Soutenance de Stage

Méthodes de volumes finis sur carte graphique nVidia pour Euler compressible

L.Besson S.Hecht M.Isnard

ENS Cachan

2 juillet 2012

Encadrants F.De Vuyst, J.M.Ghidaglia ; D.Chauveheid, S.Benjelloun.

Aujourd'hui

De quoi va-t-on parler ? Trois points :

- nos problématiques ;
- nos réalisations théoriques ;
- nos réalisations pratiques.

Plan de la présentation

- **Introduction** :
  que fait-on ? dans quel intérêt ?
- **Préliminaires : Équation d'advection** Rappels.
- **Équations d'Euler (théorie)** :
  équations générales en 3D, méthode de *splitting*, schéma en 1D et généralisation ;
  démonstration de notre schéma assistée par ordinateur (`Maple`).
- **Équations d'Euler (pratique)** :
  Difficultés rencontrées, simulations en `C` et en `CUDA` ;
  affichages via la combinaison du format de données `VTK` et du logiciel d'affichage `ParaView` (1D, 2D, 3D ; avec ou sans terme source ; plusieurs conditions de bords) ;
  affichage *directement sur la carte graphique* via `OpenGL`, début d'intéractivité dans les simulations (via `GLut`).
- **Optimisation du code séquentiel et parallèle** :
  Première comparaison des performances, optimiser la visualisation, et résultats finaux.

Plan de la présentation

- **Introduction** :
  que fait-on ? dans quel intérêt ?
- **Préliminaires : Équation d'advection** Rappels.
- **Équations d'Euler (théorie)** :
  équations générales en 3D, méthode de *splitting*, schéma en 1D et généralisation ;
  démonstration de notre schéma assistée par ordinateur (`Maple`).
- **Équations d'Euler (pratique)** :
  Difficultés rencontrées, simulations en `C` et en `CUDA` ;
  affichages via la combinaison du format de données `VTK` et du logiciel d'affichage `ParaView` (1D, 2D, 3D ; avec ou sans terme source ; plusieurs conditions de bords) ;
  affichage *directement sur la carte graphique* via `OpenGL`, début d'intéractivité dans les simulations (via `GLut`).
- **Optimisation du code séquentiel et parallèle** :
  Première comparaison des performances, optimiser la visualisation, et résultats finaux.

Plan de la présentation

- **Introduction** :
  que fait-on ? dans quel intérêt ?
- **Préliminaires : Équation d'advection** Rappels.
- **Équations d'Euler (théorie)** :
  équations générales en 3D, méthode de *splitting*, schéma en 1D et généralisation ;
  démonstration de notre schéma assistée par ordinateur (`Maple`).
- **Équations d'Euler (pratique)** :
  Difficultés rencontrées, simulations en `C` et en `CUDA` ;
  affichages via la combinaison du format de données `VTK` et du logiciel d'affichage `ParaView` (1D, 2D, 3D ; avec ou sans terme source ; plusieurs conditions de bords) ;
  affichage *directement sur la carte graphique* via `OpenGL`, début d'intéractivité dans les simulations (via `GLut`).
- **Optimisation du code séquentiel et parallèle** :
  Première comparaison des performances, optimiser la visualisation, et résultats finaux.

Plan de la présentation

- **Introduction** :
  que fait-on ? dans quel intérêt ?
- **Préliminaires : Équation d'advection** Rappels.
- **Équations d'Euler (théorie)** :
  équations générales en 3D, méthode de *splitting*, schéma en 1D et généralisation ;
  démonstration de notre schéma assistée par ordinateur (`Maple`).
- **Équations d'Euler (pratique)** :
  Difficultés rencontrées, simulations en `C` et en `CUDA` ;
  affichages via la combinaison du format de données `VTK` et du logiciel d'affichage `ParaView` (1D, 2D, 3D ; avec ou sans terme source ; plusieurs conditions de bords) ;
  affichage *directement sur la carte graphique* via `OpenGL`, début d'intéractivité dans les simulations (via `GLut`).
- **Optimisation du code séquentiel et parallèle** :
  Première comparaison des performances, optimiser la visualisation, et résultats finaux.

Plan de la présentation

- **Introduction** :
  que fait-on ? dans quel intérêt ?
- **Préliminaires : Équation d'advection** Rappels.
- **Équations d'Euler (théorie)** :
  équations générales en 3D, méthode de *splitting*, schéma en 1D et généralisation ;
  démonstration de notre schéma assistée par ordinateur (`Maple`).
- **Équations d'Euler (pratique)** :
  Difficultés rencontrées, simulations en `C` et en `CUDA` ;
  affichages via la combinaison du format de données `VTK` et du logiciel d'affichage `ParaView` (1D, 2D, 3D ; avec ou sans terme source ; plusieurs conditions de bords) ;
  affichage *directement sur la carte graphique* via `OpenGL`, début d'intéractivité dans les simulations (via `GLut`).
- **Optimisation du code séquentiel et parallèle** :
  Première comparaison des performances, optimiser la visualisation, et résultats finaux.

Introduction

Que fait-on ?

Simulations numériques de problèmes de mécanique des fluides.

Intérêt ? Deux principaux

Industriel : industrie pétrolière, métallurgie, pétro-chimie ;

Pédagogique : des problèmes bien étudiés donc bien connus ; mais des simulations numériques qui peuvent toujours s'améliorer, via de nouveaux outils (nVidia CUDA, openMP).

Des applications gourmandes en temps de calcul !

- nécessite une grosse puissance de calcul pour avoir une bonne précision et une bonne stabilité ;
- mais ces problèmes sont *"bien"* parallélisables (schémas locaux).

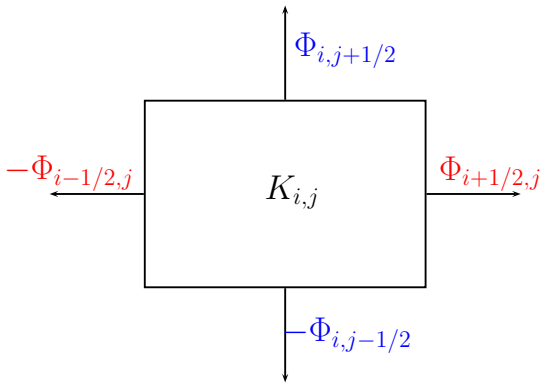
$\Longrightarrow$ Programmation parallèle sur cartes graphiques : outil nVidia CUDA.

L'équation d'avection (en 2D)

$$\frac{\partial u}{\partial t} + div(u\overrightarrow{c}) = 0.$$

$u : \mathbb{R}_+ \times \mathbb{R}^2 \to \mathbb{R}$, représente une quantité associée à une particule de fluide ;
$\overrightarrow{c} = [a, b]^T$: est une vitesse, représentant le courant.

On discrétise en *grilles cartésiennes* en temps et en espace, puis on intègre et on applique la formule de Green sur *chaque* volume de contrôle.



- Valeur moyenne sur un volume de contrôle $K_{i,j}$:
  $u_{i,j} = \frac{1}{\Delta x \Delta y} \int_{K_{i,j}} u d\Omega$
- Flux numérique :
  $\phi^n_{i+1/2,j} = (u\overrightarrow{c} . \overrightarrow{n})_{i+1/2,j}$

Schéma numérique

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t_n} + \frac{1}{\Delta x \Delta y}[\phi_{i+1/2,j}^{n}\Delta y + \phi_{i-1/2,j}^{n}\Delta y + \phi_{i,j+1/2}^{n}\Delta x + \phi_{i,j-1/2}^{n}\Delta x] = 0 \quad (1)$$

en posant $\phi_{i+1/2,j}^{n} = (\overrightarrow{c}\, u.\overrightarrow{n})_{i+1/2,j}$ (voir [DeDu]).

Choix de la méthode de calcul des flux numériques

Méthode VFFC (d'après [Ghi99]) :

$$\phi_{i+1/2,j}^{n} = \frac{F(u_{i,j}) + F(u_{i+1,j})}{2}.\overrightarrow{n} - sgn(\overrightarrow{c}.\overrightarrow{n})\frac{F(u_{i+1,j}) - F(u_{i,j})}{2}.\overrightarrow{n} \quad (2)$$

On utilise le *splitting directionnel* : on se ramène à plusieurs schémas 1D successifs.

Équation d'advection et extension du modèle

Rappels

Simulation de l'équation d'advection, à vitesse c variable.

