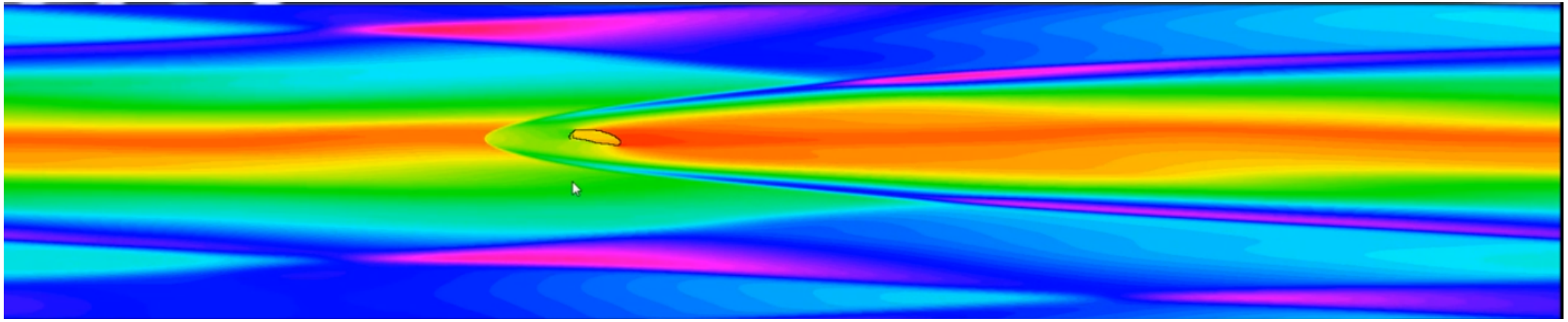


Quelques exemples de calculs instantanés de fluides sur GPU

Christophe Labourdette, Florian De Vuyst

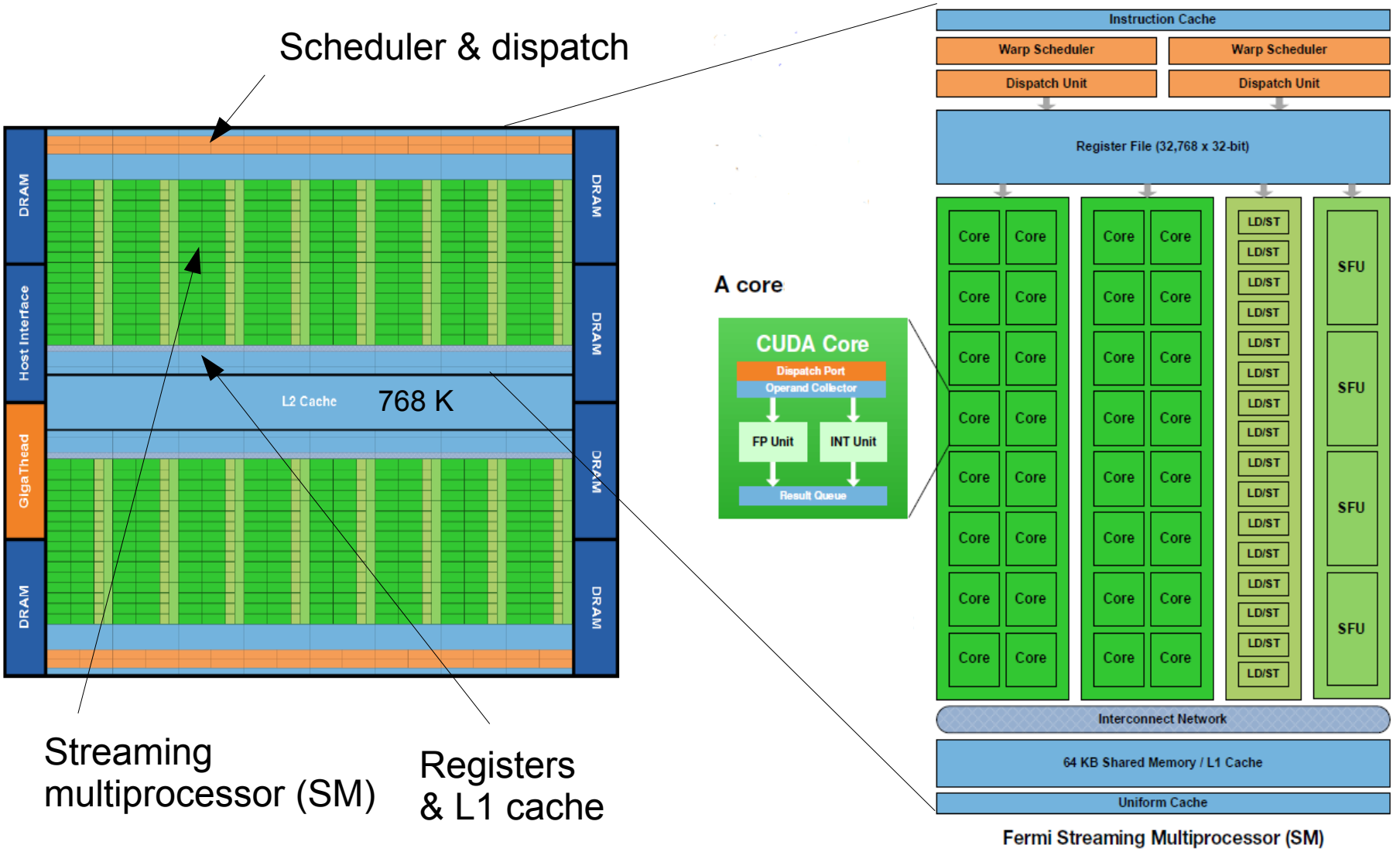
Centre de Mathématiques et de leurs Applications,
CNRS UMR 8536, ENS Cachan

