

Journée Farman
Ecole Normale Supérieure de Cachan
Pavillon des Jardins, 22 novembre 2012

Redesign de méthodes numériques sur GPU
Numerical methods revisited for GPU computing

Florian De Vuyst

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

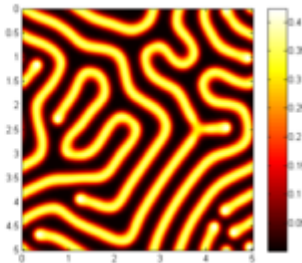
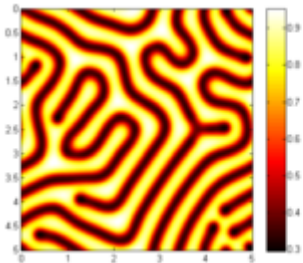
devuyst@cmla.ens-cachan.fr

Goal

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

Scope of the talk :

Fluid-type Partial Differential problems on GPU



Staff, people, acknowledgements

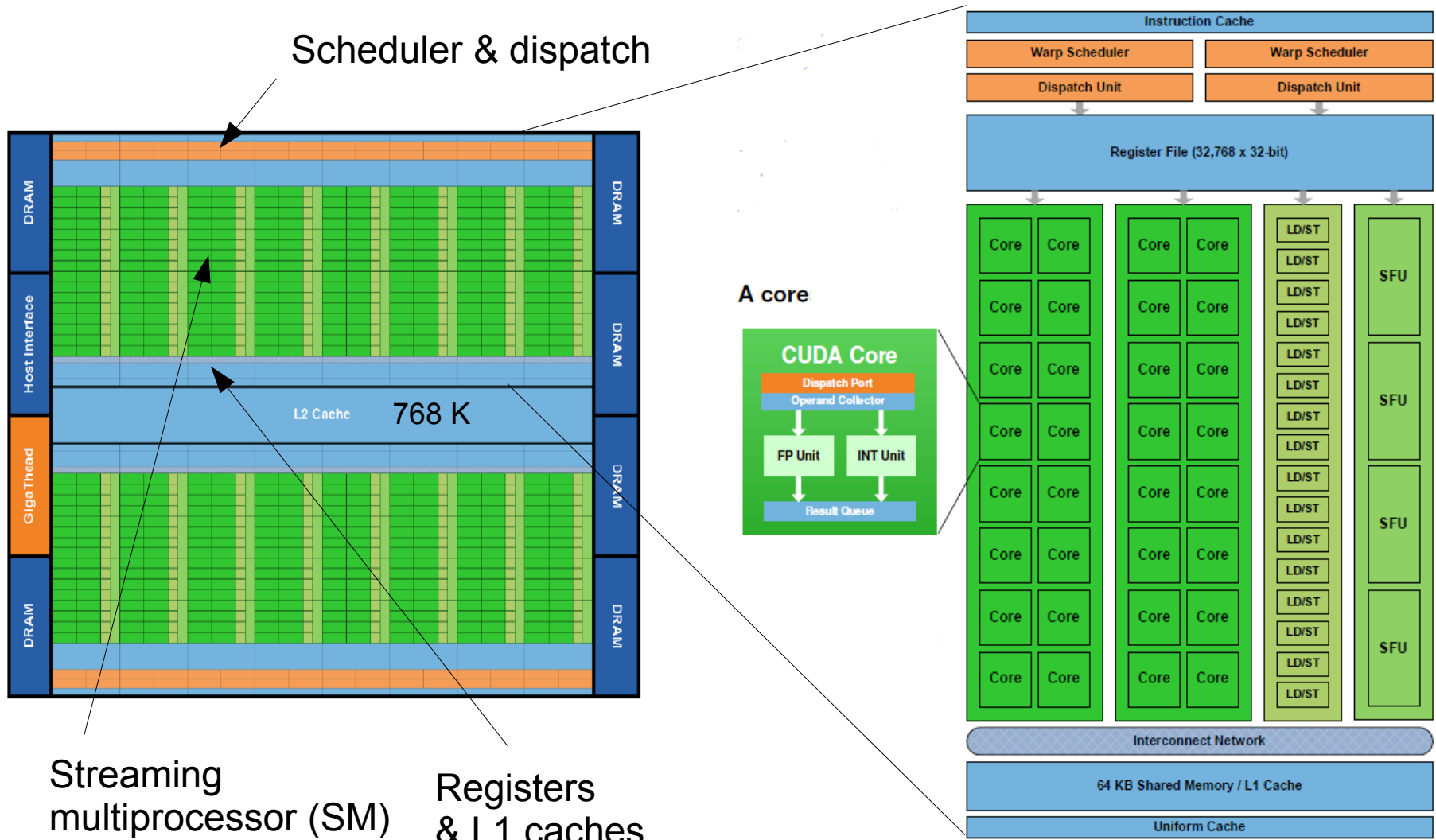
- Farman project FSI-LBM-GPU : C. Rey, R. Bennacer, M. Cremonesi, C. Labourdette
- GPU CMLA's emerging team : C. Labourdette, F. Pascal, L. Quivy, N. Vayatis
- Collaborators : F. Salvarani (U. Pavia), D. Bouche (CEA & CMLA), S. Faure, L. Gouarin (LMO Orsay)
- Students : L. Besson, S. Hecht, M. Isnard (Dpt Maths & Comp. Sci. Dpt)
- Support : Farman, NVIDIA Grant, CEA DAM DIF, Fondation Mathématique Jacques Hadamard, FMH Labex

