# SCHNAPS : Solveur Conservatif Hyperbolique Non-linéaire Appliqué aux PlaSmas

IRMA Strasbourg

10 mars 2016

## Table des matières

# 1 Introduction

Dans ce chapitre, nous rappelons d'abord rapidement le formalisme de la méthode Galerkin Discontinu (GD). Cette méthode est très générale puisqu'elle autorise des maillages non-conformes ainsi que des degrés d'approximation non-uniforme. Cependant sa programmation dans un cadre trop général risque d'être inefficace sur un ordinateur hybride possédant plusieurs niveaux de mémoires et de parallélisme : les indirections mémoire, les trop grandes variations de paramètres, un rangement arbitraire des données en mémoire peuvent conduire à des performances désastreuses. Nous allons donc présenter ici les choix que nous avons fait afin d'obtenir un bon compromis entre la généralité de la méthode et son efficacité. Le résultat de ces compromis est le logiciel SCHNAPS développé depuis deux à l'IRMA Strasbourg.

SCHNAPS est un acronyme de "Solveur Conservatif Hyperbolique Non-linéaire Appliqué aux PlasmaS". Il s'agit d'une bibliothèque C99 pour résoudre numériquement des systèmes de lois de conservation sur des ordinateurs massivement parallèles, avec plusieurs niveaux de parallélisme, par exemple un gros cluster comprenant à la fois des CPU et des GPU.

# 2 Méthode Galerkin Discontinu (GD)

## 2.1 Systèmes de lois de conservation

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 $x \in \Omega$, $transfertsx = (x_1, x_2, x_3)$ 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 $\Omega$ 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.

## 2.2 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 conservationtransferts

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

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 $\delta^j$ 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 $6 \times 6$

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

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

hyperbolique, aussi appelé système de Friedrichs

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

## 2.3 Formalisme GD général

Pour définir l'approximation GD, nous devons d'abord construire un maillage de $\Omega$. 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 $\varphi_i^L$, $i = 1 \cdots p_L$ 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 $\varphi_i^L$ 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 $\partial L$. 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 $\partial L$. Par conséquent, comme dans la méthode des volumes finis, nous devons utiliser un flux numérique $F(W_L, W_R, n_{LR})$, introduit dans la section précédente. Le flux numérique satisfait souvent les 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 le 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.

# 3 SCHNAPS : philosophie

Dans cette section nous tentons de donner une vue d'ensemble synthétique de la conception du logiciel.

## 3.1 StarPU

Afin d'exploiter les architectures d'ordinateurs hybrides les plus récentes, il est indispensable de pouvoir soumettre de façon asynchrone des opérations de calcul ou de transferts (copies mémoire ou communications). Ces opérations, appelées tâches ("tasks" en anglais), dépendent en général les unes des autres. Ce sont les dépendances entre les tâches qui permettent de déterminer l'ordre dans lequel elle doivent être exécutées et si certaines d'entre elles peuvent être réalisées en parallèle.

Les dépendances entre tâches sont souvent représentée par un graphe. Les noeuds du graphe correspondent aux tâches. La présence d'un arc orienté de la tâche A vers la tâches B indique qu'il faut attendre la fin de A avant de lancer B. Voir Figure 1. En général, la construction d'un graphe des tâches ne nécessite que la connaissance des dépendances entre les données. Il suffit d'indiquer sur quelles données chaque tâche va opérer et si ces données sont accédées en lecture (R), écriture (W) ou lecture/écriture (RW). Il existe une littérature plus qu'abondante sur ces techniques algorithmiques. Ce n'est pas l'objet de cette thèse de rentrer trop dans les détails. Nous renvoyons par exemple à [4, 3].

La programmation efficace et générale de ce type d'algorithme est compliquée [6]. Nous avons donc décidé de déléguer la répartition des tâches sur les processeurs disponibles à une bibliothèque informatique spécialisée. Ce type de bibliothèque, aussi appelée "support d'exécution" ou "runtime" en anglais permet de décrire à la fois le code des tâches ainsi que le type d'accès aux données.