Contact : labour@cmla.ens-cachan.fr

Outline

1. Lattice Boltzmann method for 2D unsteady Navier-Stokes equations
2. Finite Volume method for 2D compressible Euler equations
3. Particle method for 3D kinetic free transport equation on moving domains
4. Experience feedback
5. Future works

nVIDIA GPU FERMI compute architecture



(2*16)*16 = 512 cores

[NVIDIA FERMI compute architecture white paper, 2011]

Hardware in use at CMLA

- PC Workstation with nVIDIA TESLA C2070 (nVIDIA Equipment Grant 2011)

448 cores, FERMI
6 GB Mem
250W

512 GFlops Double Precision (peak)
1,03 TFlops Single Precision (peak)

- Gamer PC Dell Alienware, nVIDIA GTX 580M

384 cores, FERMI
2.0 GB DDR5 Mem

950 GFlops Single Precision (peak)

1. Lattice Boltzmann method for 2D unsteady Navier-Stokes equations

- A mesoscopic paradigm to model PDE problems
- A « smoothed » Lattice Gas automaton model
- Navier-Stokes case : based on a very simple approximation and discretization of the Boltzmann-type equation

Lattice BGK model

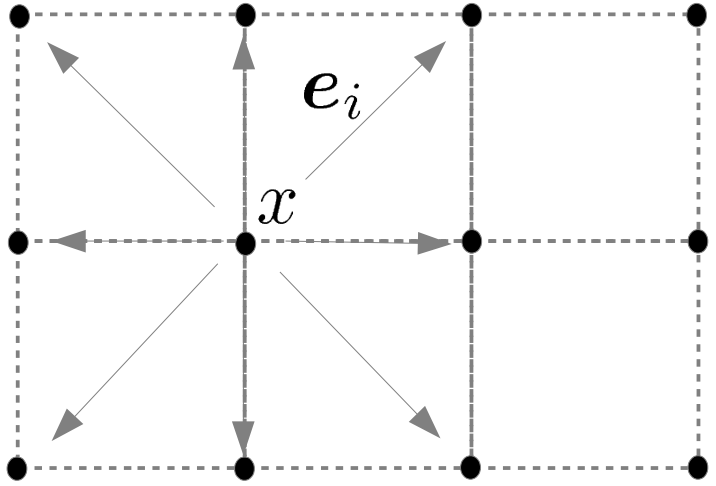
Based on a simple discretization of the Boltzmann equation with BGK approximation for the collision term :

$$f(x + \boldsymbol{e}_i \Delta t, \boldsymbol{e}_i, t + \Delta t) - f(x, \boldsymbol{e}_i, t) = \frac{1}{\tau} \left[f_i^{eq}(\rho, \boldsymbol{u}) - f(x, \boldsymbol{e}_i, t) \right]$$

$$\Delta x = \Delta t = 1$$

Relaxation toward some equilibrium...

