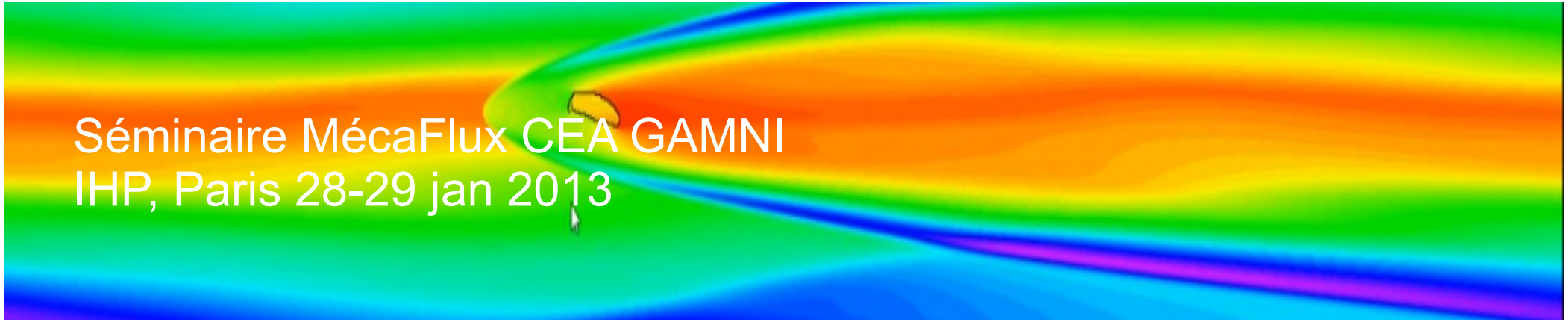


Séminaire MécaFlux CEA GAMNI
IHP, Paris 28-29 jan 2013

Calcul instantané de fluides incompressibles/compressibles sur GPU, visualisation et interactivité temps réel

Florian De Vuyst

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

devuyst@cmla.ens-cachan.fr

Outline

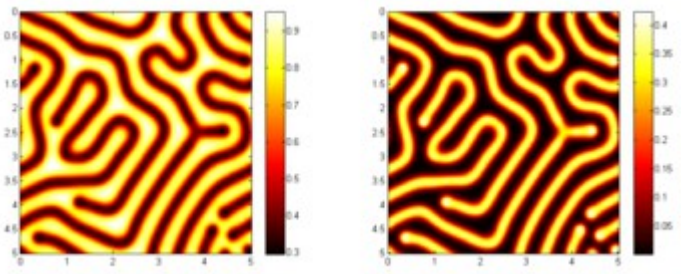
1. GPU: architecture, programming model, performance
2. Suitable numerical methods
3. Ex1 - Lattice Boltzmann LB methods for Navier-Stokes equations
4. Ex2 - Suitable Finite Volume methods (FVM)
5. Ex3 – Remapped-Lagrange Eulerian solvers
6. Foreseen applications
7. DEMO

Goal

Rethink or revisit the numerical methods for CFD in order to get high performances on (General Purpose) Graphics Processing Units (GPU)