Outline

1. GPU architecture
2. GPU programming model
3. Key factors of performance
4. Impact on the choice of (suitable) numerical methods
5. Ex1 - Lattice Boltzmann methods for Navier-Stokes equations
6. Ex2 - Suitable Finite Volume methods (FVM)
7. Ex3 – Lagrange-remap eulerian transport solvers
8. GPU & DIGISCOPE equipex (display wall project)
9. DEMO

1. NVIDIA GPU architecture

nVIDIA GPU *FERMI* compute architecture



Fermi Streaming Multiprocessor (SM)
(schematic from Nvidia)

Test hardware at CMLA

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



448 cores, FERMI
6 GB Mem
250 W

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)

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>

2. GPU programming model

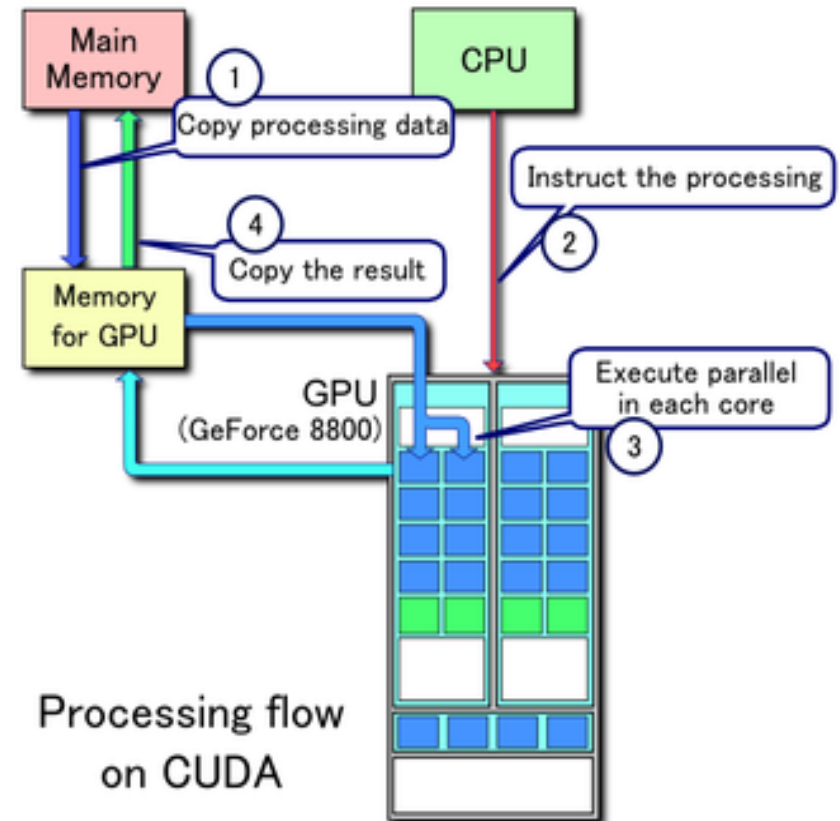
Programming model : different abstraction levels