Champ de vitesse $[u, v]$ tournant, maillage 512×512 points 1024 étapes.

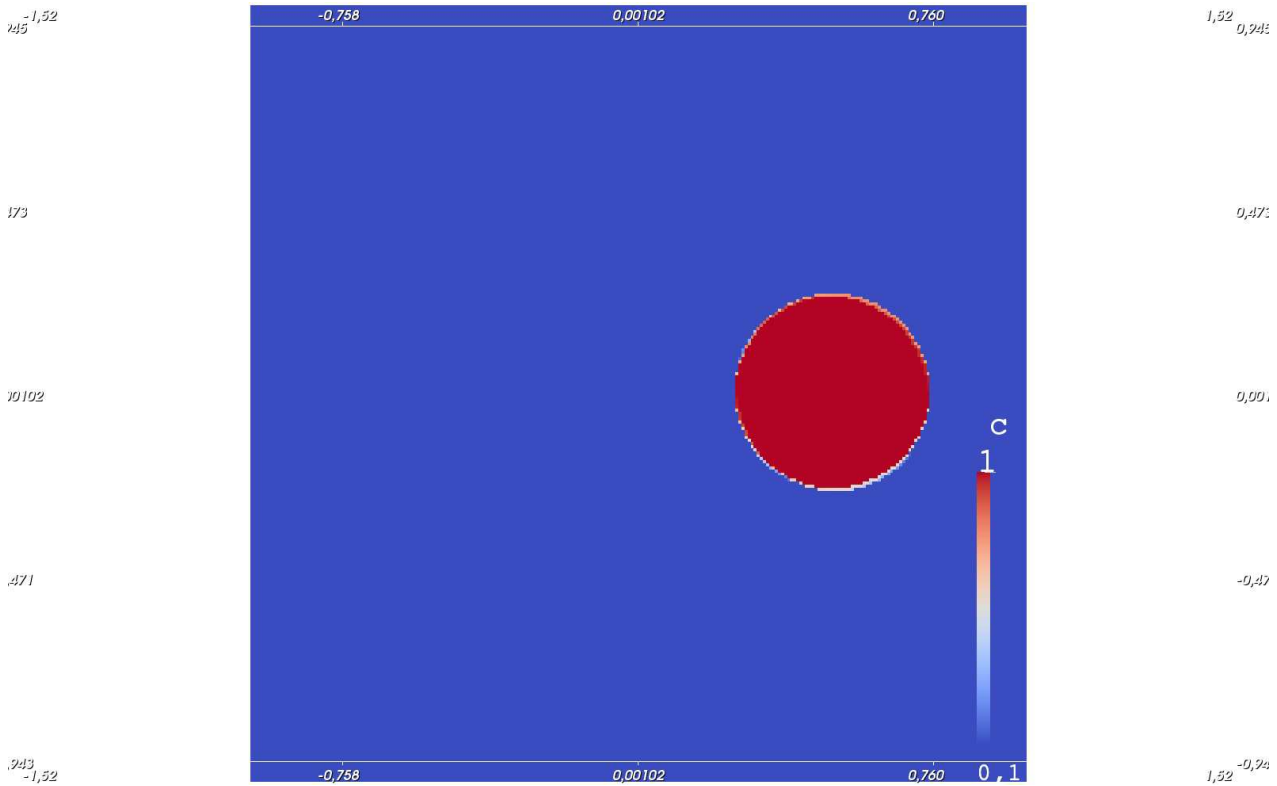


FIGURE: Affichage en 2D de c.

Équation d'advection et extension du modèle

CPU ? GPU ? CUDA ? (voir [KrSa])

- **CPU** : processeur, hôte ;
- **GPU** : carte graphique, périphérique ;
- Un **kernel** est une portion de code parallèle à exécuter sur le périphérique. Chacune de ses instances s'appelle un **thread** ;
- Une **grille** est constituée de **blocs**. Chaque bloc est utilisé pour plusieurs **threads**.

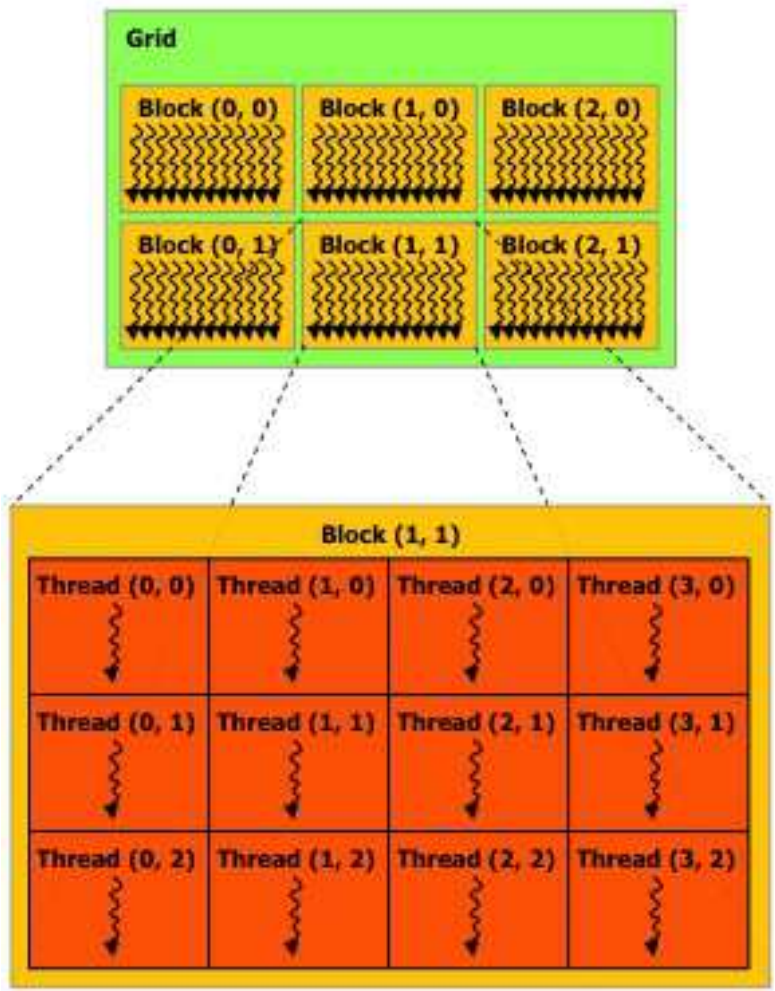


Figure 2-1. Grid of Thread Blocks

Modélisent le comportement d'un **fluide compressible non visqueux**.

Les équations physiques

- Équation de *continuité* (conservation de la masse) :

$$\partial_t \rho + div(\rho.\overrightarrow{W}) = 0; \tag{3}$$

- Équations du *mouvement* (1 par dimension) :

$$\partial_t(\rho.\overrightarrow{W}) + div(\rho.\overrightarrow{W} \otimes \overrightarrow{W} + p.Id) = 0; \tag{4}$$

- Équation d'*énergie* (température) où $E := e + \frac{1}{2}.W^2$:

$$\partial_t(\rho E) + div((\rho E + p).\overrightarrow{W}) = 0. \tag{5}$$

$\Longrightarrow$ *Il manque une équation pour fermer le système !*

Modélisent le comportement d'un **fluide compressible non visqueux**.

Les équations physiques

- Équation de *continuité* (conservation de la masse) :

$$\partial_t \rho + div(\rho.\overrightarrow{W}) = 0; \tag{3}$$

- Équations du *mouvement* (1 par dimension) :

$$\partial_t(\rho.\overrightarrow{W}) + div(\rho.\overrightarrow{W} \otimes \overrightarrow{W} + p.Id) = 0; \tag{4}$$

- Équation d'*énergie* (température) où $E := e + \frac{1}{2}.W^2$:

$$\partial_t(\rho E) + div((\rho E + p).\overrightarrow{W}) = 0. \tag{5}$$

$\Longrightarrow$ *Il manque une équation pour fermer le système !*

Modélisent le comportement d'un **fluide compressible non visqueux**.

Les équations physiques

- Équation de *continuité* (conservation de la masse) :

$$\partial_t \rho + div(\rho.\overrightarrow{W}) = 0; \tag{3}$$

- Équations du *mouvement* (1 par dimension) :

$$\partial_t(\rho.\overrightarrow{W}) + div(\rho.\overrightarrow{W} \otimes \overrightarrow{W} + p.Id) = 0; \tag{4}$$

- Équation d'*énergie* (température) où $E := e + \frac{1}{2}.W^2$:

$$\partial_t(\rho E) + div((\rho E + p).\overrightarrow{W}) = 0. \tag{5}$$

$\Longrightarrow$ *Il manque une équation pour fermer le système !*

Modélisent le comportement d'un **fluide compressible non visqueux**.