Nous avons choisi le support d'exécution StarPU `http://starpu.gforge.inria.fr/`. Il s'agit d'un logiciel développé depuis plus de 10 ans à l'Inria Bordeaux [1]. Au coeur de StarPU il y a les "codelettes" (ou "codelet" en anglais). Une codelette est une fonction écrite en C99 qui réalise l'implémentation d'une tâche. Une même tâche peut avoir plusieurs implémentations, donc plusieurs codelettes, différentes. Par exemple, pour une opération de calcul donnée, il est possible de programmer plusieurs codelettes avec des optimisations différentes, ainsi qu'une ou plusieurs codelettes pour divers type d'accélérateurs (GPU ou Xeon Phi par exemple). Le fait de pouvoir lancer les codelettes sur plusieurs types de matériel justifie le nom de StarPU que l'on peut aussi orthographier *PU : tous les types de processeurs (C ou G PU) sont accessibles. Une tâche est alors déterminée par ses codelettes ainsi que par la description du type d'accès aux données (R, W ou RW).

Une fois les codelettes et les tâches décrites, le programmeur n'a plus qu'à décrire son algorithme de manière séquentielle en soumettant les tâches à StarPU dans un ordre qui produirait le résultat correct. StarPU se charge alors de distribuer les calculs sur les processeurs et les accélérateurs disponibles sur le noeud de calcul. En fonction des dépendances, les tâches peuvent être réorganisées et exécutées en parallèle si possible. Par ailleurs StarPU dispose de plusieurs stratégies d'ordonnancement ("scheduling") qui permettent de tenir compte des efficacités relatives de codelettes, des vitesses de transfert mémoire entre les accélérateurs, etc. afin d'offrir la meilleure exploitation possible des ressources de calcul.

## 3.2 MPI

StarPU est efficace pour répartir les tâches sur un noeud de calcul d'un supercalculateur. Typiquement, un tel noeud est aujourd'hui constitué d'un ou plusieurs CPU multicoeur et, de plus en plus souvent, d'un ou plusieurs accélérateurs de type GPU. Pour les communications entre noeuds, StarPU utilise l'environnement MPI ("Message Passing Interface"). Pour l'instant, StarPU n'est pas capable de distribuer les tâches entre les noeuds MPI. En revanche, il est capable de générer automatiquement les tâches de communications nécessaires au déroulement correct du graphe des tâches. Pour cela, l'utilisateur doit établir une distribution initiale des données et concevoir un logiciel dans lequel chaque noeud MPI possède une copie du graphe des tâches. Chaque noeud MPI doit alors (formellement) lancer toutes les tâches du graphe. Même si tous le graphe est parcouru, la perte d'efficacité est faible car seules les tâches pour lesquelles les données sont disponibles seront effectivement réalisées. Si une tâche a besoin d'accéder en lecture à des données d'un noeud MPI voisin, StarPU génère automatiquement les dépendances et les transferts nécessaires.

## 3.3 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 de type GPU ou CPU multicoeur. Afin d'écrire les codelettes spécifiques pour le GPU, nous avons choisi l'environnement OpenCL. OpenCL permet d'écrire et d'exécuter en parallèle des programmes (appelés *kernels*) sur tous les processeurs d'un accélérateur.

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 un mécanisme afin de partager du code source de fonction avec OpenCL. Lorsqu'une portion de code source de SCHNAPS est inclus entre deux balises #pragma start_opencl ... #pragma end_opencl, cette portion de code sera également compilée une deuxième fois à l'exécution pour des appels OpenCL. Les fonctions SCHNAPS se comportent donc comme des fonctions C traditionnelles, mais elles peuvent être appelées indifféremment 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. Ces copies mémoire sont coûteuses

et doivent être traitées avec précaution pour éviter notamment les conflits d'accès. Auparavant nous gérions ces transferts nous-mêmes grâces au mécanisme des évènements ("events") OpenCL. Grâce aux évènements, OpenCL permet également l'implémentation d'un graphe des tâches et de ses dépendances. Cependant il faut décrire "à la main" les dépendance et il n'y a pas de mécanisme automatique pour les inférer à partir de l'accès aux données. Depuis l'intégration de StarPU à SCHNAPS nous avons donc préféré déléguer cette gestion au support d'exécution.

## 3.4 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 (entourées des balises balises #pragma start_opencl ... #pragma end_opencl), car elles doivent pouvoir être appelées depuis les kernels OpenCL.

## 3.5 Sous-domaines

Afin d'atteindre de bonnes performances en calcul parallèle, SCHNAPS repose sur plusieurs niveaux de parallélisme. Le premier niveau correspond à celui des sous-domaines. C'est un parallélisme à gros grain. Les transferts entre sous-domaines sont réalisés grâce à la bibliothèque MPI par StarPU.