$$\boldsymbol{e}_0 = 0, \ \boldsymbol{e}_1 = (1,0), \ \boldsymbol{e}_2 = (1,1), ...$$



Moments:

$$\sum_{i \in \mathcal{S}} f(x, \boldsymbol{e}_i, t) = \rho(x, t),$$

$$\sum_{i \in \mathcal{S}} \boldsymbol{e}_i \, f(x, \boldsymbol{e_i}, t) = \rho \boldsymbol{u}(x, t).$$

Lattice BGK model

- Unknows $f_i(x,t) \equiv f(x, \boldsymbol{e}_i, t),\ i \in \{0, ..., 8\}.$

- LB equation

$$f_i(x + \boldsymbol{e}_i, t + \Delta t) = f_i(x,t) + \omega\ [f_i^{eq}(\rho(x,t), \boldsymbol{u}(x,t)) - f_i(x,t)]$$

- Requirements

$$\omega = \frac{1}{\tau}.$$

$$\sum_{i \in \mathcal{S}} f_i^{eq}(\rho, \boldsymbol{u}) = \rho,$$

$$\sum_{i \in \mathcal{S}} \boldsymbol{e}_i\, f_i^{eq}(\rho, \boldsymbol{u}) = \rho \boldsymbol{u}.$$

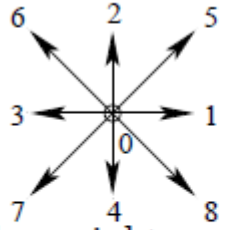
+ some galilean invariance conditions

2D and 3D lattices & equilibrium distribution

D2Q9

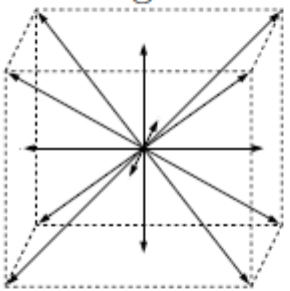
$$f_i^{eq} = \rho\, w_i \left[1 + 3\boldsymbol{u}\cdot\boldsymbol{e}_i + \frac{9}{2}(\boldsymbol{u}\cdot\boldsymbol{e}_i)^2 - \frac{3}{2}\boldsymbol{u}\cdot\boldsymbol{u}\right]$$

For D2Q9 we have



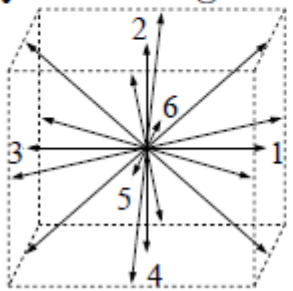
$$w_i = \begin{cases} 4/9 & i = 0 \\ 1/9 & i = 1,2,3,4 \\ 1/36 & i = 5,6,7,8 \end{cases}$$

For D3Q15 the weights are:



$$w_i = \begin{cases} 2/9 & i = 0 \\ 1/9 & i = 1-6 \\ 1/72 & i = 7-14 \end{cases}$$

For D3Q19 the weights are:



$$w_i = \begin{cases} 1/3 & i = 0 \\ 1/18 & i = 1-6 \\ 1/36 & i = 7-18 \end{cases}$$

Source : [A. Wagner 08]

Practical implementation « stream-and-collide »

1. Stream step

$$\tilde{f}_i(x,t) = f_i(x + \boldsymbol{e}_i, t)$$

2. Collision step

- Compute the moments

$$(\rho, \rho\boldsymbol{u})(x,t) = \sum_{i \in \mathcal{S}} (1, \boldsymbol{e}_i)\, \tilde{f}_i(x,t)$$

- Compute the equilibrium function

$$\tilde{f}_i^{eq} = \rho\, w_i\, [...]$$

- Collision step

« ET VOILA ! »

$$f_i(x, t + \Delta t) = \tilde{f}_i(x,t) + \omega \left[\tilde{f}_i^{eq}(\rho(x,t), \boldsymbol{u}(x,t)) - \tilde{f}_i(x,t) \right]$$