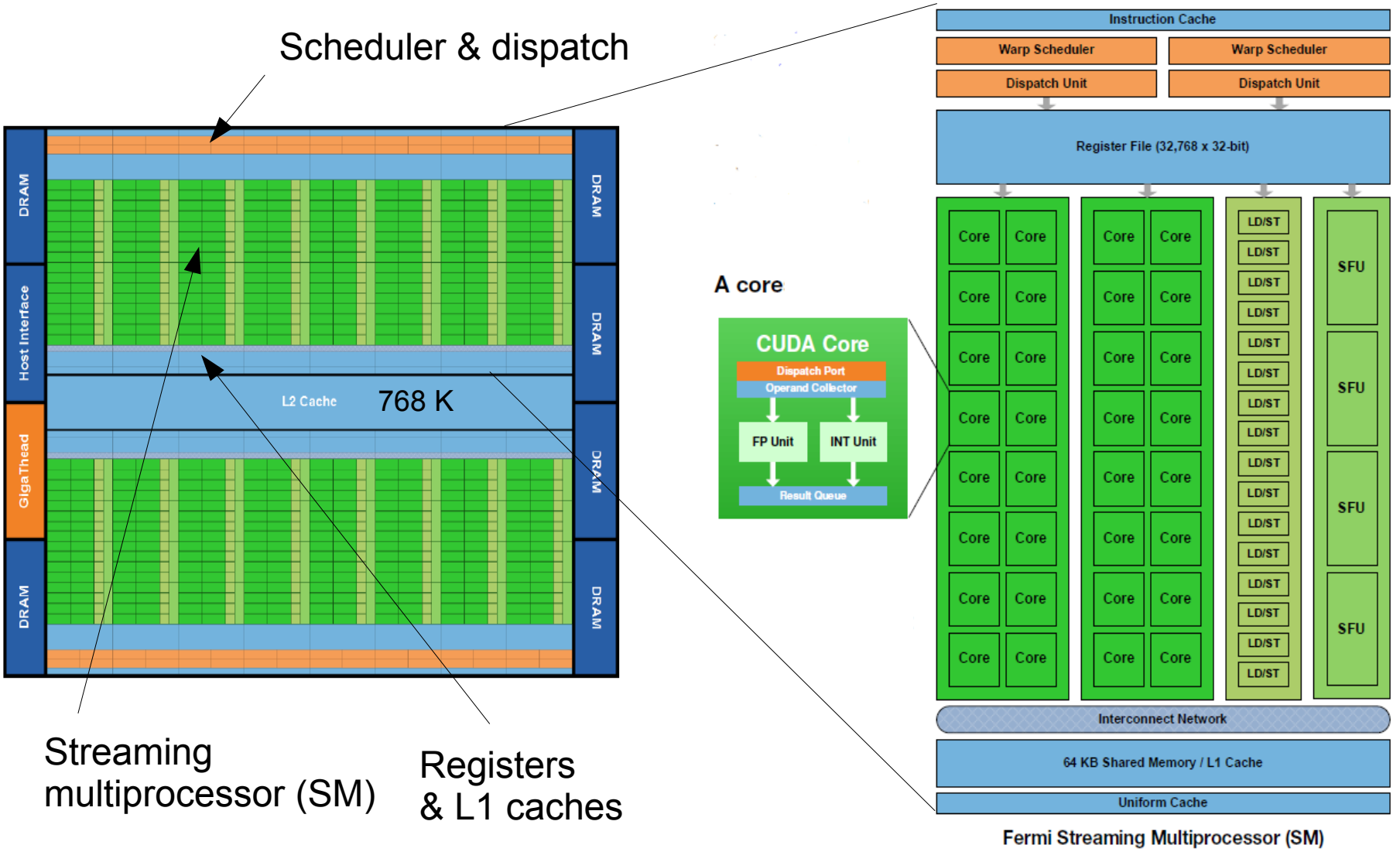
Scope of the talk :

Fluid-like Partial Differential problems on GPU



1. NVIDIA GPU architecture

nVIDIA GPU *FERMI* compute architecture



(2*16)*16 = 512 cores

(schematic from Nvidia)

Recent boards – Kepler family

NVIDIA GeForce GTX 690 (game)

- 2x1536 cores ! 300 W
- 8 streaming multiprocessors (SM)
- Memory bandwidth 192 GB/sec
- MEM 4 GB (2048 MB per GPU)
- DRAM bus memory 512-bit GDDR5
- 2x1.8 Tflops SP, 2x130 Gflops DP

NVIDIA TESLA K20X (HPC)

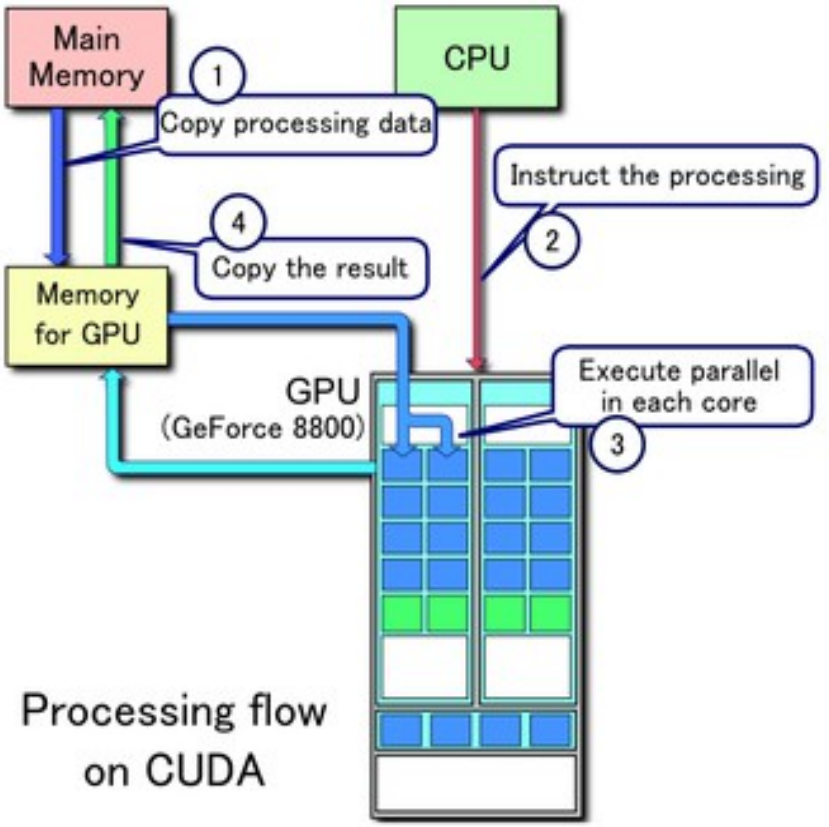
- 2688 cores !
- 14 streaming multiprocessors (SM)
- Memory bandwidth 250 GB/sec
- MEM 6 GB
- 3.95 Tflops SP, 1.31 Tflops DP