Les équations physiques

- Équation de *continuité* (conservation de la masse) :

$$\partial_t \rho + div(\rho.\overrightarrow{W}) = 0; \tag{3}$$

- Équations du *mouvement* (1 par dimension) :

$$\partial_t(\rho.\overrightarrow{W}) + div(\rho.\overrightarrow{W} \otimes \overrightarrow{W} + p.Id) = 0; \tag{4}$$

- Équation d'*énergie* (température) où $E := e + \frac{1}{2}.W^2$:

$$\partial_t(\rho E) + div((\rho E + p).\overrightarrow{W}) = 0. \tag{5}$$

$\Longrightarrow$ *Il manque une équation pour fermer le système !*

Équations d'Euler (Théorie)

Présentation des équations (2)

On utilise l'équation d'état des *gaz parfaits* (implique une restriction du domaine d'application de ces équations) :

Équation d'état des gaz parfaits

$$e = \frac{1}{\gamma - 1} \times \frac{p}{\rho}. \tag{6}$$

Notations (inspirée de [Buf06])

- p : pression, ρ : masse volumique, $\overrightarrow{W} = [u, v, w]^T$: vitesse du fluide (1, 2 ou 3 composantes) ;
- e : énergie interne massique du fluide, γ : rapport des chaleurs spécifiques (constant) et vaut $\gamma = 1.4$ pour un gaz diatomique (air) ;
- $\otimes$: produit tensoriel, Id : matrice identité ;
- c est la "*vitesse du son*" : $c = \sqrt{\frac{\gamma \times p}{\rho}}$.

On peux maintenant présenter les équations sous forme vectorielle :

Équations d'Euler sous forme vectorielle

$$\overrightarrow{U} \equiv [\rho; \rho\overrightarrow{W}; E]^T = [\rho; \rho u; \rho v; \rho w; E]^T; \tag{7}$$

$$\frac{\partial}{\partial t}\overrightarrow{U} + \frac{\partial}{\partial x}\overrightarrow{F}(\overrightarrow{U}) + \frac{\partial}{\partial y}\overrightarrow{G}(\overrightarrow{U}) + \frac{\partial}{\partial z}\overrightarrow{H}(\overrightarrow{U}) = \overrightarrow{S}(\overrightarrow{U}). \tag{8}$$

Les flux associés sont alors définis par :

$$\overrightarrow{F}(\overrightarrow{U}) \equiv \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(E+p) \end{bmatrix}; \quad \overrightarrow{G}(\overrightarrow{U}) \equiv \begin{bmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ v(E+p) \end{bmatrix}; \quad \overrightarrow{H}(\overrightarrow{U}) \equiv \begin{bmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ w(E+p) \end{bmatrix}.$$

$\overrightarrow{S}(\overrightarrow{U})$ est un terme source , qui peut être nul *(détails plus loin)*.

Équations d'Euler (Pratique)

Visualisation via GNUPLOT : exemple pour Euler 1D

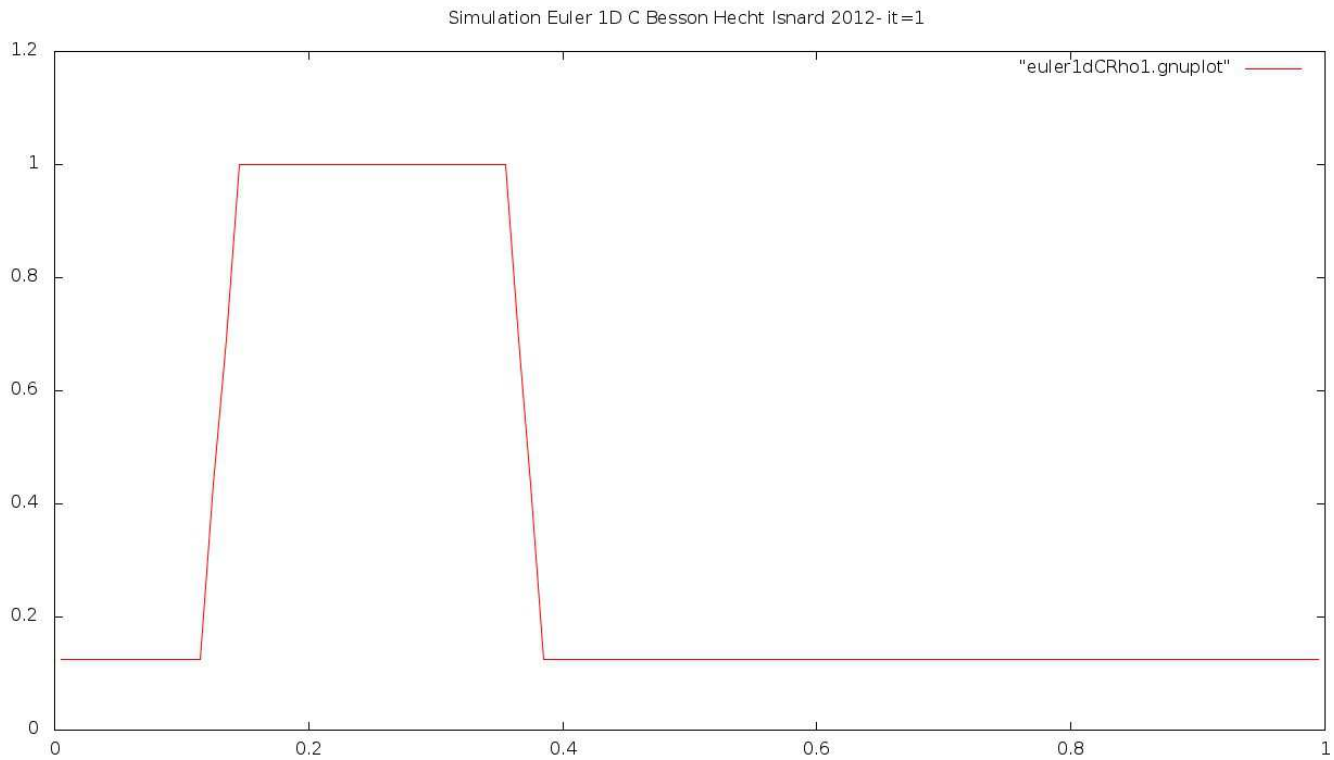


FIGURE: Affichage en 1D de ρ pour Euler compressible (GNUPlot). 400 points, condition de bord périodique, condition initiale créneau.

On utilise une méthode de *splitting* : trois étapes mono-dimensionnelles. On va détailler le schéma en 3D (d=3) selon x.

Discrétisation en temps et espace

Exactement les mêmes notations et conventions que pour l'équation d'advection : on intègre sur une cellule $K_{i,j,k}$:

$$\frac{\partial}{\partial t}\int_{K_{i,j,k}} \overrightarrow{U}\, dV \simeq \frac{\overrightarrow{U}^{n+1}_{i,j,k} - \overrightarrow{U}^{n}_{i,j,k}}{\Delta t^n}\mathbf{Vol}(K_{i,j,k}). \tag{9}$$

Dès lors, pour $(i \in [0; N-1], j \in [0; M-1], k \in [0; O-1])$, on a :

$$\frac{\overrightarrow{U}^{n+1}_{i,j,k} - \overrightarrow{U}^{n}_{i,j,k}}{\Delta t_n} + \frac{1}{\Delta x \Delta y \Delta z}[(\overrightarrow{\phi x}^n_{i+1/2}\Delta y\Delta z + \overrightarrow{\phi x}^n_{i-1/2}\Delta y\Delta z) + \tag{10}$$

$$(\overrightarrow{\phi y}^n_{j+1/2}\Delta x\Delta z + \overrightarrow{\phi y}^n_{j-1/2}\Delta x\Delta z) + (\overrightarrow{\phi z}^n_{k+1/2}\Delta x\Delta y + \overrightarrow{\phi z}^n_{k-1/2}\Delta x\Delta y)] = 0.$$

Rappel de l'expression des flux ϕ pour l'advection (selon x)

$$\phi^n_{i+1/2,j} = \frac{F(u_{i,j}) + F(u_{i+1,j})}{2}\vec{n} - sgn(\vec{c}.\vec{n})\frac{F(u_{i+1,j}) - F(u_{i,j})}{2}\vec{n} \quad (11)$$

De même on a ici :

Pour les flux selon x (toujours VFFC)

Se transforme ici en (On n'écrit pas les j, k pour chaque valeur) :

$$\overrightarrow{\phi x}^n_{i+1/2} = \frac{\vec{F}(\vec{U}^n_{i+1}) + \vec{F}(\vec{U}^n_i)}{2} - sign(A^n_{i+1/2})\frac{\vec{F}(\vec{U}^n_{i+1}) - \vec{F}(\vec{U}^n_i)}{2} \quad (12)$$

où on a posé $A^n_{i+1/2}$ la **matrice jacobienne** du vecteur $\vec{F}(\vec{U}^n_{i+1/2})$.