Connection with Navier-Stokes equations
(multiscale Chapman-Enskog analysis)

$$\nabla \cdot \boldsymbol{u} = 0 + O(\Delta t^2),$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \frac{1}{\rho}\nabla p - \nu \Delta \boldsymbol{u} = O(\Delta t^2 + \Delta t M^2).$$

with $\boxed{\nu = \frac{1}{3}(2\tau - 1).}$ Requires : $u = M \ll 1.$

Artificial EOS:

$$p = p(\rho) = \rho\, c^2.$$

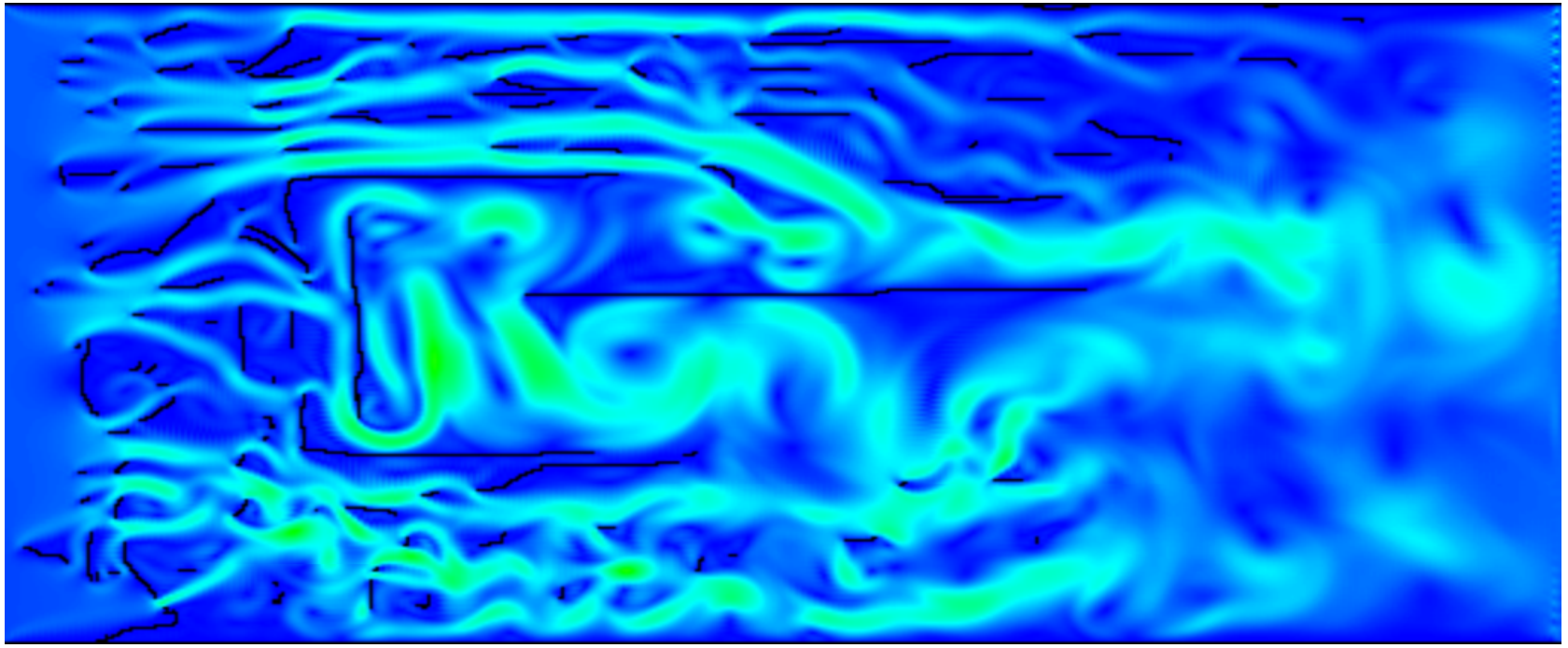
Von Neuman linear stability analysis of LBGK scheme:

$$\tau > \frac{1}{2} \quad \leftrightarrow \quad \omega < 2.$$

Benefits

- Very simple to implement (explicit schemes)
- « O(100-lines code) » Navier-Stokes !
- Wall and fluid BC OK
- Complex geometries by immersed boundary approaches
- Appears to be suitable for GPU parallel computing (stream-and-collide procedure)