http://www.nvidia.com/object/tesla-servers.html

Programming model : different abstraction levels

- Different APIs :
  - CUDA *thrust* library API
  - CUDA runtime API
  - Driver API

Higher level



- NVIDIA CUDA (Compute Uinified Device Architecture) = parallel computing platform & programming model
- Others : OpenCL, OpenACC

Basics of CUDA language

- Define blocks of threads
- Define the number of threads per blocks
- Define device (GPU) kernel functions from callable by the host (__global__)
- Copy arrays from host-to-device and device-to-host

- __device__ : kernels calls from the device
- __shared__ : shared memory (by multiprocessor)
- Kernel call : `my_kernel <<< nb_blocks, block_size >>> (..., ...);`

Key factors of performance

- Multiprocessor occupancy
- Byte-per-flop ratio (mem bandwidth vs FP operations)
- Memory management : registers, cache, coalesced read/write memory
- Warp divergence : be careful to trees of conditional branches
- Host-device PCIe bus bottleneck
- Load balancing, synchronization

- Data structures : Structures of arrays, arrays of structures, structures of arrrays of structures, accessors, hasch tables, specialized structs (float4) or data structures (textures), ...

NVIDIA tools for occupancy optimization

CUDA GPU Occupancy Calculator

Click Here for detailed instructions on how to use this occupancy calculator.
For more information on NVIDIA CUDA, visit http://developer.nvidia.com/cuda

Just follow steps 1, 2, and 3 below! (or click here for help)

1.) Select Compute Capability (click):	3,5	(Help)
1.b) Select Shared Memory Size Config (bytes)	49152	
2.) Enter your resource usage:		
Threads Per Block	256	(Help)
Registers Per Thread	32	
Shared Memory Per Block (bytes)	4096	

(Don't edit anything below this line)

3.) GPU Occupancy Data is displayed here and in the graphs:		
Active Threads per Multiprocessor	2048	(Help)
Active Warps per Multiprocessor	64	
Active Thread Blocks per Multiprocessor	8	
Occupancy of each Multiprocessor	100%	

Physical Limits for GPU Compute Capability:	3,5
Threads per Warp	32
Warps per Multiprocessor	64
Threads per Multiprocessor	2048
Thread Blocks per Multiprocessor	16
Total # of 32-bit registers per Multiprocessor	65536
Register allocation unit size	256
Register allocation granularity	warp
Registers per Thread	255
Shared Memory per Multiprocessor (bytes)	49152
Shared Memory Allocation unit size	256
Warp allocation granularity	4
Maximum Thread Block Size	1024

Allocated Resources	Per Block	Limit Per SM	= Allocatable Blocks Per SM
Warps (Threads Per Block / Threads Per Warp)	8	64	8
Registers (Warp limit per SM due to per-warp reg cour	8	64	8
Shared Memory (Bytes)	4096	49152	12

Note: SM is an abbreviation for (Streaming) Multiprocessor

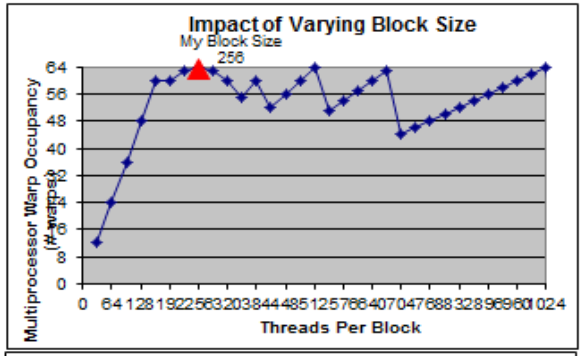
Maximum Thread Blocks Per Multiprocessor	Blocks/SM	* Warps/Block	= Warps/SM
Limited by Max Warps or Max Blocks per Multiproces	8	8	64
Limited by Registers per Multiprocessor	8	8	64
Limited by Shared Memory per Multiprocessor	12		