Un niveau intermédiaire de parallélisme est piloté par StarPU qui distribue les tâches de calcul sur les accélérateurs du noeud MPI. Enfin, un troisième niveau de parallélisme à grain fin est piloté au niveau des accélérateurs OpenCL.

Les maillages de SCHNAPS sont organisés pour suivre cette hiérarchie. D'abord, le domaine $\Omega$ est découpé en *sous-domaines*. Chaque sous-domaine est associé à un noeud MPI. Un noeud MPI ne possède que les données de son sous-domaine. Mais il dispose aussi d'informations minimales sur le reste du maillage afin de pouvoir construire le graphe des tâches StarPU du calcul global.

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

## 3.6 Macrocellules et champs

Par ailleurs, chaque sous-domaine est lui-même découpé en macrocellules. Les macrocellules sont obtenus grâce à un diffémorphisme (transformation géométrique) qui envoie le cube de référence $[0,1]^3$ sur un hexaèdre courbé. Il est possible d'envisager des transformation trilinéaires : hexaèdre à 8 noeud de type H8 dans la terminologie élément fini ; ou des transformations quadratiques : hexaèdre à 20 noeud de type H20.

Les macro-cellules sont ensuite subdivisées dans chaque directions en sous-cellules partageant les mêmes caractéristiques (modélisation physique, interpolation). Notons que dans chaque macrocellule, les sous-cellules sont organisées suivant une grille courbe mais régulière et que les données géométriques de type H8 ou H20 sont partagées entre toutes les sous-cellules. Cette organisation a deux avantages. Elle permet d'abord de manipuler des structures de données géométriques légères car seul un maillage grossier est nécessaire. C'est ce maillage grossier (macro-maillage), qui est partagé entre les noeuds MPI et qui permet de construire le graphe des tâches StarPU. D'autre part, l'organisation régulière des sous-cellules permet d'envisager des optimisations très efficaces notamment dans les calculs sur GPU. Les paramètres de découpage sont en général décrit dans deux tableaux deg[0 :2] et raf[0 :2]. deg[i] représente le degré d'interpolation dans la direction du maillage logique i et raf[i] le nombre de sous-cellules dans cette direction. Dans la direction i il y a donc (deg[i]+1)*raf[i] points d'interpolation. Il est possible d'avoir des paramètres de discrétisation différents suivant chaque direction, ce qui permet par exemple de réaliser des calculs 1D ou 2D, même si la structure du maillage est toujours 3D.

A chaque macro-cellule du macro-maillage est associé un champ. Cette structure contient les valeurs des variables conservatives sur les points d'interpolation à l'intérieur des sous-cellules.

Les échanges de données entre macrocellules et le calcul des conditions aux limites sont assurés par une structure d'interface. Cette structure est en particulier chargée d'extraire les données de bord des macrocellules dans des tampons. Ces tampons sont très utiles pour optimiser les dépendances de données et offrir plus de liberté à StarPU pour optimiser le parcours du graphe des tâches

Nous réalisons les calculs sur les champ des macrocellules à partir de codelettes

