Un solveur volumes finis interactif pour les fluides non-visqueux sur GPU.

Retour d'expérience ...

Julien Bost (Numtech)

Vivien Clauzon (UBP)

Laboratoire de Mathématiques

Université Blaise Pascal et CNRS

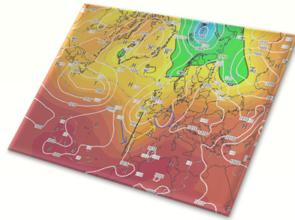
Journées Calcul 2009

Le contexte

- Post-doc en partenariat avec l'entreprise Numtech
- Calcul des champs météo à l'échelle «locale»
- Calcul de la dispersion des polluants dans l'atmosphère
- Mise en place de systèmes opérationnels : aide à la décision : cas chronique et accidentel

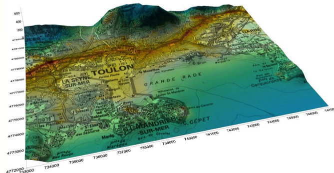
Le contexte

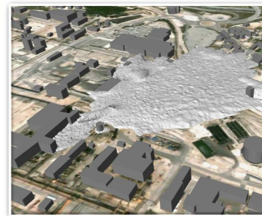
- Post-doc en partenariat avec l'entreprise Numtech
- Calcul des champs météo à l'échelle «locale»
- Calcul de la dispersion des polluants dans l'atmosphère
- Mise en place de systèmes opérationnels : aide à la décision : cas chronique et accidentel



Le contexte

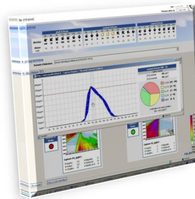
- Post-doc en partenariat avec l'entreprise Numtech
- Calcul des champs météo à l'échelle «locale»
- Calcul de la dispersion des polluants dans l'atmosphère
- Mise en place de systèmes opérationnels : aide à la décision : cas chronique et accidentel

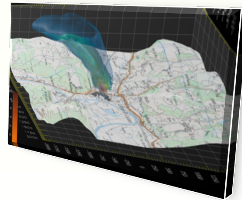




Le contexte

- Post-doc en partenariat avec l'entreprise Numtech
- Calcul des champs météo à l'échelle «locale»
- Calcul de la dispersion des polluants dans l'atmosphère
- Mise en place de systèmes opérationnels : aide à la décision : cas chronique et accidentel

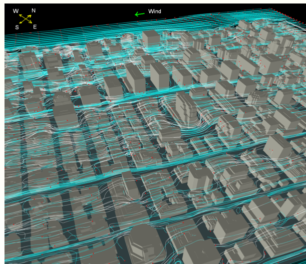


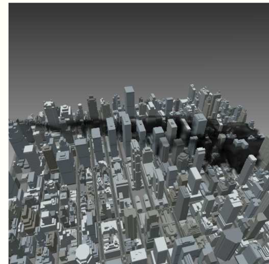


Le contexte

$\Rightarrow$ Besoin de toujours plus de performances

Idée fin 2008 : le calcul sur GPU





Le GPGPU : historique rapide

- Le terme **GPGPU** est apparu en 2002 (Mark Harris) : thèse sur la simulation des nuages pour les jeux vidéo.
- L'avant CUDA :
  - Langages de shader : *GLSL (OpenGL), HLSL (Microsoft DirectX), Cg (Nvidia), Sh, ...*
  - Bibliothèques haut niveau : *BrookGPU, ZippyGPU, GPU++, ...*
- **CUDA** (Compute Unified Device Architecture) est publié en 2007, vraiment utilisé à partir de 2009.
- Aujourd'hui : CUDA 2.3, **OpenCL**, HMPP (CAPS), GreenIT, ...
- Acteurs français : Genci, CCRT, Bull, OpenGPU, GdT, ...
- Demain : Nvidia Fermi, Larabee (Intel), ...

Nouveau type de programmation ?