$\Longrightarrow$ *La difficulté est le calcul de cette matrice signe !*

Définition

$sign(A)$ est la matrice semblable à A et dont les valeurs propres sont les signes des valeurs propres de A ;
ie : $\ker sign(A) = \{sign(\lambda_1); \ldots; sign(\lambda_p)\}$, où $sign(x) := 1$ si $x > 0$, $:= -1$ si $x < 0$ et $:= 0$ si $x = 0$, et $\ker A = \{\lambda_1; \ldots; \lambda_p\}$.

Théorème (Calcul de $A^{\overrightarrow{n}}$ et $sign(A^{\overrightarrow{n}})$)

La matrice $A^{\overrightarrow{n}}$ est diagonalisable.
On peut alors écrire : $A^{\overrightarrow{n}} = L^{\overrightarrow{n}}.\Lambda^{\overrightarrow{n}}.R^{\overrightarrow{n}}$, et $sign(A^{\overrightarrow{n}}) = L^{\overrightarrow{n}}.sign(\Lambda^{\overrightarrow{n}}).R^{\overrightarrow{n}}$.

Le papier [GhiPa] nous donne les valeurs des matrices de vecteurs propres à gauche L , à droite R et des valeurs propres Λ.
Le calcul "à la main" n'est pas faisable.

Équations d'Euler (Théorie)

Valeurs des matrices $A^{\vec{n}}$ et $\Lambda^{\vec{n}}$

Nous avons retrouvé les valeurs formelles de ces matrices par une feuille de calcul formelle via `Maple`.

Notre matrice $A^{\vec{n}}$ vaut

$$A^{\vec{n}} = \begin{bmatrix} 0 & \vec{n} & 0 \\ K.\vec{n} - (\vec{W}.\vec{n})\vec{n} & \vec{W} \otimes \vec{n} - k\,\vec{n} \otimes \vec{W} + (\vec{u}.\vec{n})Id & k\vec{w} \\ (K-H)\vec{W}.\vec{n} & H\vec{n} - k(\vec{W}.\vec{n})\vec{W} & (1+k)\vec{W}.\vec{n} \end{bmatrix} \tag{13}$$

Valeurs propres $\Lambda^{\vec{n}}$ selon l'interface de normale $\vec{n}$

$$\Lambda^{\vec{n}} = \mathbf{diag}[\vec{W}.\vec{n} - c; \vec{W}.\vec{n}; \vec{W}.\vec{n}; \vec{W}.\vec{n}; \vec{W}.\vec{n} + c] \tag{14}$$

Notations : $k := \gamma - 1$, $H := e + \frac{1}{2}W^2 + \frac{p}{\rho}$, $K := c^2 + k(W^2 - H)$.

On ne détaille pas la démonstration, mais comme pour l'équation d'advection on arrive à (voir [Buf06] ou [DeDu]) :

Condition **CFL**

Notre schéma numérique est *stable* **ssi** on a :

$$\forall n,\ \ \Delta t_n = CFL.\frac{\min(\Delta x, \Delta y, \Delta z)}{c^n_{i,j,k} + \max\limits_{i,j,k} |W^n_{i,j,k}|} \tag{15}$$

où CFL est le nombre de Courant-Friedrich-Lévy, tel que $0 \leq CFL \leq 1$ (comme pour l'advection).

En pratique, il vaut mieux prendre $CFL = 0.5$ ou $CFL = 0.6$ plutôt que d'espérer être stable pour $CFL = 0.99$.
Et être *numériquement stable* pour ≤ 1 et instable dès que > 1 est un indicateur de la correction de la simulation.

Une "parallélisation" pas triviale depuis notre code séquentiel

Un code séquentiel qu'on avait déjà conçu comme assez économe, et qui a été délicat à passer en parallèle.

Beaucoup d'échange de données entre les mémoires GPU et CPU :

- Δt_n se calcule comme un maximum sur le vecteur de vitesse $\overrightarrow{W}$: on est confronté à un problème délicat d'algorithmique parallèle (**réduction**). Pour le moment, on fait ce calcul en séquentiel sur le CPU *(ce qui implique de* **tripler** *les échanges de mémoire !)* ;
  - Pour l'écriture des données ρ et p à chaque boucle (soit $8NMO$ octets par boucle de temps : pour un maillage $100 \times 100 \times 100$ on a déjà des transferts de l'ordre de la dizaine de mégaoctet par étape). **Jusqu'à 40% du temps de la simulation !** ;

Une "parallélisation" pas triviale depuis notre code séquentiel

Un code séquentiel qu'on avait déjà conçu comme assez économe, et qui a été délicat à passer en parallèle.
Beaucoup d'échange de données entre la mémoire **vive** et **morte** :

Δt_n se calcule comme un maximum sur le vecteur de vitesse $\overrightarrow{W}$: on est confronté à un problème délicat d'algorithmique parallèle (**réduction**). Pour le moment, on fait ce calcul en séquentiel sur le CPU *(ce qui implique de* **tripler** *les échanges de mémoire !)* ;

- Pour l'écriture des données ρ et p à chaque boucle (soit $8NMO$ octets par boucle de temps : pour un maillage $100 \times 100 \times 100$ on a déjà des transferts de l'ordre de la dizaine de mégaoctet par étape). **Jusqu'à 40% du temps de la simulation !** ;

Équations d'Euler (Théorie)

Différentes conditions aux limites

Dans les équations précédentes, le vecteur des variables conservatives $\overrightarrow{U}$ dépend des flux aux interfaces : à gauche $(i-1/2)$ et à droite $(i+1/2)$, en bas $(j-1/2)$ et en haut $(j+1/2)$, et en arrière $(k-1/2)$ et en avant $(k+1/2)$.

Mais ?

Comment calculer les flux aux interfaces extrêmes ?

$\Longrightarrow$ *Différents types de conditions aux limites !*

Conditions limites périodiques

On choisit $\overrightarrow{F}(\overrightarrow{U}^n_N) = \overrightarrow{F}(\overrightarrow{U}^n_0)$; et $\overrightarrow{F}(\overrightarrow{U}^n_{-1}) = \overrightarrow{F}(\overrightarrow{U}^n_{N-1})$; Idem pour j,M selon y et k,O selon z.

En 1D modélise un tube circulaire. En 2D, l'atmosphère. En 3D, rien de très physique !

Conditions limites absorbantes

On choisit $\overrightarrow{F}(\overrightarrow{U}^n_N) = \overrightarrow{F}(\overrightarrow{U}^n_{N-1})$; et $\overrightarrow{F}(\overrightarrow{U}^n_{-1}) = \overrightarrow{F}(\overrightarrow{U}^n_0)$; Idem pour j,M selon y et k,O selon z.

Modélise un tube (en 1D), une surface de lac (en 2D) ou une piscine (en 3D) de taille infinie, lorsque les conditions initiales sont assez éloignées des bords.

Conditions limites de mur

Pour calculer les flux aux interfaces extrêmes, on suppose $\overrightarrow{W}.\overrightarrow{n} = 0$.
Il y a plusieurs stratégies pour calculer la pression de la cellule intérieure (stratégie *du pauvre*, du *pas pauvre*, et une autre plus explicite).

Modélise un tube (en 1D), une surface de lac (en 2D) ou une piscine (en 3D) avec des bords sur lesquelles se réfléchissent les ondes étudiées.