[Qian, d'Humières, Lallemand 92], [Ginzburg, d'Humières, Krafczyk, Lallemand, 2002], [Chikatamarla, Ansumali, Karlin 06] [Tosi, Ubertini, Succi 06], [Brownlee, Gorban, Levesley 06], [Xu, Luan, Tang, Tao 09] Finite Volume interpretation [Dubois, Lallemand 08]



< demo >

2. Finite Volume method for 2D compressible Euler equations

$$\partial_t \rho + \nabla \cdot (\rho \boldsymbol{u}) = 0,$$

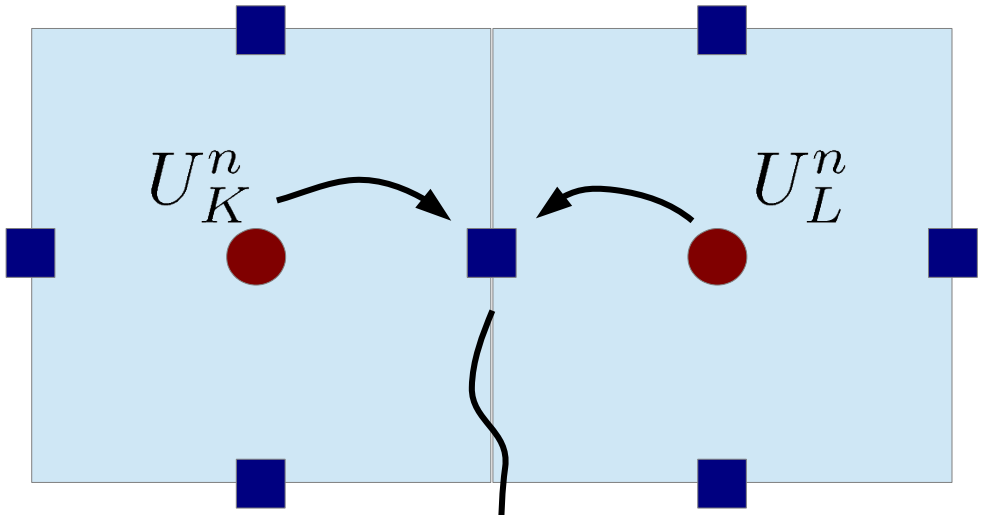
$$\partial_t (\rho \boldsymbol{u}) + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u}) + \nabla p = 0,$$

$$\partial_t (\rho E) + \nabla \cdot ((\rho E + p)\boldsymbol{u}) = 0.$$

$$E = e + \frac{1}{2}|\boldsymbol{u}|^2, \quad e = \frac{1}{\gamma - 1}\frac{p}{\rho}, \ \gamma \in (1, 3].$$

- Nonlinear hyperbolic conservative system
- Solutions generally develop discontinuities
- Numerical framework : (stable) finite volumes

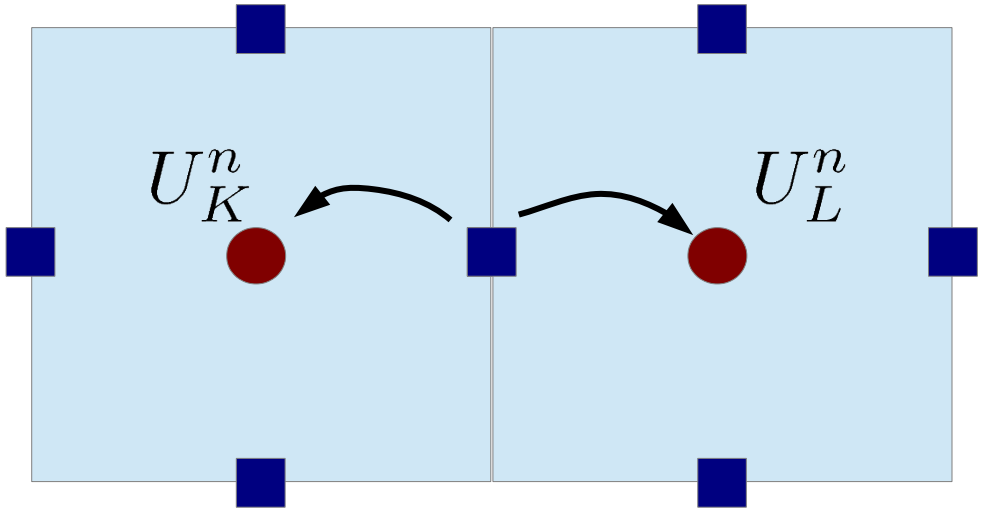
Flux Difference Splitting (FDS) vs Flux Vector Splitting (FVS) methods on GPU