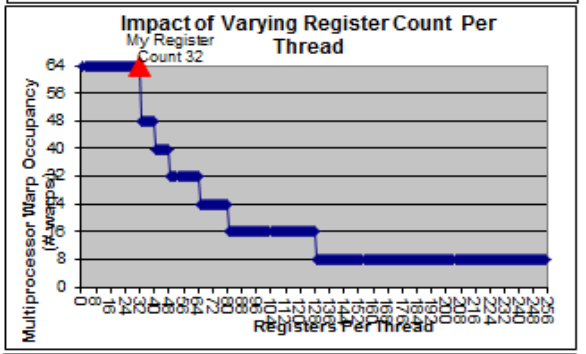
Note: Occupancy limiter is shown in orange

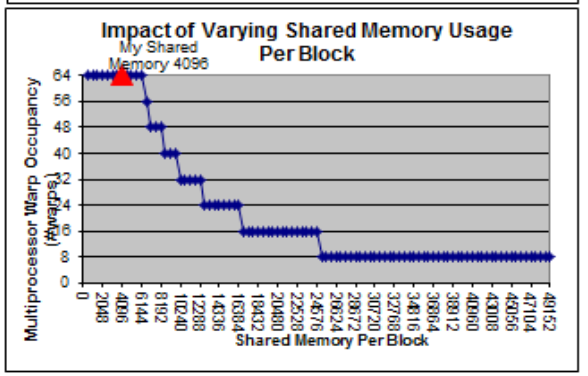
Physical Max Warps/SM = 64
Occupancy = 64 / 64 = 100%

CUDA Occupancy Calculator	
Version:	5,1
Copyright and License	

Your chosen resource usage is indicated by the red triangle on the graphs. The other data points represent the range of possible block sizes, register counts, and shared memory allocation.







Impact on the design of numerical methods

Impact on the choice of numerical methods

- **Cartesian** grids are preferred (grid of blocks, neighboring cell access)
- Time-dependent problems : **explicit** schemes are preferred
- **Compact** spatial stencils
- **Uniform** spatial stencils (memory access patterns)
- A change of **model paradigm** can be useful :
  - Navier-Stokes → Lattice Boltzmann automata
  - Fokker-Planck equations → Underlying stochastic differential problem → particle method, independent trajectories
  - Vlasov equations → transport equations + pdf reduction
  - Reaction-diffusion eqs → SME + probabilistic transition ...

Ex1 - Lattice Boltzmann methods for Navier-Stokes equations

Lattice BGK model

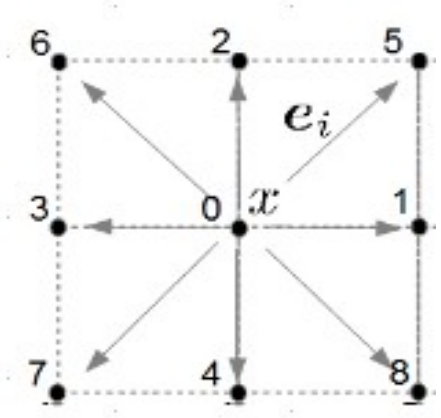
Based on a simple discretization of the Boltzmann equation with BGK approximation for the collision term :

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

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

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

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 :

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

Connection with the incompressible NS equations
(multiscale Chapman-Enskog analysis)

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

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

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

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:

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

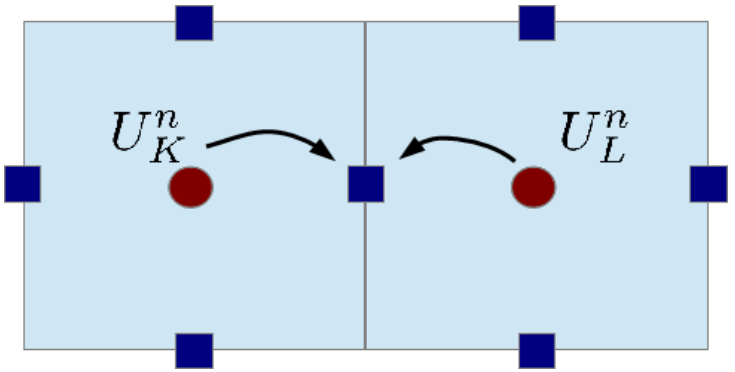
Achievements

- NS solver with runtime visualization and real time interaction (adding some obstacles on the fly …)
- Reasonable time advance on large grids (5120x1440)