Équations d'Euler (Pratique)

Visualisation via GNUPLOT : différentes conditions limites

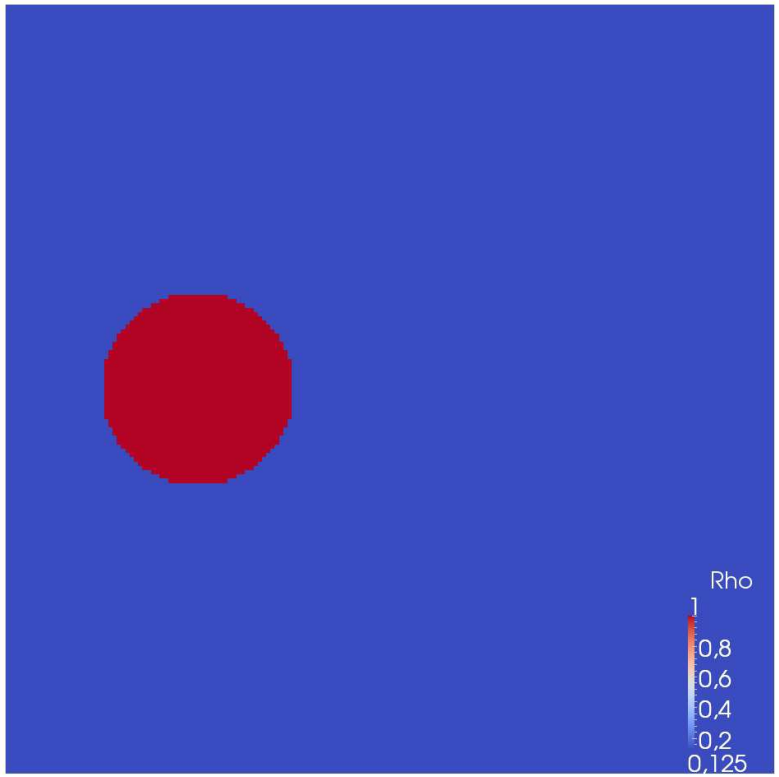


FIGURE: Affichage en 2D pour Euler compressible (ParaView) : conditions périodiques.

Équations d'Euler (Pratique)

Visualisation via GNUPLOT : différentes conditions limites

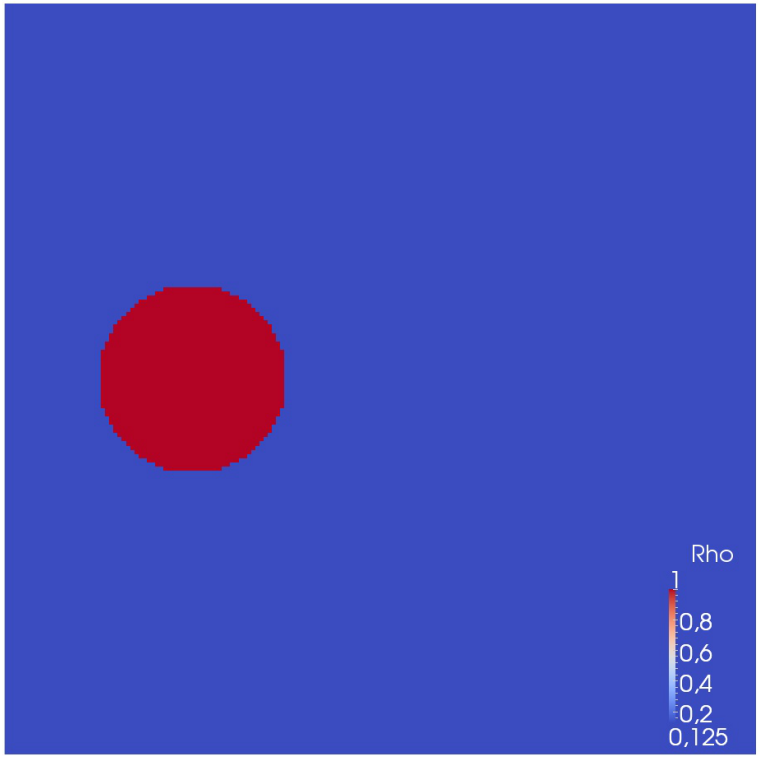


FIGURE: Affichage en 2D pour Euler compressible (ParaView) : conditions absorbantes.

Équations d'Euler (Pratique)

Visualisation via GNUPLOT : différentes conditions limites

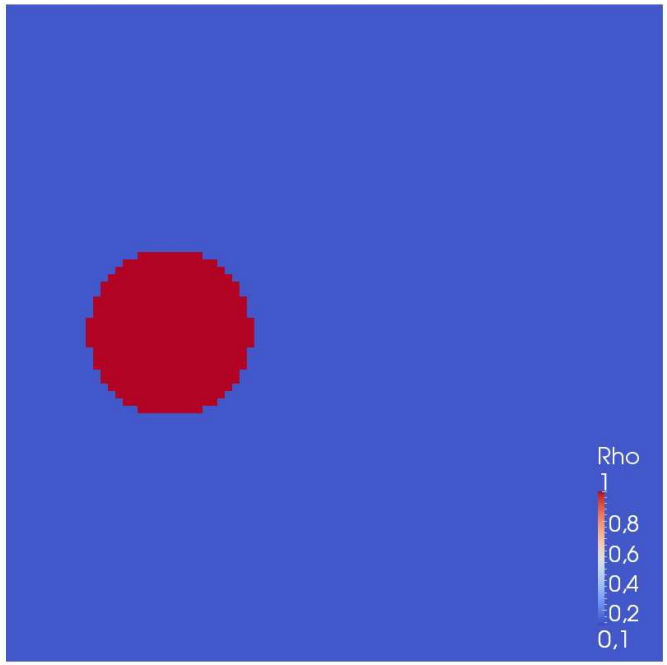


FIGURE: Affichage en 2D pour Euler compressible (ParaView) : conditions de mur.

Équations d'Euler (Théorie)

Terme source S(U)

Dans les équations d'Euler présentées plus haut, on peut rajouter un terme source $\overrightarrow{S}(\overrightarrow{U})$.

$\overrightarrow{S}(\overrightarrow{U})$ peut représenter la force de gravité

$$\overrightarrow{S}(\overrightarrow{U}) = \begin{bmatrix} 0 \\ \rho.\overrightarrow{g} \\ \rho.\overrightarrow{g}.\overrightarrow{W} \end{bmatrix} \tag{16}$$

Cela revient à rajouter la force de gravité dans l'équation du mouvement, et le travail du poids dans l'équation de l'énergie.
(Rappel : $\overrightarrow{g} = [0; 0; -g_0]^T, g_0 = 9.81 m.s^{-2}$*)*

Dans notre résolution, on peut ajouter *vraisemblablement* ce terme dans chacune des trois étapes (mais une seule fois en tout). Nous avons choisi de le faire dans le splitting en z.

Optimisation du code séquentiel et parallèle

Première comparaison des performances (solveur advection)

Les paramètres de la comparaison sont d'abord 256×256 points, puis 512×512, ensuite 1024×1024 et enfin 2048×2048 points, et 1024 étapes en temps.

Codes	Temps mis (256x256)	(512x512)	(1024x1024)	(2048x2048)
C	43s	2m32s	10m41s	40m21s
CUDA	37s	39s	42s	45s

Bref, il est immédiat de constater que, comme prévu, le calcul parallélisé avec CUDA est le plus rapide, mais surtout **peu dépendant du nombre de points** !
Nous précisons qu'ici n'est pas compté le temps de *compilation*, mais que l'*initialisation* et les *écritures dans les fichiers* .vtk sont comptées (une étape sur 20 seulement).

- **GNUplot et VTK+Paraview** Ces deux outils nécessitent une étape que l'expérience à montrée comme étant la **plus couteuse en temps de calcul** : l'écriture des valeurs du vecteur à afficher, une à une, dans un fichier de la mémoire morte de l'ordinateur.
  En effet, de nombreuses simulations ont permis de montrer que jusqu'à 40% du temps pouvait être dépensé cette phase de création du fichier, écriture, et fermeture du fichier.
- **OpenGL** Nous nous sommes interessé à un troisième moyen de visualisation : OpenGL (Open Graphics Library) . C'est une bibliothèque qui permet un affichage en 3D *en temps réel*.