FDS case

i) **Send** states at interfaces
ii) Computes numerical fluxes

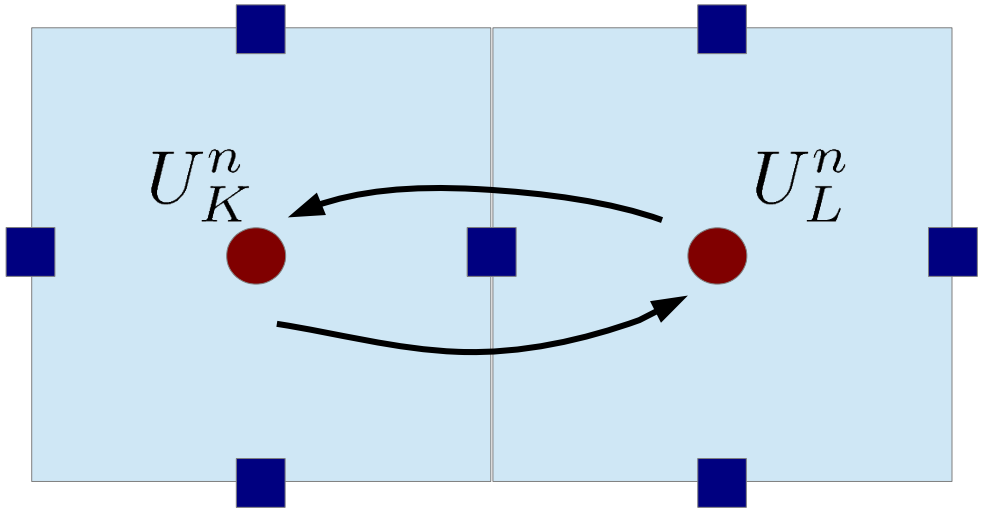
$$\Phi(U_K^n, U_L^n, \nu_{KL}) = \frac{F(U_K^n) + F(U_L^n)}{2} - \frac{1}{2}\int_{\Gamma_{KL}} |A|(s)\, U'(s)\, ds$$



iii) **Send** fluxes at cell centers
iv) Update states

Flux Difference Splitting (FDS) vs Flux Vector Splitting (FVS) methods on GPU

FVS case



$$\Phi(U_K^n, U_L^n, \nu_{KL}) = F^+(U_K^n) + F^-(U_L^n)$$

i) Compute FVS at cell centers

ii) **Send** F+ and F- to the neighboring cells

iii) Update states

→ FVS appears to reduce DRAM communications by 2.

[De Vuyst, Acta Applicandae Mathematicae, 2012, submitted]

Exemple of Flux Vector Splitting

Van Leer's Flux Vector Splitting with Hänel's energy flux correction :

$$F_\rho^+ = \frac{\rho c}{4}(M+1)^2\,1(|M| \le 1) + \max(u,0)\,1(|M|>1)$$

$$p^+ = \frac{p}{4}(M+1)^2(2-M)\,1(|M| \le 1) + p\,1(M>1)$$

$$F_{\rho u}^+ = u\,F_\rho^+ + p^+,$$

$$F_{\rho v}^+ = v\,F_\rho^+,$$

$$F_{\rho E}^+ = H\,F_\rho^+, \quad H = E + p/\rho.$$

NB: very simple to compute.

$$F^-(U) = F(U) - F^+(U).$$

GPU CUDA implementation

CUDA kernel prototypes

```
90  //
91  // CUDA kernel prototypes
92  //
93  __global__ void init_kernel (int pitch, var *d_vars);
94  __global__ void setPlotData_kernel(int pitch, var *d_vars, float *plot_data);
95  __global__ void computePrimitiveVariables_kernel(int pitch, var *d_vars, int *solid_data);
96  __global__ void computeFVSx_kernel(int pitch, var *d_vars, flux *fpp, flux *fmm, int *solid_data);
97  __global__ void computeFVSy_kernel(int pitch, var *d_vars, flux *gpp, flux *gmm, int *solid_data);
98  __global__ void updateScheme_kernel(float hx, float hy, float dt,
99      int pitch, var *d_vars, flux *fpp, flux *fmm, flux *gpp, flux *gmm, int *solid_data);
```