- Issue du jeu video : gestion de texture, de vertex, travail par bloc, programmation de *shaders*
- Hybride à la manière du Cell (IBM) des PlayStation3
- Proche de l'idéologie vectorielle de NEC ou Cray (SIMD)

Mais pas seulement

- Gestion de milliers de threads : on parle de **streaming**
- Hardware très rapide mais restrictif
- Semble représenter/préparer l'avenir (calculateur hybride)

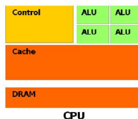
Nouveau type de programmation ?

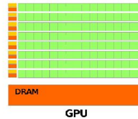
- Issue du jeu video : gestion de texture, de vertex, travail par bloc, programmation de *shaders*
- Hybride à la manière du Cell (IBM) des PlayStation3
- Proche de l'idéologie vectorielle de NEC ou Cray (SIMD)

Mais pas seulement

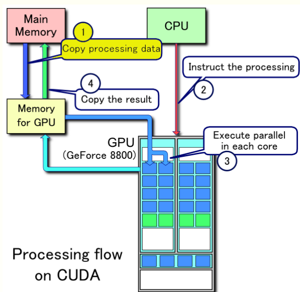
- Gestion de milliers de threads : on parle de **streaming**
- Hardware très rapide mais restrictif
- Semble représenter/préparer l'avenir (calculateur hybride)

$\Rightarrow$ **Pas adapté à tous les besoins !**





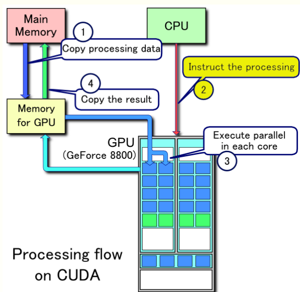
CUDA



- Ecriture de wrappers : transfert de données ① et execution des kernels ②
- Ecriture de kernels ③
- On récupère les données ④
- Interopérabilité avec OpenGL et DirectX (affichage)

```
float *a = new float [256*256];
int Size = 256*256 * sizeof(float);
//allocation de a_Gpu non décrite
cudaMemcpy ( a_Gpu , a, Size , cudaMemcpyHostToDevice );
```

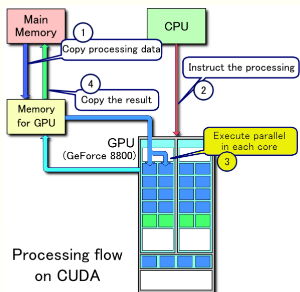
CUDA



- Ecriture de wrappers : transfert de données ① et execution des kernels ②
- Ecriture de kernels ③
- On récupère les données ④
- Interopérabilité avec OpenGL et DirectX (affichage)

```
#define CUDA_BLOCK 256
#define CUDA_GRID 256
kernel_exemple<<<CUDA_GRID,CUDA_BLOCK>>> (a_Gpu, b_Gpu, c);
```

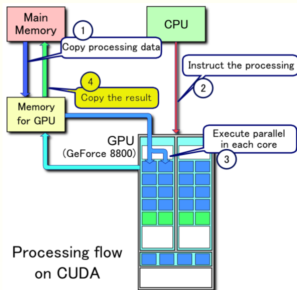
CUDA



- Ecriture de wrappers : transfert de données ① et execution des kernels ②
- Ecriture de kernels ③
- On récupère les données ④
- Interopérabilité avec OpenGL et DirectX (affichage)

```
__global__ kernel_exemple(float *a, float* b, float c){
        k = GET_THREAD_ID(k);
        b[k] = a[k] * c;
}
```

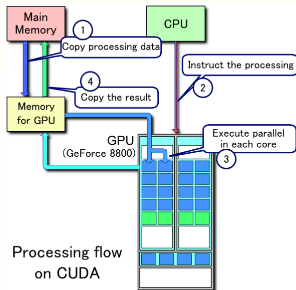
CUDA



- Ecriture de wrappers : transfert de données ① et execution des kernels ②
- Ecriture de kernels ③
- On récupère les données ④
- Interopérabilité avec OpenGL et DirectX (affichage)

```
cudaMemcpy ( a_Gpu , a, Size , cudaMemcpyDeviceToHost );
```

CUDA



- Ecriture de wrappers : transfert de données ① et execution des kernels ②
- Ecriture de kernels ③
- On récupère les données ④
- Interopérabilité avec OpenGL et DirectX (affichage)

Solveur fluide compressible

Pourquoi ce cas test ?