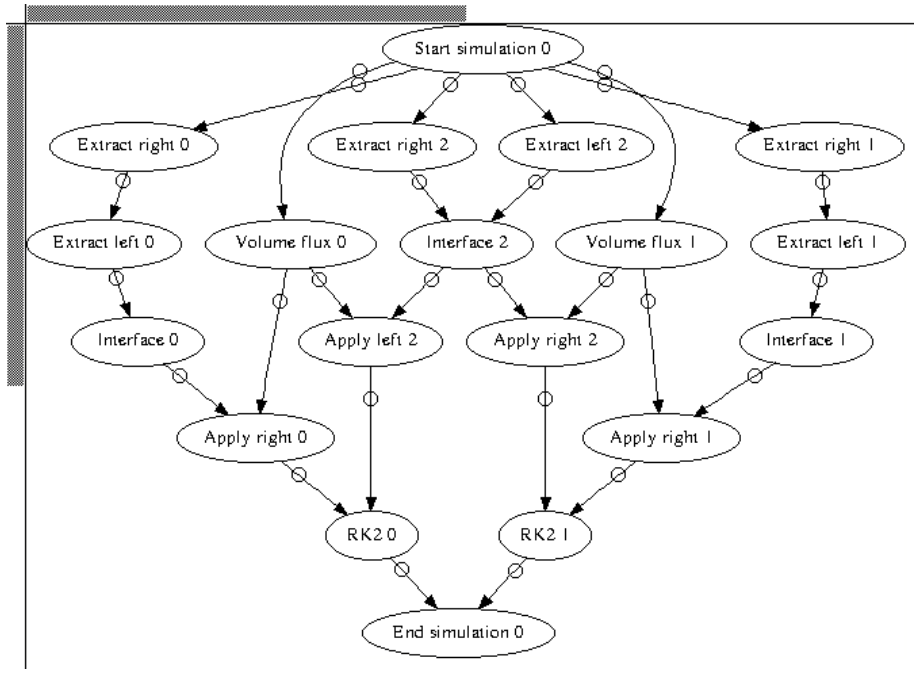


FIGURE 1 – Graphe des tâches pour un calcul dans un sous-domaine contenant 3 zones. Les tâches sont représentées par les noeuds du graphe et les flèches représentent les dépendances entre les tâches (une tâche doit attendre la fin des tâches dont elle dépend avant de démarrer).

StarPU. Ces codelettes calculent les flux, les termes sources dans les macrocellules. Les codelettes calculent également les flux provenant des interfaces. Ils réalisent enfin les extractions de données volumiques vers les interfaces. Nous avons écrit à chaque fois deux versions des codelettes : une version C pour CPU et une version OpenCL qui peut s'exécuter sur un CPU multicoeur ou un GPU.

## 3.7 Simulation

Une *simulation* est une collection de champs sur des macrocellules. Cette classe est conçue pour contenir les algorithmes à appliquer sur les macrocellules. En particulier, c'est cette classe qui lance les tâches StarPU.

## 3.8 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 H8 ou H20. A l'avenir nous envisageons de traiter également des tétraèdres ou d'autres éléments courbes.

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). En général, dans SCHNAPS nous utilisons des éléments quadratiques de type H20, mais la description du H8 est plus simple. 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

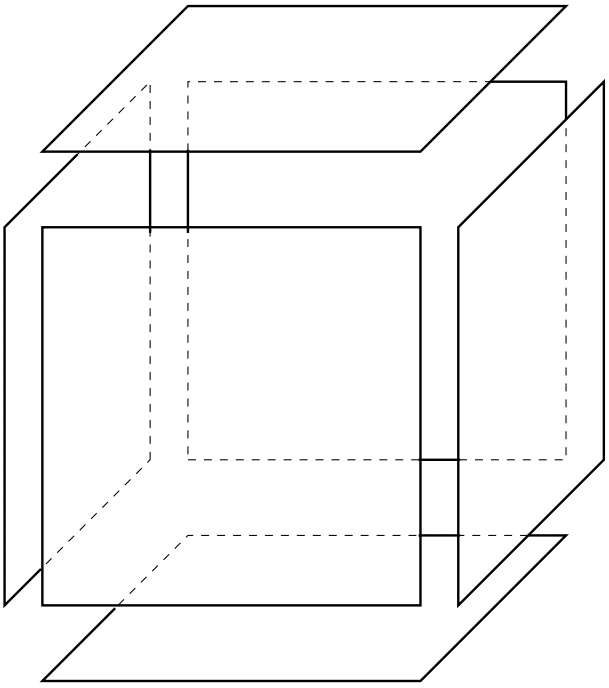

FIGURE 2 – Cube de référence.

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 $\tau_L$ 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 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 struture geometry. Cette classe contient des fonctions SCHNAPS (qui pourront donc être appelées depuis les kernel OpenCL) pour calculer la transformation géométrique $\tau$ en un point de référence, son gradient, son jacobien, son inverse, le vecteur normal dans le cas d'un élément surfacique, *etc.*

## 3.9 Interpolation

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

Pour l'instant dans SCHNAPS, pour des raisons d'efficacité, nous utilisons une approximation nodale adaptée à la quadrature numérique (voir [2, 5]) : les champs sont définis aux points d'intégration de Gauss-Lobatto des éléments. Ce choix permet d'accéder directement aux champs pour l'intégration numérique dans le volume