Time advance

Very easy !

```
574     for (int i=0; i<FREQ; i++) {
575     //
576     computePrimitiveVariables_kernel<<<grid, block>>>(pitch, d_vars, solid_data);
577     // Compute FVS in x direction
578     computeFVSx_kernel<<<grid, block>>>(pitch, d_vars, fpp, fmm, solid_data);
579     // Compute FVS in y direction
580     computeFVSy_kernel<<<grid, block>>>(pitch, d_vars, gpp, gmm, solid_data);
581     // Apply the Finite Volume scheme
582     updateScheme_kernel<<<grid, block>>>( hx,  hy,  dt, pitch, d_vars,  fpp, fmm, gpp, gmm, solid_data);
583     //
584     } // for int i
```

Data structures : technical aspects

Coalesced memory is a critical keypoint

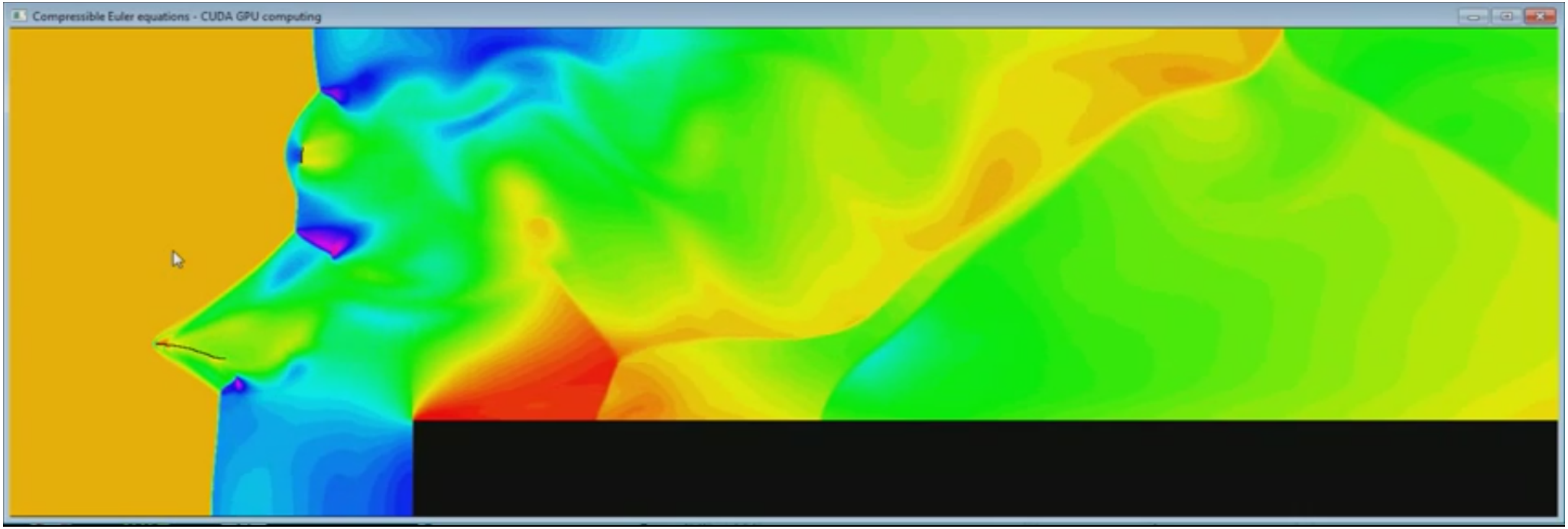
Q: Array-of-Structs (AoS) or Struct-of-ArrayS (SoA) ?

$$U = (\rho,\, \rho u,\, \rho v,\, \rho E)^T$$

```
U[i2d(i,j)].rho = …          // AoS
                             // Coalesced memory for struct

U.rho[i2d(i,j)] = …          // SoA
                             // Coalesced memory for array
```

Option : AoSoA, forced alignment, texture memory, other ?