- Equations d'Euler compressibles
- Discrétisation sur maillage structuré
- Méthode volumes finis
- Montée en ordre MUSCL
- Solveur de Riemann HLLC (Harten, Lax, Van Leer, 1983)
- Schéma en temps explicite d'ordre 1
- Initialement pas encore traité/publié !

Solveur fluide compressible

Pourquoi ce cas test ?

- Equations d'Euler compressibles
  - Modélise un écoulement fluide non visqueux
  - **Nombreux de choix** dans les méthodes et les schémas numériques
- Discrétisation sur maillage structuré
- Méthode volumes finis
- Montée en ordre MUSCL
- Solveur de Riemann HLLC (Harten, Lax, Van Leer, 1983)
- Schéma en temps explicite d'ordre 1
- Initialement pas encore traité/publié !

Solveur fluide compressible

Pourquoi ce cas test ?

- Equations d'Euler compressibles
- Discrétisation sur maillage structuré
  - Cas simple (**accès mémoire rapide**) d'abord 2D, puis 3D ...
  - Mais codé pour faciliter le passage au non structuré
- Méthode volumes finis
- Montée en ordre MUSCL
- Solveur de Riemann HLLC (Harten, Lax, Van Leer, 1983)
- Schéma en temps explicite d'ordre 1
- Initialement pas encore traité/publié !

Solveur fluide compressible

Pourquoi ce cas test ?

- Equations d'Euler compressibles
- Discrétisation sur maillage structuré
- Méthode volumes finis
  - Méthode assez «**locale**» (meilleurs accès mémoire)
  - Peu de changement pour le 3D et/ou le non-structuré
- Montée en ordre MUSCL
- Solveur de Riemann HLLC (Harten, Lax, Van Leer, 1983)
- Schéma en temps explicite d'ordre 1
- Initialement pas encore traité/publié !

Solveur fluide compressible

Pourquoi ce cas test ?

- Equations d'Euler compressibles
- Discrétisation sur maillage structuré
- Méthode volumes finis
- Montée en ordre MUSCL
  - Méthode connue (L^∞ stability of the MUSCL methods, *Stéphane Clain, Vivien Clauzon*)
  - Conserve une bonne **localité**
- Solveur de Riemann HLLC (Harten, Lax, Van Leer, 1983)
- Schéma en temps explicite d'ordre 1
- Initialement pas encore traité/publié !

Solveur fluide compressible

Pourquoi ce cas test ?

- Equations d'Euler compressibles
- Discrétisation sur maillage structuré
- Méthode volumes finis
- Montée en ordre MUSCL
- Solveur de Riemann HLLC (Harten, Lax, Van Leer, 1983)
    - Bon compromis robustesse/précision
    - Bonne **intensité arithmétique** sur des opérations «simples» (pas trop de sqrt, min, max, /, if, . . .)
- Schéma en temps explicite d'ordre 1
- Initialement pas encore traité/publié !

Solveur fluide compressible

Pourquoi ce cas test ?

- Equations d'Euler compressibles
- Discrétisation sur maillage structuré
- Méthode volumes finis
- Montée en ordre MUSCL
- Solveur de Riemann HLLC (Harten, Lax, Van Leer, 1983)
- Schéma en temps explicite d'ordre 1
  - La question explicite versus implicite est importante en GPU !
  - Schéma simple **peu consommateur en mémoire** (souvent moins de 1Go sur les cartes vidéo ! ! !)
- Initialement pas encore traité/publié !