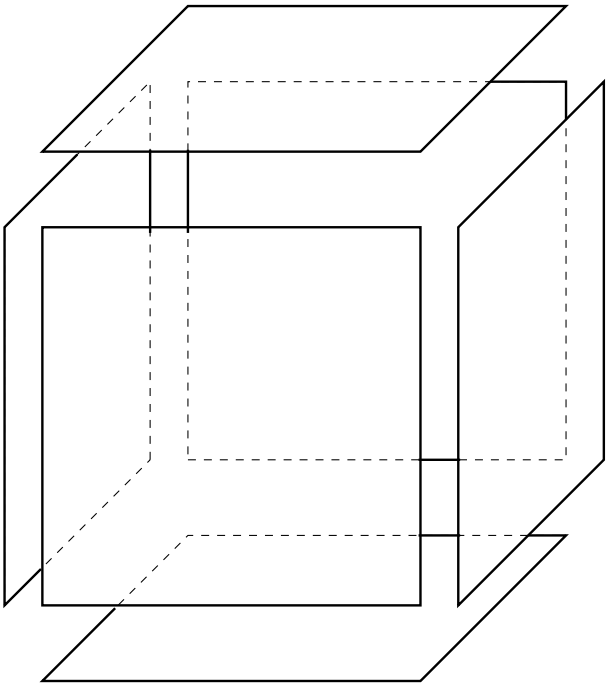

FIGURE 3 – Cube de référence.

et sur les faces. Ce choix assure aussi que les matrices masses locales sont diagonales.

Pour commencer, nous décrivons par exemple comment nous interpolons les champs sur un élément de type H8 avec des polynômes de degré $d$. Pour simplifier, nous considérons d'abord le cas où la macrocellule correspondante ne contient qu'une seule sous-cellules. C'est à dire nous supposons que raf[0]=raf[1]=raf[2]=1 dans les trois directions.

Pour l'interpolation nous fixons d'abord dans la cellule $L$ un degré $d$. Nous considérons les $(d+1)$ points de Gauss-Lobatto $(\xi_i)_{i=0\cdots d}$ sur $[0,1]$, et les poids d'intégration correspondants $\omega_i$ . Nous notons aussi $I_k$ le $k^{\text{ième}}$ polynôme de Lagrange associé aux points $\xi_i$. 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 $q = (d+1)^2k + (d+1)j + i$ 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 `interpolation` 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.

## 3.10 Interpolation de Gauss-Lobatto avec des sous-cellules

En fait, chaque macrocellule peut-être découpée en plusieurs sous-cellules. Pour l'interpolation nous fixons d'abord dans la macrocellule $L$ 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 $\omega_i$. 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 $\xi_i$. 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.

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

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 3.8 pour retrouver la position du point de Gauss-Lobatto dans le volume à l'aide des formules suivantes

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

Les indices de la sous-cellule sont donnés par

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

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

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 $\Pi$ la transformation qui fait passer de la numérotation $q'$ sur le bord $\partial L$ à la numérotation $q$ à l'intérieur de la cellule

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

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

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

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

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

## 3.11 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 $\partial L$, 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 $n_{r\tilde{r}}$.

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

to do (je m'en occupe)

## 4 Programmation de la MHD dans SCHNAPS

### 4.1 Commentaires sur la programmation

Description des fonctions fluxnum, etc.

### 4.2 Premières validations

Validation pseudo 1D. Orszag tang 2D périodique

### 4.3 Tests de performances

comparaisons : ordre, StarPU, 2D/3D activation ou désactivation des CPU/GPU dans StarPU.

## Références


[1] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. Starpu : a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation : Practice and Experience*, 23(2) :187–198, 2011.

[2] Gary Cohen, Xavier Ferrieres, and Sébastien Pernet. A spatial high-order hexahedral discontinuous galerkin method to solve maxwell's equations in time domain. *Journal of Computational Physics*, 217(2) :340–363, 2006.

[3] Michel Cosnard and Emmanuel Jeannot. Compact dag representation and its dynamic scheduling. *Journal of Parallel and Distributed Computing*, 58(3) :487–514, 1999.

[4] Michel Cosnard and Michel Loi. Automatic task graph generation techniques. In *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on*, volume 2, pages 113–122. IEEE, 1995.

[5] Andreas Klöckner, Tim Warburton, Jeff Bridge, and Jan S Hesthaven. Nodal discontinuous galerkin methods on graphics processors. *Journal of Computational Physics*, 228(21) :7863–7882, 2009.

[6] Thomas Strub. *Résolution numérique des équations de Maxwell tridimensionnelles instationnaires sur architecture massivement multicoeur*. PhD thesis, PhD thesis, University of Strasbourg, 2015.