- Different APIs :

- Higher level ↑
- CUDA *thrust* library API
 - CUDA runtime API
 - Driver API

- NVIDIA CUDA (Compute Unified Device Architecture) = parallel computing platform & programming model

- Others : OpenCL, OpenACC



thrust sample code (resembles the C++ STL lib)



<http://thrust.github.com/>

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/reduce.h>
#include <thrust/functional.h>
#include <cstdlib>

int main(void)
{
    // generate random data on the host
    thrust::host_vector<int> h_vec(100);
    thrust::generate(h_vec.begin(), h_vec.end(), rand);

    // transfer to device and compute sum
    thrust::device_vector<int> d_vec = h_vec;
    int x = thrust::reduce(d_vec.begin(), d_vec.end(), 0, thrust::plus<int>());
    return 0;
}
```

Pretty easy to use, especially for **parallel reduction**, but if performance improvement is needed, go to a lower abstraction level API (CUDA runtime)

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 >>> (... , ...);`

CUDA sample code

```
1 #include <stdio.h>
2 #include <cuda.h>
3 // Kernel that executes on the CUDA device
4 __global__ void sq_array(float *a, int N)
5 {
6     int idx = blockIdx.x * blockDim.x + threadIdx.x;
7     if (idx<N) a[idx] = a[idx] * a[idx];
8 }
9 //
10 int main(void)
11 {
12     float *a_h, *a_d; // Pointers to host (h) & device (d) arrays
13     const int N = 10000000; // Number of elements in arrays
14     size_t size = N * sizeof(float);
15     a_h = (float *) malloc(size); // allocate array on host
16     cudaMalloc((void **) &a_d, size); // (CUDA) allocate array on device
17     // Initialize host array and copy it to CUDA device
18     for (int i=0; i<N; i++) a_h[i] = (float)i;
19     cudaMemcpy(a_d, a_h, size,
20     cudaMemcpyHostToDevice); // (CUDA) copy from host to device
21     // Do calculation on device:
22     int block_size = 32;
23     int n_blocks = N/block_size + (N%block_size == 0 ? 0:1);
24
25     sq_array <<< n_blocks, block_size >>> (a_d, N); // (CUDA) kernel call
26     // Retrieve result from device and store it in host array
27     cudaMemcpy(a_h, a_d, sizeof(float)*N, cudaMemcpyDeviceToHost);
28     // Printing the results ...
29     for (int i=0; i<N; i++) printf("%d %f\n", i, a_h[i]);
30     // free memory
31     free(a_h); cudaFree(a_d);
32     // done.
33 }
```

3. Key factors of performance

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 arrays of structures, accessors, hasch tables, specialized structs (float4) or data structures (textures), ...

NVIDIA tools for occupancy optimization

CUDA GPU Occupancy Calculator