Solveur fluide compressible

Pourquoi ce cas test ?

- Equations d'Euler compressibles
- Discrétisation sur maillage structuré
- Méthode volumes finis
- Montée en ordre MUSCL
- Solveur de Riemann HLLC (Harten, Lax, Van Leer, 1983)
- Schéma en temps explicite d'ordre 1
- Initialement pas encore traité/publié !

  Beaucoup de travaux en incompressible sur la base du GPUGems ou autre (semi lagrangien, QUICK, . . .)

Solveur fluide compressible

Pourquoi ce cas test ?

- Equations d'Euler compressibles
- Discrétisation sur maillage structuré
- Méthode volumes finis
- Montée en ordre MUSCL
- Solveur de Riemann HLLC (Harten, Lax, Van Leer, 1983)
- Schéma en temps explicite d'ordre 1
- Initialement pas encore traité/publié !

  Initialement **peu de travaux en compressible** . . .puis
  - Euler 2D/3D en dimensionnal splitting (T. Brandvik, G. Pullan ou F. Château, R. Teyssier), 2009
  - NSSUS (Stanford NS solver) (E. Elsen, P. Legresley, E. Darve), JCP sept. 2008
  - NHD, Godunov solver, 3D, MPI (Takayuki Muranushi), sept. 2009

Position du problème

Dans un domaine $\Omega \subset \mathbb{R}^3$, on cherche à résoudre l'équation de transport

$$\begin{aligned} &\forall t > 0,\ \forall \boldsymbol{x} \in \Omega, \qquad && \frac{\partial u}{\partial t}(\boldsymbol{x}, t) + \mathsf{div}\big(\boldsymbol{F}(u(\boldsymbol{x}, t))\big) = 0, \\ &\forall \boldsymbol{x} \in \Omega, && u(\boldsymbol{x}, 0) = U_0(\boldsymbol{x}). \end{aligned}$$

On cherche $U_i(t)$ une approximation de la moyenne de l'inconnue sur K_i

Forme semi-discrète

$$|K_i| \underbrace{\frac{dU_i}{dt}(t)}_{\text{valeurs aux éléments}} = - \sum_{j \in \mathscr{V}(i)} |S_{ij}|\, F_{ij} \underbrace{\left(U_{ij}(t), U_{ji}(t)\right)}_{\text{valeurs aux faces}}.$$

Position du problème

On cherche $U_i(t)$ une approximation de la moyenne de l'inconnue sur K_i

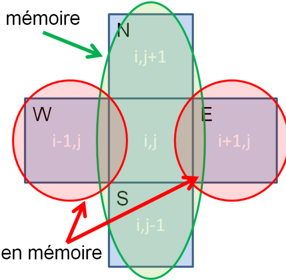
Forme semi-discrète

$$|K_i| \underbrace{\frac{dU_i}{dt}(t)}_{\text{valeurs aux éléments}} = - \sum_{j \in \mathscr{V}(i)} |S_{ij}| \, F_{ij} \underbrace{\left(U_{ij}(t), U_{ji}(t) \right)}_{\text{valeurs aux faces}}.$$

- $\dfrac{dU_i}{dt}(t)$: discrétisé par un **schéma en temps** d'ordre 1.
- $U_{ij}(t)$: valeur sur les interfaces par la **méthode MUSCL**.
- F_{ij} : **flux numérique** à calculer (solveur de Riemann ici).