NS screen saver
5120x1440

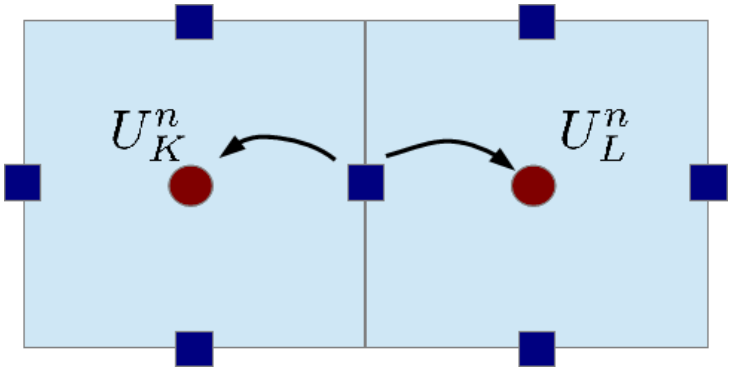
Ex2 - Suitable FV methods

FDS case



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

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

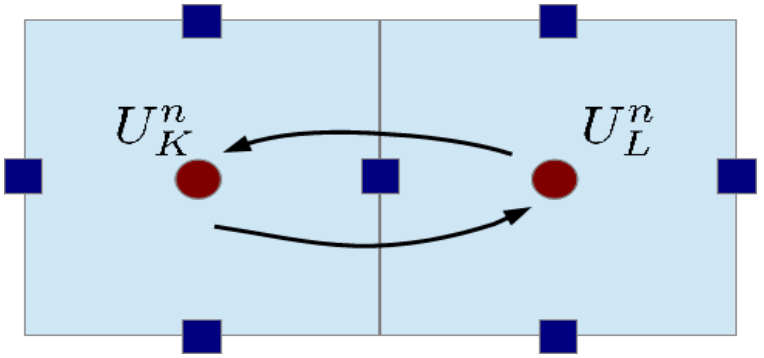


iii) **Send** fluxes at cell centers
iv) Update states

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

FVS case

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



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 Mathematicae Applicandae, 2013, under revision]