- **GNUplot et VTK+Paraview** Ces deux outils nécessitent une étape que l'expérience à montrée comme étant la **plus couteuse en temps de calcul** : l'écriture des valeurs du vecteur à afficher, une à une, dans un fichier de la mémoire morte de l'ordinateur.
  En effet, de nombreuses simulations ont permis de montrer que jusqu'à 40% du temps pouvait être dépensé cette phase de création du fichier, écriture, et fermeture du fichier.
- **OpenGL** Nous nous sommes interessé à un troisième moyen de visualisation : OpenGL (Open Graphics Library) . C'est une bibliothèque qui permet un affichage en 3D *en temps réel*.

Code	Temps mis (512x512)	Temps mis (256x256)
GNUplot	4m59s	3m27s
VTK	43s	31s
OpenGL	29s	20s

- l'affichage intéractif avec GNUplot est le plus lent, car il nécessite l'impression du triple de données, l'appel du sous processus, et le traitement des données ;
- la mesure pour l'affichage via VTK est légérement faussée, car ne contient que le temps pris par l'impression et ne compte pas la visualisation post-calcul ;
- il est rassurant de constater que l'investissement en OpenGL aura apporté un vrai plus sur le plan des performances : plus de 30% de gagné !

Optimisation du code séquentiel et parallèle

Difficulté du développement, du débuggage et donc de l'optimisation en CUDA

- Des outils de développement difficilement utilisables (mauvais contact avec *cuda-gdb* et *nvvp*) ;
- Des essais de parallélisation plus "fine" sans résultats satisfaisants (calcul de Δt, calcul matriciel) ;
- Le choix des paramètres de parallélisation (`THREADS` et `GRIDS`) est difficule : beaucoup de tâtonnement !

Une comparaison pour un maillage 728×728, 1024 étapes et des conditions de mur.

Code	En écrivant en .vtk	Sans écrire en .vtk
C	46m28s	32m20s
CUDA	12m3s	5m19s
$\longrightarrow$ Un **speed-up** de près de 7 !		

Optimisation du code séquentiel et parallèle

Difficulté du développement, du débuggage et donc de l'optimisation en CUDA

- Des outils de développement difficilement utilisables (mauvais contact avec *cuda-gdb* et *nvvp*) ;
- Des essais de parallélisation plus "fine" sans résultats satisfaisants (calcul de Δt, calcul matriciel) ;
- Le choix des paramètres de parallélisation (`THREADS` et `GRIDS`) est difficule : beaucoup de tâtonnement !

Une comparaison pour un maillage 728×728, 1024 étapes et des conditions de mur.

Code	En écrivant en .vtk	Sans écrire en .vtk
C	46m28s	32m20s
CUDA	12m3s	5m19s
$\longrightarrow$ Un **speed-up** de près de 7 !		

Optimisation du code séquentiel et parallèle

Difficulté du développement, du débuggage et donc de l'optimisation en CUDA

- Des outils de développement difficilement utilisables (mauvais contact avec *cuda-gdb* et *nvvp*) ;
- Des essais de parallélisation plus "fine" sans résultats satisfaisants (calcul de Δt, calcul matriciel) ;
- Le choix des paramètres de parallélisation (`THREADS` et `GRIDS`) est difficile : beaucoup de tâtonnement !

Une comparaison pour un maillage 728×728, 1024 étapes et des conditions de mur.

Code	En écrivant en .vtk	Sans écrire en .vtk
C	46m28s	32m20s
CUDA	12m3s	5m19s
$\longrightarrow$ Un **speed-up** de près de 7 !		

Conclusion

Un petit bonus !

- OpenGL $\rightarrow$ affichage interne pendant le calcul ;
- OpenGL $\rightarrow$ intéraction !

Pas encore de simulation intéractive fonctionnelle, mais nous y étions presque !
L'affichage via OpenGL en CUDA n'a pas fonctionné aussi bien qu'en C.

Conclusion

Un petit bonus !

- OpenGL → affichage interne pendant le calcul ;
- OpenGL → intéraction !

Pas encore de simulation intéractive fonctionnelle, mais nous y étions presque !
L'affichage via OpenGL en CUDA n'a pas fonctionné aussi bien qu'en C.

Conclusion

- **Euler C** Nous avons réalisé un solveur volume finis pour l'équation d'Euler compressible en 3D en C ;
- **Visualisation** Nous visualisons les données numériques calculées en *temps réel* via OpenGL ou en *post-mortem* via le couple VTK et ParaView ;
- **CUDA** Nous avons porté le code 2D sur GPU via CUDA : un développement pas facile et parfois laborieux ;
- **Optimisation (1)** Nous avons déjà amélioré les performances de nos codes Euler2D en C et CUDA. Nous gagnons presque 10% en temps de calcul avec cette première couche ;
- **Optimisation (2)** L'objectif initial était d'atteindre un **speed-up** de 100, nous avons réussi à avoir presque 6.
- **Intéractif** N'ajoute rien sur le plan mathématique, mais est intéressant sur le plan informatique ... et permet d'ajouter un effet festif indéniable !

Conclusion

- **Euler C** Nous avons réalisé un solveur volume finis pour l'équation d'Euler compressible en 3D en `C` ;
- **Visualisation** Nous visualisons les données numériques calculées en *temps réel* via OpenGL ou en *post-mortem* via le couple VTK et ParaView ;
- **CUDA** Nous avons porté le code 2D sur GPU via `CUDA` : un développement pas facile et parfois laborieux ;
- **Optimisation (1)** Nous avons déjà amélioré les performances de nos codes Euler2D en `C` et `CUDA`. Nous gagnons presque 10% en temps de calcul avec cette première couche ;
- **Optimisation (2)** L'objectif initial était d'atteindre un **speed-up** de 100, nous avons réussi à avoir presque 6.
- **Intéractif** N'ajoute rien sur le plan mathématique, mais est intéressant sur le plan informatique ... et permet d'ajouter un effet festif indéniable !

Conclusion

- **Euler C** Nous avons réalisé un solveur volume finis pour l'équation d'Euler compressible en 3D en `C` ;
- **Visualisation** Nous visualisons les données numériques calculées en *temps réel* via OpenGL ou en *post-mortem* via le couple VTK et ParaView ;
- **CUDA** Nous avons porté le code 2D sur GPU via `CUDA` : un développement pas facile et parfois laborieux ;
- **Optimisation (1)** Nous avons déjà amélioré les performances de nos codes Euler2D en `C` et `CUDA`. Nous gagnons presque 10% en temps de calcul avec cette première couche ;
- **Optimisation (2)** L'objectif initial était d'atteindre un **speed-up** de 100, nous avons réussi à avoir presque 6.
- **Intéractif** N'ajoute rien sur le plan mathématique, mais est intéressant sur le plan informatique ... et permet d'ajouter un effet festif indéniable !

Conclusion

- **Euler C** Nous avons réalisé un solveur volume finis pour l'équation d'Euler compressible en 3D en `C` ;
- **Visualisation** Nous visualisons les données numériques calculées en *temps réel* via OpenGL ou en *post-mortem* via le couple VTK et ParaView ;
- **CUDA** Nous avons porté le code 2D sur GPU via `CUDA` : un développement pas facile et parfois laborieux ;
- **Optimisation (1)** Nous avons déjà amélioré les performances de nos codes Euler2D en `C` et `CUDA`. Nous gagnons presque 10% en temps de calcul avec cette première couche ;
- **Optimisation (2)** L'objectif initial était d'atteindre un **speed-up** de 100, nous avons réussi à avoir presque 6.
- **Intéractif** N'ajoute rien sur le plan mathématique, mais est intéressant sur le plan informatique ... et permet d'ajouter un effet festif indéniable !

Conclusion

- **Euler C** Nous avons réalisé un solveur volume finis pour l'équation d'Euler compressible en 3D en `C` ;
- **Visualisation** Nous visualisons les données numériques calculées en *temps réel* via OpenGL ou en *post-mortem* via le couple VTK et ParaView ;
- **CUDA** Nous avons porté le code 2D sur GPU via `CUDA` : un développement pas facile et parfois laborieux ;
- **Optimisation (1)** Nous avons déjà amélioré les performances de nos codes Euler2D en `C` et `CUDA`. Nous gagnons presque 10% en temps de calcul avec cette première couche ;
- **Optimisation (2)** L'objectif initial était d'atteindre un **speed-up** de 100, nous avons réussi à avoir presque 6.
- **Intéractif** N'ajoute rien sur le plan mathématique, mais est intéressant sur le plan informatique ... et permet d'ajouter un effet festif indéniable !