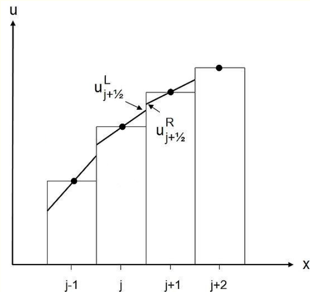
Méthode MUSCL

Pour chaque direction (N,S,E,W), reconstruction de pentes amont et aval (pas de dimensionnal splitting, on prépare l'extension en non structuré), puis limitation de pente (*minmod*)



$$\text{minmod}(r) = \begin{cases} \text{si } r \leq 0, & 0. \\ \text{sinon} & \min(1, r) \end{cases}$$

Pour chaque direction (N,S,E,W), reconstruction de pentes amont et aval (pas de dimensionnal splitting, on prépare l'extension en non structuré), puis limitation de pente (*minmod*)

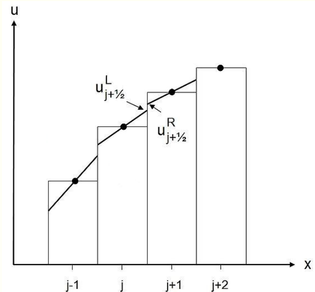


$$p^- = \frac{u_j - u_{j-1}}{\Delta} \qquad p^+ = \frac{u_{j+1} - u_j}{\Delta} \qquad p^{++} = \frac{u_{j+2} - u_{j+1}}{\Delta}$$

$$u^L_{j+1/2} = u_j + \frac{\Delta}{2} p^+ \text{minmod}\left(\frac{p^-}{p^+}\right)$$

$$u^R_{j+1/2} = u_{j+1} - \frac{\Delta}{2} p^+ \text{minmod}\left(\frac{p^+}{p^{++}}\right)$$

Pour chaque direction (N,S,E,W), reconstruction de pentes amont et aval (pas de dimensionnal splitting, on prépare l'extension en non structuré), puis limitation de pente (*minmod*)



Première fonction «device»

```
__device__ float Minmod(float slopeUp,float slopeDown)
{
        if((slopeUp * slopeDown<=0.))
                return(0.);
        float r=fabs(slopeUp)/fabs(slopeDown);
        return(min(r,ONE));
}
```

Equations d'Euler

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

- 4 équations en 2D (5 équations en 3D) + loi de fermeture :
  - Équation de continuité (conservation de la masse)
  - 2 équations du mouvement (3 équations en 3D)
  - Équation d'énergie (température)
  - Équation des gaz parfaits

- Domaine d'application :
  - Écoulement rapide dans l'air (Mach > 0.5)
  - Propagation d'onde (explosion, accoustique)

Sous forme conservative, le vecteur inconnu est $\boldsymbol{U} = {}^t(\rho, \rho u, \rho v, \rho w, E)$

$$\frac{\partial \boldsymbol{U}}{\partial t} + \frac{\partial \boldsymbol{F}(\boldsymbol{U})}{\partial x} + \frac{\partial \boldsymbol{G}(\boldsymbol{U})}{\partial y} + \frac{\partial \boldsymbol{H}(\boldsymbol{U})}{\partial z} = 0.$$

Les flux associés sont alors définis par

$$\boldsymbol{F}(\boldsymbol{U}) = \begin{pmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ \rho uw \\ u(E+P) \end{pmatrix}, \qquad \boldsymbol{G}(\boldsymbol{U}) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ \rho vw \\ v(E+P) \end{pmatrix},$$

$$\boldsymbol{H}(\boldsymbol{U}) = \begin{pmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + P \\ w(E+P) \end{pmatrix}.$$

Solveur HLLC

Bonne intensité arithmétique

- 89 Flop
- 12 variables d'entrées, 6 variables de sortie

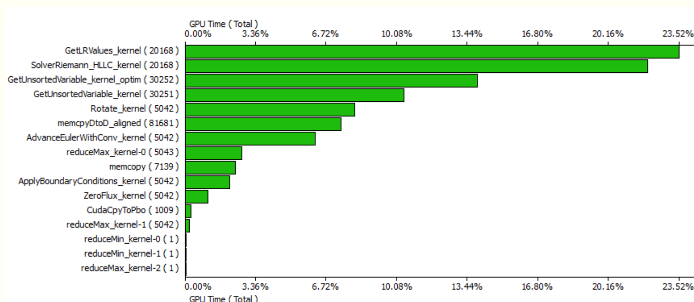
Gourmand en registres

- 25 variables temporaires : 22 registres utilisés
- Occupation du GPU de seulement 33%
- Difficile de faire mieux . . .