Technical aspects

Q: arrays-of-structs (AoS) or structs-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
```

Use of float4 ...

Ex3 – Remapped-Lagrange Eulerian solvers

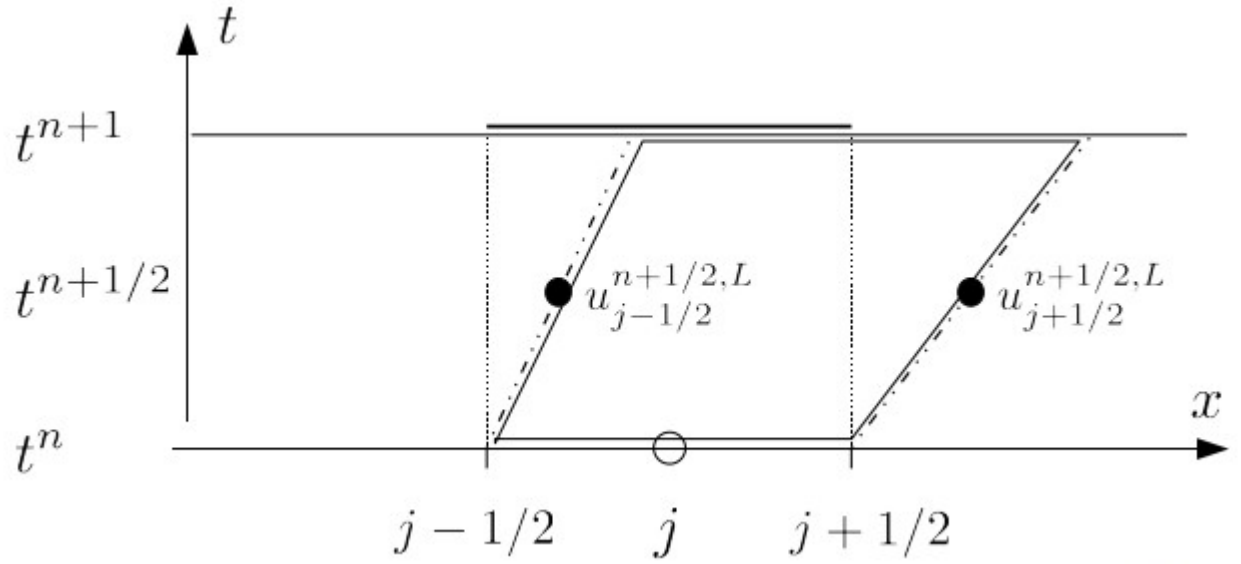
Lagrange-remap (LR) schemes

- Define a piecewise $\mathscr{C}^1$ Lagrange transformation operator $\mathscr{L}(\boldsymbol{x}, t; t_0, \boldsymbol{x}_0) : \ d\boldsymbol{x}/dt = \boldsymbol{u}(\boldsymbol{x}), \ \boldsymbol{x}(t=0) = \boldsymbol{x}_0$.
- Use the Reynolds transport theorem:

$$\frac{d}{dt}\int_{\Omega_t} q(\boldsymbol{x}, t)\, d\boldsymbol{x} = \int_{\Omega_t} \{\partial_t q + \nabla \cdot (q\boldsymbol{u})\}\, d\boldsymbol{x}$$

  for any moving domain Ω_t, quantity q.
- Finite volume approximation
- Move the mesh according to a Lagrange operator $\mathscr{L}$ over $[t^n, t^{n+1})$
- Project the quantities on the initial mesh.

Conservative reformulation



Geometric conservation law (GCL):

$$h_j^{n+1,L} = h + \Delta t\,(u_{j+1/2}^{n+1/2,L} - u_{j-1/2}^{n+1/2,L}).$$

Mass conservation law:

$$\rho_j^{n+1,L} h_j^{n+1,L} = h\rho_j^n.$$

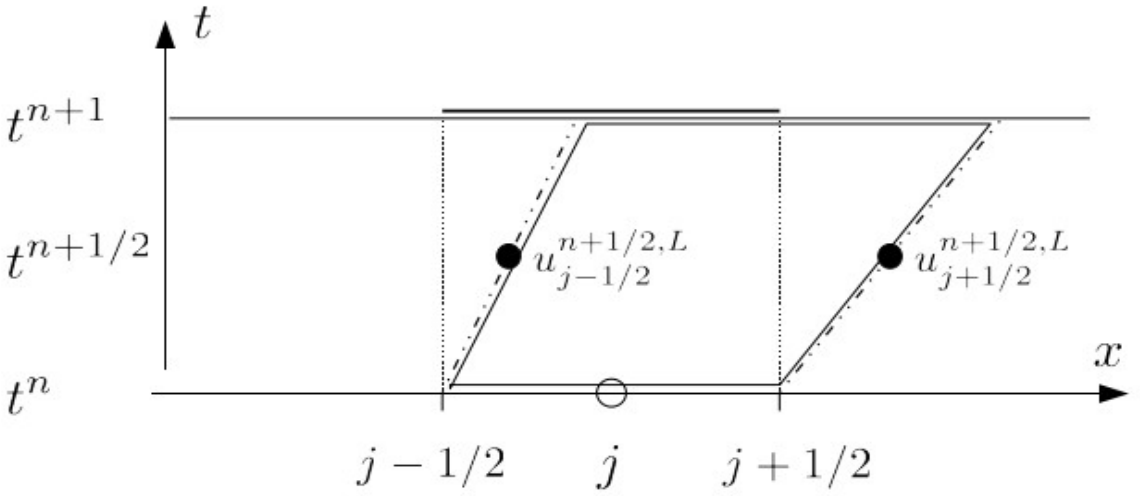
Conservative reformulation (2)

i.e.

$$\rho_j^{n+1,L} = \frac{\rho_j^n}{1 + \frac{\Delta t}{h}(\Delta u)_j^{n+1/2,L}}, \quad (\Delta u)_j^{n+1/2,L} := u_{j+1/2}^{n+1/2,L} - u_{j-1/2}^{n+1/2,L}.$$

Projection step:

$$\rho_j^{n+1} = \frac{1}{h}\int_{I_j} \mathscr{I}\rho^{n+1,L}(x)\,dx = \frac{1}{h}\int_{I_j^{n+1,L}} \ldots - \ldots + \ldots .$$



Conservative reformulation (3)

Mass balance rewritting : under some convenient CFL condition, we have

$$\begin{aligned} h\rho_j^{n+1} &= h_j^{n+1,L}\rho_j^{n+1,L} - \Delta t \rho_{j+1/2}^{upw,n+1} u_{j+1/2}^{n+1/2,L} + \Delta t \rho_{j+1/2}^{upw,n+1} u_{j-1/2}^{n+1/2,L} \\ &= h\rho_j^n - \Delta t \rho_{j+1/2}^{upw,n+1} u_{j+1/2}^{n+1/2,L} + \Delta t \rho_{j+1/2}^{upw,n+1} u_{j-1/2}^{n+1/2,L} \end{aligned}$$

in the form

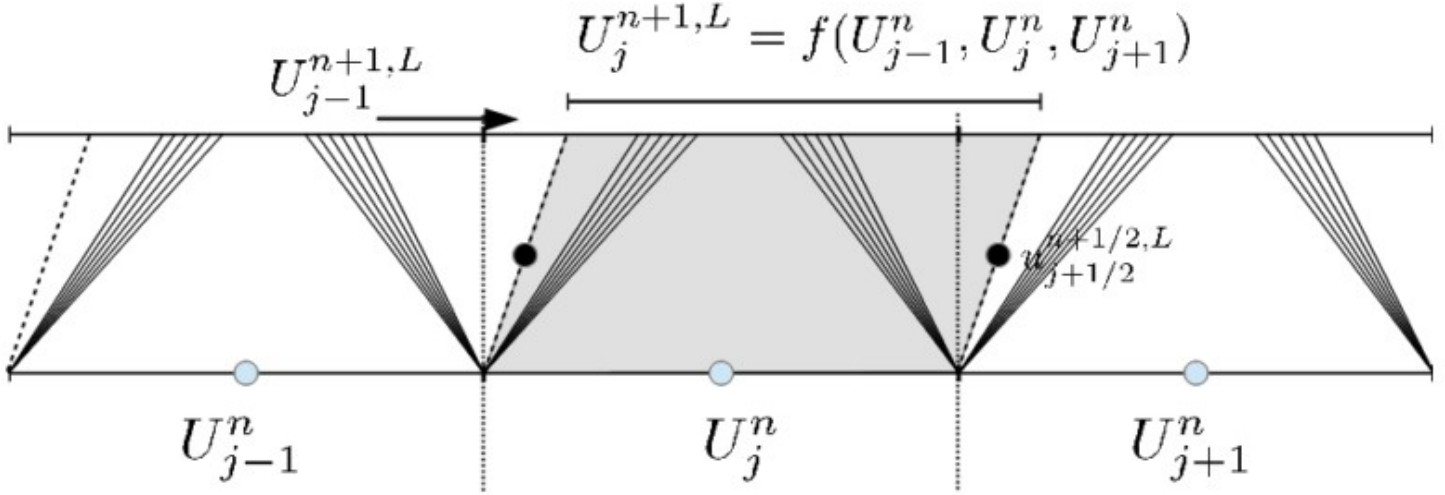
$$\rho_j^{n+1} = \rho_j^n - \frac{\Delta t}{h}\left(\Phi_{m,j+1/2}^{n+1/2,n+1} - \Phi_{m,j-1/2}^{n+1/2,n+1}\right),$$

$$\Phi_{m,j+1/2}^{n+1/2,n+1} = \rho_{j+1/2}^{upw,n+1} u_{j+1/2}^{n+1/2,L}.$$

[De Vuyst, Fochesato, Loubère, Saas, Motte, Ghidaglia, preprint paper 2013]

Remark

First-order (1st-order remap) LR schemes are actually 5-point schemes !

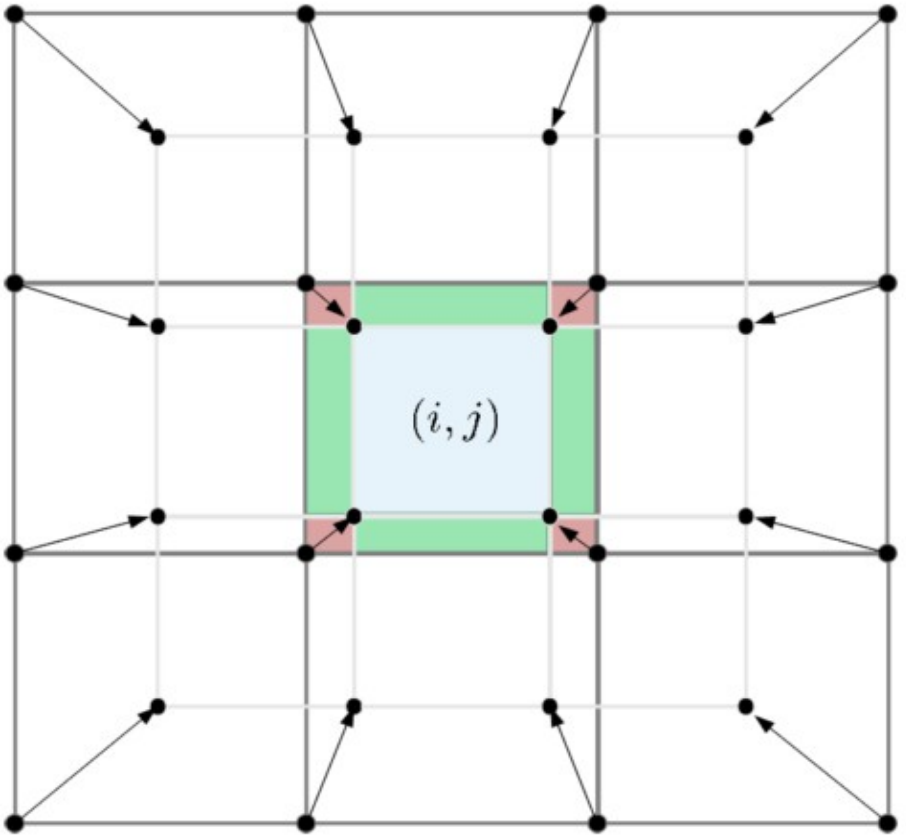


$$U_j^{n+1} = U_j^n - \frac{\Delta t}{h}\left(\Phi_{j+1/2}^{n,n+1} - \Phi_{j-1/2}^{n,n+1}\right),$$

$$\Phi_{j+1/2}^{n,n+1} = \Phi\left(U_{j-1}^n, U_j^n, U_{j+1}^n, U_{j+2}^n, \Delta t\right).$$

→ Large stencil method : limited GPU performance because of lot of memory reads.

2D case



⇒ 1st order LR is a 25-point scheme !
Multidimensional corner effects