Conclusion

- **Euler C** Nous avons réalisé un solveur volume finis pour l'équation d'Euler compressible en 3D en `C` ;
- **Visualisation** Nous visualisons les données numériques calculées en *temps réel* via OpenGL ou en *post-mortem* via le couple VTK et ParaView ;
- **CUDA** Nous avons porté le code 2D sur GPU via `CUDA` : un développement pas facile et parfois laborieux ;
- **Optimisation (1)** Nous avons déjà amélioré les performances de nos codes Euler2D en `C` et `CUDA`. Nous gagnons presque 10% en temps de calcul avec cette première couche ;
- **Optimisation (2)** L'objectif initial était d'atteindre un **speed-up** de 100, nous avons réussi à avoir presque 6.
- **Intéractif** N'ajoute rien sur le plan mathématique, mais est intéressant sur le plan informatique ... et permet d'ajouter un effet festif indéniable !

Conclusion

Fin

Merci pour votre attention.
N'hésitez pas à poser vos questions !

Bibliographie

Livres :

DeDu *Systèmes hyperboliques de lois de conservation*, de Bruno Despres et Francois Dubois ;

SaKr *CUDA by example*, de Jason Sanders et Edward Kandrot ;

Ghi99 *An overview of the VFFC-method and tools for the simulation of two-phase flows* de J. M. Ghidaglia ;

GhiPa *On Boundary conditions for multidimensional hyperbolic systems of conservation laws in the finite volume framework* de J. M. Ghidaglia et F. Pascal ;

Sites web, articles :

- `http://www.nvidia.fr/object/what_is_cuda_new_fr.html` Page de `NVidia CUDA` ;

Buf06 `http://ufrmeca.univ-lyon1.fr/~buffat/COURS/AERO_HTML/courshtml1.html` Cours de dynamique des gaz, par Marc BUFFAT ;

L'équation sous forme différentielle en 2D

$$\partial_t u + div(\overrightarrow{c}\, u) = 0 \tag{17}$$

$u : \mathbb{R}_+ \times \mathbb{R}^2 \to \mathbb{R}$: représente une quantité associée à une particule de fluide,
$\overrightarrow{c} = (a, b)^T$: est une vitesse, représentant le courant,

On discretise en espace :
Nombre de colonne : ncsd ;
Nombre de ligne : nlsd ;
La valeur de u dans le carre (i, j) est $u[(i-1)ncsd + j]$

...	...	...	...	...
...	...	...	...	...
...	...	(i,j)	...	...
...	...	...	...	...
1	2	3	...	...

```
1  Write(nom_graphique, u, N, M);
2  fprintf(gnuplot_ptr, "unset key\nset pm3d map\nset time\nset
       xlabel 'x_i, i=0..%d'\nset ylabel 'y_j, j=0..%d'\nset
       zlabel 'Valeurs numeriques de u[i,j]'\nset grid xtics
       ytics\nset hidden3d\nset nokey\nset title 'Besson Hecht
       Isnard 2012 - Param:[c_x,c_y,t,borne]=[%f,%f,%d,%d] - it=%
       d'\n", N, M, a, b, type_cond_init, borne, it);
3  fprintf(gnuplot_ptr, "pause 0.00005\nsplot \"%s\" with lines\
       n", nom_graphique); fflush(gnuplot_ptr);
4  //  Pour ecrire les donnees, afin de les afficher "en temps
       reel"
5  void Write(char *nom, float *f, int n, int m){
6    int i,j;  FILE *file=fopen(nom, "w"); int pasn, pasm;
7    pasn = 1+(n / NOMBRE_MAX_X);  pasm = 1+(m / NOMBRE_MAX_Y);
8    for (i=0; i<n; i = i+pasn){ for (j=0; j<m; j = j+pasm){
9        fprintf(file, "\t\t%f\t\t%f\t\t%f\n", (0.5*hx + i*hx),
             (0.5*hy + j*hy), f[I2D(n,i,j)]);
10     } fprintf(file, "\n");  }
11   fflush(file); }
```

```
1  // Pour ecrire notre matrice rho ou P dans un fichier VTK
2  __host__ void WriteVtk3D(char *nom, double *f, int N, int M,
       int O, float hx, float hy, float hz, char* nom_variable,
       int debugint){
3  (.. // passage pas interessant )
4    fprintf(file,"# vtk DataFile Version 3.0\n# Valeurs
         Euler_3D_POINTS CUDA\nASCII\nDATASET STRUCTURED_POINTS\n
         ");
5    fprintf(file,"DIMENSIONS %d %d %d\n", N,  M,  O);
6    fprintf(file,"ORIGIN %f %f %f\n", 0.5*hx, 0.5*hy, 0.5*hz);
7    fprintf(file,"SPACING %f %f %f\n",  hx, hy, hz);
8    fprintf(file,"POINT_DATA %d\n",   N*M*O);
9    fprintf(file,"SCALARS %s float\n",  nom_variable);
10   fprintf(file,"LOOKUP_TABLE default\n");
11       // L'etape d'ecriture est ici sequentielle avec VTK !
12   for (i=0; i<M; i ++){ for (j=0; j<N; j ++){ for (l=0; l<O;
         l ++){
13         fprintf(file, "%f\n",f[i*N*O + j*O + l]); } } }
14   fflush(file); fclose(file); } }
```

Et à la fin de chaque étape de la boucle en temps :

```
1  // On imprime Rho
2  sprintf(nom_graphique, "Ecrire_valeurs_euler_3D_CUDA_%
       d_rho_%s.vtk", it, nom_conditions_initiales(
       GET_CONDITION_INITIALES));
3  printf("Etape %d :\ten cours ...\tecriture dans le fichier
       externe <%s> en cours [RHO]...\n", it, nom_graphique);
4  WriteVtk3D(nom_graphique, rho, N, M, O, hx, hy, hz, "Rho",
       GET_DEBUG__VOIR_VTK);
```

$\Longrightarrow$ *C'est moins "simple" !*

Et à la fin de chaque étape de la boucle en temps :

```
1  // On imprime Rho
2  sprintf(nom_graphique, "Ecrire_valeurs_euler_3D_CUDA_%
       d_rho_%s.vtk", it, nom_conditions_initiales(
       GET_CONDITION_INITIALES));
3  printf("Etape %d :\ten cours ...\tecriture dans le fichier
       externe <%s> en cours [RHO]...\n", it, nom_graphique);
4  WriteVtk3D(nom_graphique, rho, N, M, O, hx, hy, hz, "Rho",
       GET_DEBUG__VOIR_VTK);
```

$\Longrightarrow$ *C'est moins "simple" !*

Équations d'Euler (Théorie)

Valeurs des matrices $L^{\overrightarrow{n}}$, et $R^{\overrightarrow{n}}$

Nous avons retrouvé les valeurs formelles de ces matrices par une feuille de calcul formelle via `Maple`. *Où on a posé* $\overrightarrow{\Omega_1}$ *et* $\overrightarrow{\Omega_2}$ *une base orthogonale directe de l'hyperplan de normale* $\overrightarrow{n}$.

Notre matrice $R^{\overrightarrow{n}}$ (selon la normale $\overrightarrow{n}$) vaut

$$R^{\overrightarrow{n}} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \\ \overrightarrow{W} - c\overrightarrow{n} & \overrightarrow{W} & \overrightarrow{\Omega_1} & \overrightarrow{\Omega_2} & \overrightarrow{W} + c.\overrightarrow{n} \\ H - (\overrightarrow{W}.\overrightarrow{n})c & H - \frac{c^2}{k} & \overrightarrow{W}.\overrightarrow{\Omega_1} & \overrightarrow{W}.\overrightarrow{\Omega_2} & H + (\overrightarrow{W}.\overrightarrow{n})c \end{bmatrix} \tag{18}$$

Notre matrice $L^{\overrightarrow{n}}$ (selon la normale $\overrightarrow{n}$) vaut

$$L^{\overrightarrow{n}} = \begin{bmatrix} \frac{1}{2c^2}(K + \overrightarrow{W}.\overrightarrow{n}\,c) & \frac{k}{c^2}(H - W^2) & -\overrightarrow{W}.\overrightarrow{\Omega_1} & -\overrightarrow{W}.\overrightarrow{\Omega_2} & \frac{1}{2c^2}(K - \overrightarrow{W}.\overrightarrow{n}\,c) \\ \frac{1}{2c^2}(-k\overrightarrow{W} - \overrightarrow{n}\,c) & \frac{k}{c^2}(\overrightarrow{W}) & \overrightarrow{\Omega_1} & \overrightarrow{\Omega_2} & \frac{1}{2c^2}(-k\overrightarrow{W} + \overrightarrow{n}\,c) \\ \frac{1}{2c^2}.k & -\frac{k}{c^2} & 0 & 0 & \frac{1}{2c^2}.k \end{bmatrix} \tag{19}$$

Équations d'Euler (Théorie)

Valeurs des matrices $L^{\overrightarrow{n}}$, et $R^{\overrightarrow{n}}$

Nous avons retrouvé les valeurs formelles de ces matrices par une feuille de calcul formelle via `Maple`. *Où on a posé* $\overrightarrow{\Omega_1}$ *et* $\overrightarrow{\Omega_2}$ *une base orthogonale directe de l'hyperplan de normale* $\overrightarrow{n}$.