Robuste et rapide

- Peu de divisions (7)
- Un seul test (if), équilibré

Solveur HLLC



- MUSCL : 23.5%
- HLLC : 22%
- Schéma en temps : 5%
- Recherche de max : 3%
- BC : 2.5%
- Copies mémoire : 36%

Etapes de l'algorithme

$$\underbrace{\boldsymbol{U}_i^n = (\rho_i^n, \rho_i^n \boldsymbol{V}_i^n, E_i^n)}_{\text{Calcul des flux, schéma en temps}} \iff \underbrace{\boldsymbol{W}_i^n = (\rho_i^n, \boldsymbol{V}_i^n, P_i^n)}_{\text{Méthode MUSCL}}$$

1 - Méthode multipente $\quad \boldsymbol{W}_i^n \to \boldsymbol{W}_{i,j}^n$

2 - Calcul des flux $\quad \boldsymbol{W}_{i,j}^n \to \boldsymbol{F}_{ij}$

3 - Changement de variables $\quad \boldsymbol{W}_{i,j}^n \to \boldsymbol{U}_{i,j}^n$

4 - Schéma en temps $\quad \boldsymbol{U}_{i,j}^n \;,\; \boldsymbol{F}_{ij} \to \boldsymbol{U}_i^{n+1}$

5 - Changement de variables $\quad \boldsymbol{U}_{i,j}^n \to \boldsymbol{W}_{i,j}^n$

Etapes de l'algorithme

$$\underbrace{\boldsymbol{U}_i^n = (\rho_i^n, \rho_i^n \boldsymbol{V}_i^n, E_i^n)}_{\text{Calcul des flux, schéma en temps}} \Longleftrightarrow \underbrace{\boldsymbol{W}_i^n = (\rho_i^n, \boldsymbol{V}_i^n, P_i^n)}_{\text{Méthode MUSCL}}$$

1 - Méthode multipente $\boldsymbol{W}_i^n \to \boldsymbol{W}_{i,j}^n$

2 - Calcul des flux $\boldsymbol{W}_{i,j}^n \to \boldsymbol{F}_{ij}$

3 - **Changement de variables** $\boldsymbol{W}_{i,j}^n \to \boldsymbol{U}_{i,j}^n$

4 - **Schéma en temps** $\boldsymbol{U}_{i,j}^n \; , \; \boldsymbol{F}_{ij} \to \boldsymbol{U}_i^{n+1}$

5 - **Changement de variables** $\boldsymbol{U}_{i,j}^n \to \boldsymbol{W}_{i,j}^n$

Premier kernel (étapes 3,4 et 5)

```
__global__ void AdvanceEulerWithConv_kernel(...){
        k = GET_THREAD_ID(k);
        //Prim to consv :: P=E and U=rho*U
        P[k] = P[k]*oogmo + HALF*Rho[k]*(U[k]*U[k]+V[k]*V[k]);
        U[k] = Rho[k]*U[k];
        V[k] = Rho[k]*V[k];
        //Add flux
        Rho[k] -= A*Fr  [k];
        U[k]   -= A*Fru[k];
        V[k]   -= A*Frv[k];
        P[k]   -= A*FE  [k];
        //Consv to prim :: back to "normal"
        float tmp = ONE/Rho[k];
        U[k] = U[k]*tmp;
        V[k] = V[k]*tmp;
        P[k] = (P[k] - HALF*Rho[k]*(U[k]*U[k]+V[k]*V[k])) * gmo;
}
//calling line in main cpp file
AdvanceEulerWithConv_kernel<<<CUDA_GRID,CUDA_BLOCK>>>(...);
```

Les principales difficultés sur GPU :

