uniforme).

Exemple utilisant MPI :

mpirun -np 2 ./bin/SCHNAPS_utest_mpi_dbg –run_test=test_simulation_mpi

Ce test exécute le même calcul que précédemment, mais sur deux devices OpenCL. Si l'ordinateur sur lequel ce test est lancé dispose d'un seul GPU, une partie des calculs est réalisés sur le CPU (qui est un device OpenCL comme un autre...)

3.5 Documentation détaillée

Pour construire la documentation, se placer dans le dossier doc :

```
cd doc
```

Lancer la création de la documentation avec doxygen

```
doxygen doxySCHNAPS
```

Visualiser la documentation :

```
firefox ./html/index.html
```

4 Travaux en cours

Nous avons prévu les étapes suivantes dans le développement de SCHNAPS :

1. Enrichissement des modèles physiques : Maxwell, MHD, *etc.*
2. Optimisation des kernels.
3. Incorporation de divers outils : fentes, plaques, sorties sur fichiers, *etc.*
4. Prise en compte des formats de fichiers AxesSim (amelet).
5. éléments d'ordre élevé (pour la prise en compte des géométrie courbes).
6. Interpolation de Gauss-Lobatto.
7. Autres.

Nous sommes arrivés à un développement de SCHNAPS qui fait que les tâches 1 à 7 peuvent être réalisées dans n'importe quel ordre et en parallèle par les contributeurs, en fonction de leurs priorités...

References

[1] G. Cohen, X. Ferrières and S. Pernet. A spatial high order hexahedral discontinuous Galerkin method to solve Maxwell's equations in time-domain. J. Comput. Phys., vol. 217, no. 2, pages 340–363, 2006.

[2] A. Klockner, T. Warburton, J. Bridge, J.S. Hesthaven, Nodal discontinuous Galerkin methods on graphics processors, Journal of Computational Physics, Volume 228, Issue 21, 20 November 2009, Pages 7863-7882