Notre matrice $R^{\overrightarrow{n}}$ (selon la normale $\overrightarrow{n}$) vaut

$$R^{\overrightarrow{n}} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \\ \overrightarrow{W} - c\overrightarrow{n} & \overrightarrow{W} & \overrightarrow{\Omega_1} & \overrightarrow{\Omega_2} & \overrightarrow{W} + c.\overrightarrow{n} \\ H - (\overrightarrow{W}.\overrightarrow{n})c & H - \frac{c^2}{k} & \overrightarrow{W}.\overrightarrow{\Omega_1} & \overrightarrow{W}.\overrightarrow{\Omega_2} & H + (\overrightarrow{W}.\overrightarrow{n})c \end{bmatrix} \quad (18)$$

Notre matrice $L^{\overrightarrow{n}}$ (selon la normale $\overrightarrow{n}$) vaut

$$L^{\overrightarrow{n}} = \begin{bmatrix} \frac{1}{2c^2}(K + \overrightarrow{W}.\overrightarrow{n}\,c) & \frac{k}{c^2}(H - W^2) & -\overrightarrow{W}.\overrightarrow{\Omega_1} & -\overrightarrow{W}.\overrightarrow{\Omega_2} & \frac{1}{2c^2}(K - \overrightarrow{W}.\overrightarrow{n}\,c) \\ \frac{1}{2c^2}(-k\overrightarrow{W} - \overrightarrow{n}\,c) & \frac{k}{c^2}(\overrightarrow{W}) & \overrightarrow{\Omega_1} & \overrightarrow{\Omega_2} & \frac{1}{2c^2}(-k\overrightarrow{W} + \overrightarrow{n}\,c) \\ \frac{1}{2c^2}.k & -\frac{k}{c^2} & 0 & 0 & \frac{1}{2c^2}.k \end{bmatrix} \quad (19)$$

Équations d'Euler (Pratique)

Difficultés rencontrées pour les simulations en C

Un schéma assez complexe

- Le schéma numérique pour effectuer des simulations numériques sur ces équations d'Euler est plus compliqué que celui utilisé pour l'équation d'advection.
- Il fait intervenir deux douzaines de variables, du calcul matriciel, beaucoup de calculs intermédiaires ; et n'est consistant que pour des conditions initiales assez régulière (rien de formel n'a été fait sur cette observation).
- Le schéma explicite présenté, en pré-calculant les valeurs formelles des matrices de passages et des valeurs propres, est un algorithme linéaire dans le nombre de points (on ne peut pas faire mieux) : de l'ordre de $s.d^2NMO$ avec une constante $s \leq 50$.

$\Longrightarrow$ Pour optimiser les performances de nos résolutions, il faut chercher du côté pratique et non théorique (schéma de complexité algorithmique déjà optimale) !

Équations d'Euler (Pratique)

Difficultés rencontrées pour les simulations en C

Un schéma assez complexe

- Le schéma numérique pour effectuer des simulations numériques sur ces équations d'Euler est plus compliqué que celui utilisé pour l'équation d'advection.
- Il fait intervenir deux douzaines de variables, du calcul matriciel, beaucoup de calculs intermédiaires ; et n'est consistant que pour des conditions initiales assez régulière (rien de formel n'a été fait sur cette observation).
- Le schéma explicite présenté, en pré-calculant les valeurs formelles des matrices de passages et des valeurs propres, est un algorithme linéaire dans le nombre de points (on ne peut pas faire mieux) : de l'ordre de $s.d^2NMO$ avec une constante $s \leq 50$.

$\Longrightarrow$ Pour optimiser les performances de nos résolutions, il faut chercher du côté pratique et non théorique (schéma de complexité algorithmique déjà optimale) !

Équations d'Euler (Pratique)

Difficultés rencontrées pour les simulations en C

Un schéma assez complexe

- Le schéma numérique pour effectuer des simulations numériques sur ces équations d'Euler est plus compliqué que celui utilisé pour l'équation d'advection.
- Il fait intervenir deux douzaines de variables, du calcul matriciel, beaucoup de calculs intermédiaires ; et n'est consistant que pour des conditions initiales assez régulière (rien de formel n'a été fait sur cette observation).
- Le schéma explicite présenté, en pré-calculant les valeurs formelles des matrices de passages et des valeurs propres, est un algorithme linéaire dans le nombre de points (on ne peut pas faire mieux) : de l'ordre de $s.d^2NMO$ avec une constante $s \leq 50$.

$\Longrightarrow$ Pour optimiser les performances de nos résolutions, il faut chercher du côté pratique et non théorique (schéma de complexité algorithmique déjà optimale) !

Équations d'Euler (Pratique)

Difficultés rencontrées pour les simulations en C

Un schéma assez complexe

- Le schéma numérique pour effectuer des simulations numériques sur ces équations d'Euler est plus compliqué que celui utilisé pour l'équation d'advection.
- Il fait intervenir deux douzaines de variables, du calcul matriciel, beaucoup de calculs intermédiaires ; et n'est consistant que pour des conditions initiales assez régulière (rien de formel n'a été fait sur cette observation).
- Le schéma explicite présenté, en pré-calculant les valeurs formelles des matrices de passages et des valeurs propres, est un algorithme linéaire dans le nombre de points (on ne peut pas faire mieux) : de l'ordre de $s.d^2NMO$ avec une constante $s \leq 50$.

$\Longrightarrow$ Pour optimiser les performances de nos résolutions, il faut chercher du côté pratique et non théorique (schéma de complexité algorithmique déjà optimale) !

Un film pour Euler 3D avec terme source : vue en coupe.

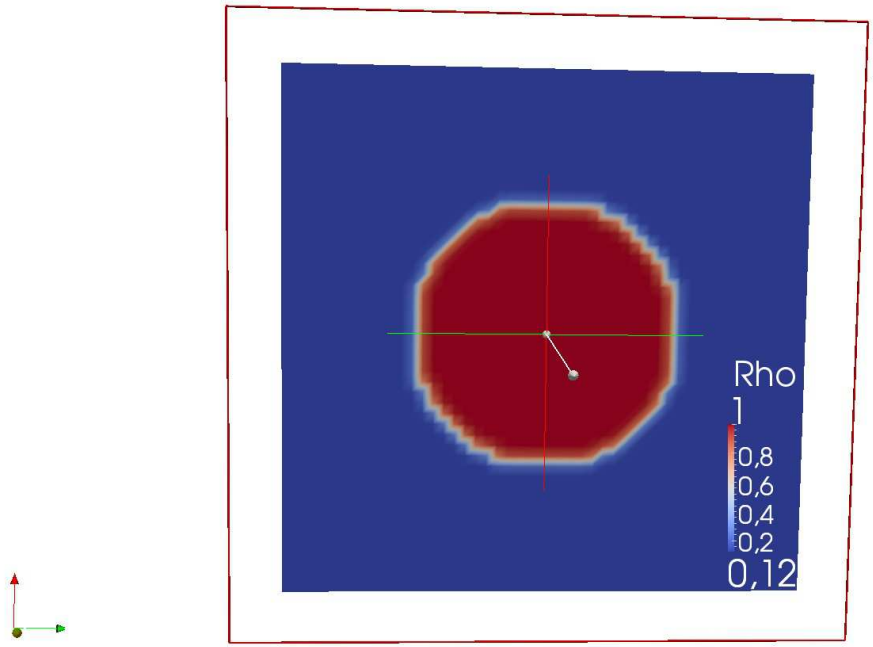


Figure: Affichage en 3D pour Euler compressible (ParaView) : conditions périodiques, avec terme source.