- *Les algorithmes*
- *Les transferts mémoires*
- *Les accès mémoire non alignés*
- *L'écriture des fichiers de sorties*
- *Le nombre de registres par thread*

Les principales difficultés sur GPU :

- *Les algorithmes*

  Problème : tous les codes ne sont pas facilement portable sur GPU !
  Solution : ils faut parfois changer complètement d'algorithme

- *Les transferts mémoires*
- *Les accès mémoire non alignés*
- *L'écriture des fichiers de sorties*
- *Le nombre de registres par thread*

Les principales difficultés sur GPU :

- *Les algorithmes*
- *Les transferts mémoires*

  Problème : lents, à éviter ! (à moins qu'ils soient asynchrones)
  Solution : compromis (tout le calcul sur GPU, même les parties «scalaires»)
- *Les accès mémoire non alignés*
- *L'écriture des fichiers de sorties*
- *Le nombre de registres par thread*

Les principales difficultés sur GPU :

- *Les algorithmes*
- *Les transferts mémoires*
- *Les accès mémoire non alignés*

  Problème : lents ! (parfois 25 fois plus lent)
  Solution : utiliser la *shared memory* (mémoire pour un bloc de thread)
- *L'écriture des fichiers de sorties*
- *Le nombre de registres par thread*

Les principales difficultés sur GPU :

- *Les algorithmes*
- *Les transferts mémoires*
- *Les accès mémoire non alignés*
- *L'écriture des fichiers de sorties*

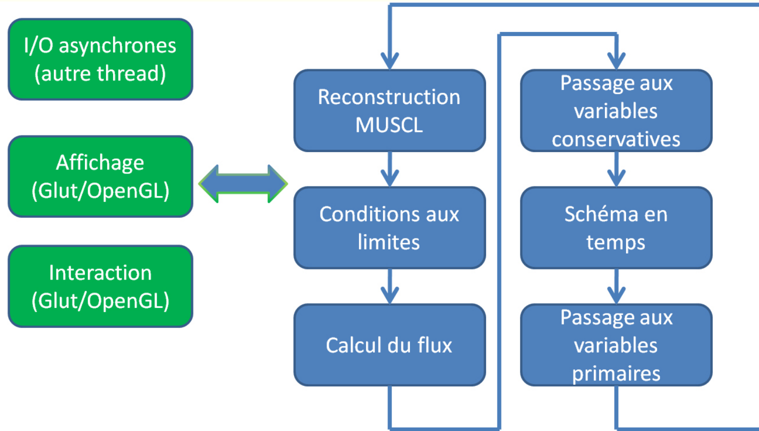
  Problème : extrêmement lent par rapport au calcul, à faire en asynchrone le plus possible
  Solution : application *multi-threadée*, écriture non-bloquante

- *Le nombre de registres par thread*

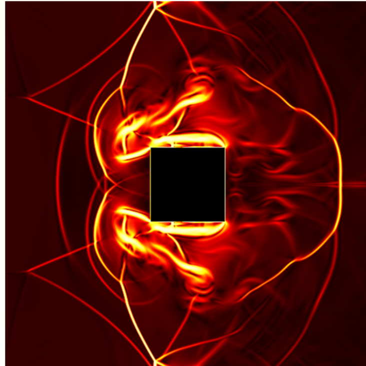
Les principales difficultés sur GPU :

- *Les algorithmes*
- *Les transferts mémoires*
- *Les accès mémoire non alignés*
- *L'écriture des fichiers de sorties*
- *Le nombre de registres par thread*

  Problème : utilisation réduite des ressources GPU sur les kernels trop «compliqués»
  Solution : découper les kernels

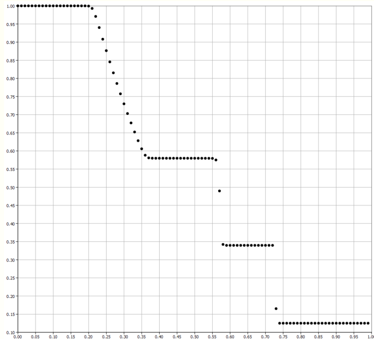


Sur CPU **sur GPU**



... Démonstration ...