Recent results

The Lagrange-schemes have been modified in order to lead to reduced spatial stencils (paper in progress)

→ Expected to obtain far better performance on GPU.

Own GPU Experience feedback and partial concluding remarks

- Potentially important speedup
- Comp. effort-vs-memory access is critical for performance
- Uniform stencil patterns are preferred
- Need to rethink both models and methods
- Certainly not suited for sophisticated methods (DG with local time-stepping, see K.D. Munz's talk …)
- Complex geometries : embedded strategy

Issue

- High-order schemes
- Unstructured meshes

… see also Vivien Clauzon's talk concluding remarks.

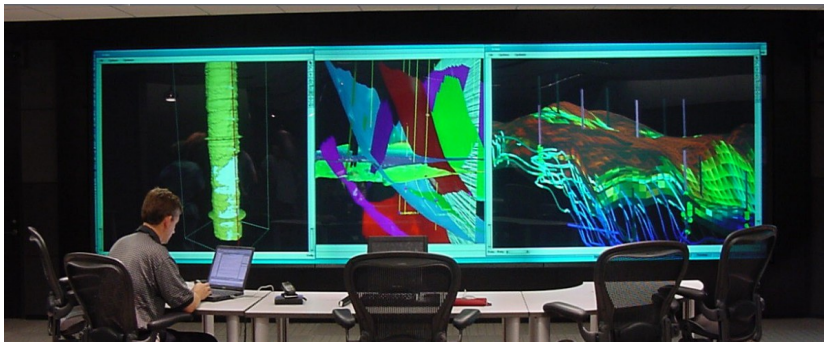
Foreseen benefits, applications

Benefits

- Surrogate model, precomputation before a realistic one

Use cases

- Conceptual and preliminary design
- Simulation and interaction with multiple devices
- Plant simulator



Courtesy : BARCO

© Dicolab Teamplayer

Acknowledgements

- Collaborators from LMT : C. Rey, R. Bennacer, M. Cremonesi
- CMLA GPU team : C. Labourdette, F. Pascal, L. Quivy, N. Vayatis
- Ext. collaborators : F. Salvarani (U. Pavia), S. Faure, L. Gouarin, B. Graille (LMO Orsay)
- Students : L. Besson, S. Hecht, M. Isnard (Dpt Maths & Comp. Sci. Dpt)

Institut Farman Grant, NVIDIA Grant, LRC MESO, LMH Labex