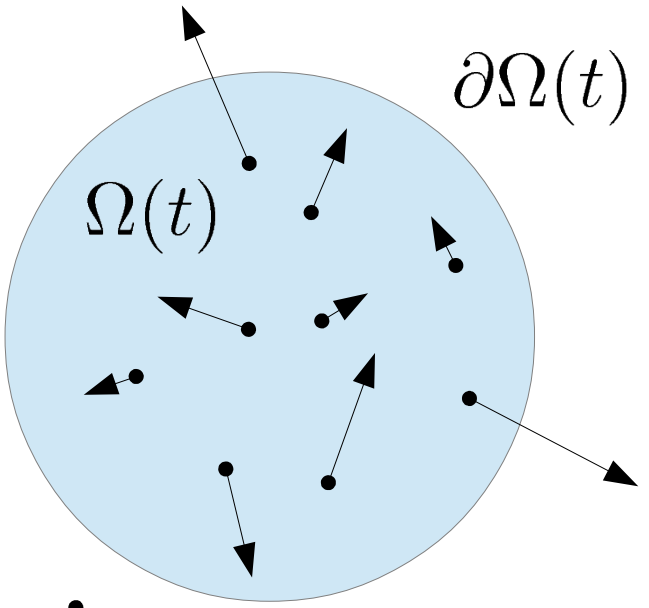
< demo >

3. Particle method for 3D kinetic free transport equation on moving domains

$$\partial_t f + \boldsymbol{v} \cdot \nabla_x f + a(x) \nabla_v f = 0, \quad x \in \Omega(t) \subset \mathbb{R}^3, \ \boldsymbol{v} \in \mathbb{R}^3, \ t > 0.$$

Particle method :

$$\begin{cases} \dot{x}_k(t) = v_k, \\ \dot{v}_k(t) = a(x_k). \end{cases}$$



$$f = \sum_{k=1}^{N} \omega_k \, \delta(x - x_k(t)) \, \delta(v - v_k(t)).$$

$$\|f(t)\|_{L^1(\Omega(t))} = \sum_{k=1}^{N} \omega_k \, 1(x_k \in \Omega(t)).$$

Requires // reduction

Transport equations, results

Hardware : 1 TESLA C2070

Nb of particles N	GPU time (sec)	GPU/CPU speedup factor
2^{17}	0.45	19.6
2^{18}	0.74	23.8
2^{19}	1.31	26.4
$2^{20} = 1,048,576$	2.32	29.7
2^{21}	4.46	31.5
2^{22}	8.89	31.1
2^{23}	18.03	30.7

TABLE 1. Total GPU time computation and speedup factor relatively to a sequential CPU computation.

$\|f(t)\|_{L^1}$

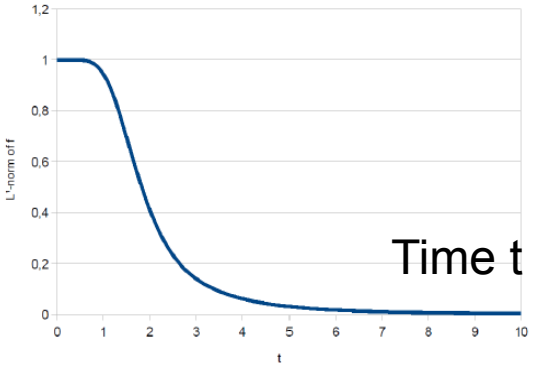


FIGURE 2. Evolution of the L^1-norm of the discrete distribution generated by the initial condition f_2^{in} into the sphere $B_{r(t)}(0)$ (linear scale).

[De Vuyst, Salvarani, submitted to CPC, 2012]

4. Experience feedback for the PDE community

- Reasonable (suboptimal) speedup (say 10 or 20) is easy to reach
- GPU Parallel computing easy for pbs on cartesian grids.
- Better performance : tradeoff to find between code readability and performance.
- Chose suitable data structures for memory coalescence: AoS vs SoA vs SoAoS, alignment ...

5. Future works

- 2D air-water 2-fluid flow with interface capturing on GPU
- Lattice Boltzmann for fluid & moving rigid bodies
- Exploration of new suitable paradigm shifts (kinetic formulation of compressible fluids)
- FVS investigation
- Splitting operator strategies (tuning computations and communications)

– Thank you for your attention –