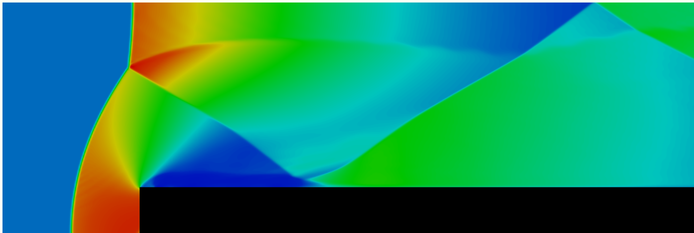
Tubes à chocs



(Toro), 1024 points suivant x, 2D

(1 point sur 10 affiché)

Marche montante



Marche montante, 512×512 points,

3 minutes sur GPU (3h sur CPU ...).

Comparer le comparable

L'utilisation parallèle

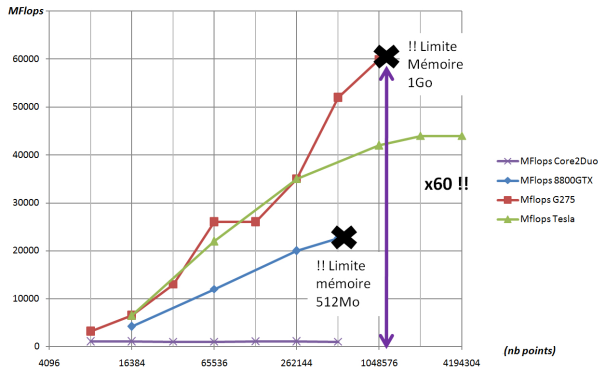
Un CPU n'a pas qu'un coeur (un GPU actuel en a 256 ...)
Il faut donc **exécuter le programme en parallèle** sur CPU (à défaut diviser par 4 le temps de calcul pour un quad-core !)

Le type de flottant

Simple précision sur GPU : il faut utiliser la **simple précision sur CPU** aussi ! Les transferts mémoire sont alors presque deux fois plus rapide !

Effort de programmation

Coder sur GPU n'est pas «simple», il faut donc comparer à un programme **«optimisé» sur CPU** (SSE en particulier)



«Il faut donner à manger à la carte graphique !»

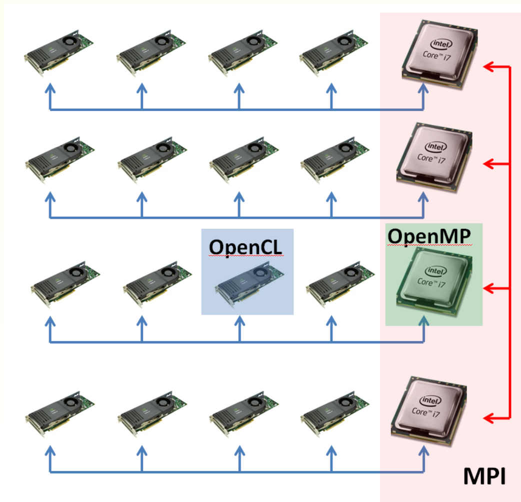
Pas assez optimisé en mémoire

- 130 floats (et 21 ints) : 604 octets / point ($1024 \times 1024 \Rightarrow 604$ Mo)
- Objectifs : moins de 64 floats

Perspectives

- Parallélisation
  - 2 cartes, «à la main» (même machine)
  - n cartes, MPI (différentes machines)
- Non structuré
  - Problèmes d'accès mémoire ? Préserver la localité !
  - Structure de donnée : travailler par petits paquets (une cellule et ces voisins)
- Diffusion (NS)
- Interface de visualisation pour le cas 3D
- Particules, scalaire passif

Perspectives



Merci