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: [\(Help\)](#)

| | |
|---|------|
| Active Threads per Multiprocessor | 2048 |
| 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 |

| = Allocatable | | | |
|---|-----------|--------------|---------------|
| Allocated Resources | Per Block | Limit Per SM | Blocks Per SM |
| Warps (Threads Per Block / Threads Per Warp) | 8 | 64 | 8 |
| Registers (Warp limit per SM due to per-warp reg count) | 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 Multiprocessor | 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 | |

[Click Here for detailed instructions on how to use this occupancy calculator.](#)

[For more information on NVIDIA CUDA, visit http://developer.nvidia.com/cuda](http://developer.nvidia.com/cuda)

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 of Varying Block Size
My Block Size: 256

Impact of Varying Register Count Per Thread
My Register Count: 32

Impact of Varying Shared Memory Usage Per Block
My Shared Memory: 4096

4. Impact on the choice of (suitable) 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

Ex1 - Lattice Boltzmann methods for Navier-Stokes equations

Lattice Boltzmann methods

- 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

References : [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]

Lattice BGK model

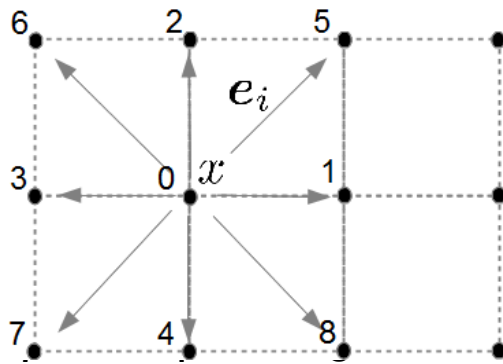
Based on a simple discretization of the Boltzmann equation with BGK approximation for the collision term :

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

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

Relaxation toward some equilibrium...

$$\mathbf{e}_0 = 0, \mathbf{e}_1 = (1, 0), \mathbf{e}_2 = (1, 1), \dots$$



Moments:

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

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

Lattice BGK model

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

- LB equation

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

- Requirements

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

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

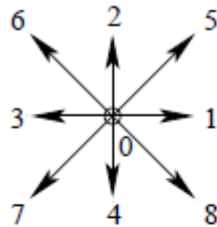
$$\sum_{i \in \mathcal{S}} \mathbf{e}_i f_i^{eq}(\rho, \mathbf{u}) = \rho \mathbf{u}.$$

+ some galilean invariance conditions

2D and 3D lattices & equilibrium distribution

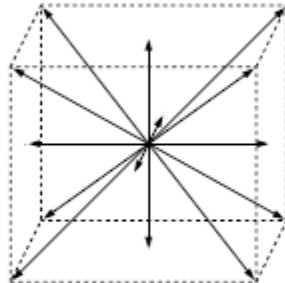
$$\text{D2Q9} \quad f_i^{eq} = \rho w_i \left[1 + 3\mathbf{u} \cdot \mathbf{e}_i + \frac{9}{2}(\mathbf{u} \cdot \mathbf{e}_i)^2 - \frac{3}{2}\mathbf{u} \cdot \mathbf{u} \right]$$

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:



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

Source : [A. Wagner 08] 23

Practical implementation « *stream-and-collide* »

1. *Stream* step

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

2. *Collision* step

- Compute the moments

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

- Compute the equilibrium function $\tilde{f}_i^{eq} = \rho w_i [\dots]$
- Collision step :

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

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

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

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

with

$$\nu = \frac{1}{3}(2\tau - 1).$$

Requires : $u = M \ll 1$.

Artificial EOS:

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

Von Neuman linear stability
analysis of LBGK scheme:

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

Ex2 - Suitable Finite Volume methods (FVM)

2. Finite Volume method for 2D compressible Euler equations

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

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

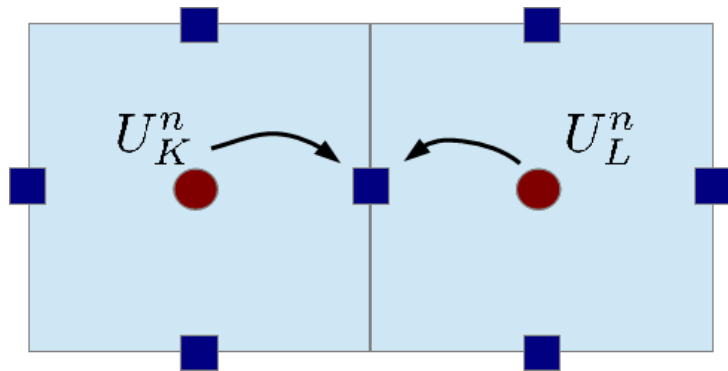
$$\partial_t (\rho E) + \nabla \cdot ((\rho E + p) \mathbf{u}) = 0.$$

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

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

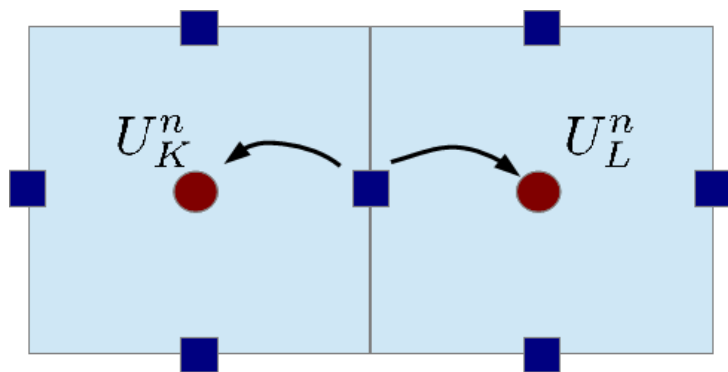
Flux Difference Splitting (FDS) vs Flux Vector Splitting (FVS) methods on GPU

FDS case



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

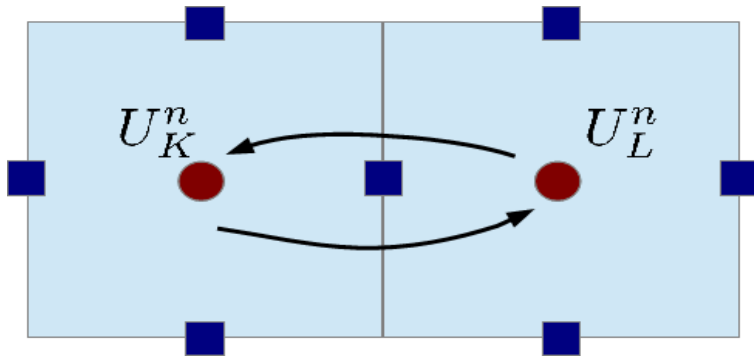
$$\Phi(U_K^n, U_L^n, \nu_{KL}) = \frac{F(U_K^n) + F(U_L^n)}{2} - \frac{1}{2} \int_{\Gamma_{KL}} |A|(s) U'(s) ds$$



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

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

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



- 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 Applicandae Mathematicae, 2012, 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

Ex3 – Lagrange-remap eulerian transport solvers

Lagrange-remap (LR) schemes

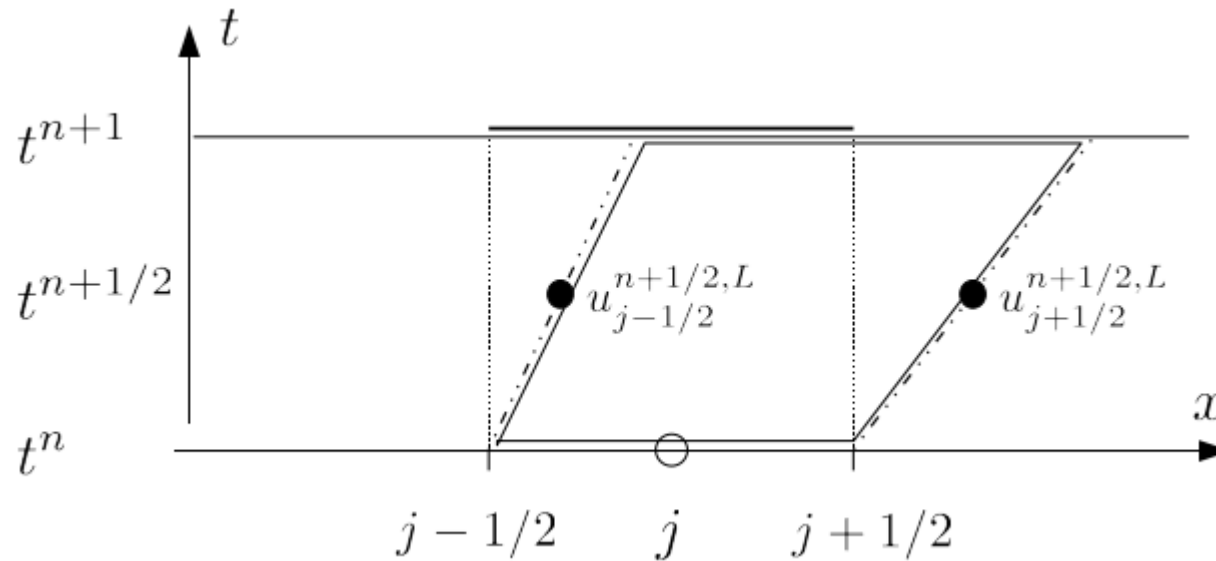
- Define a piecewise \mathcal{C}^1 Lagrange transformation operator $\mathcal{L}(\mathbf{x}, t; t_0, \mathbf{x}_0) : d\mathbf{x}/dt = \mathbf{u}(\mathbf{x}), \mathbf{x}(t=0) = \mathbf{x}_0$.
- Use the Reynolds transport theorem:

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

for any moving domain Ω_t , quantity q .

- Finite volume approximation
- Move the mesh according to a Lagrange operator \mathcal{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:

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

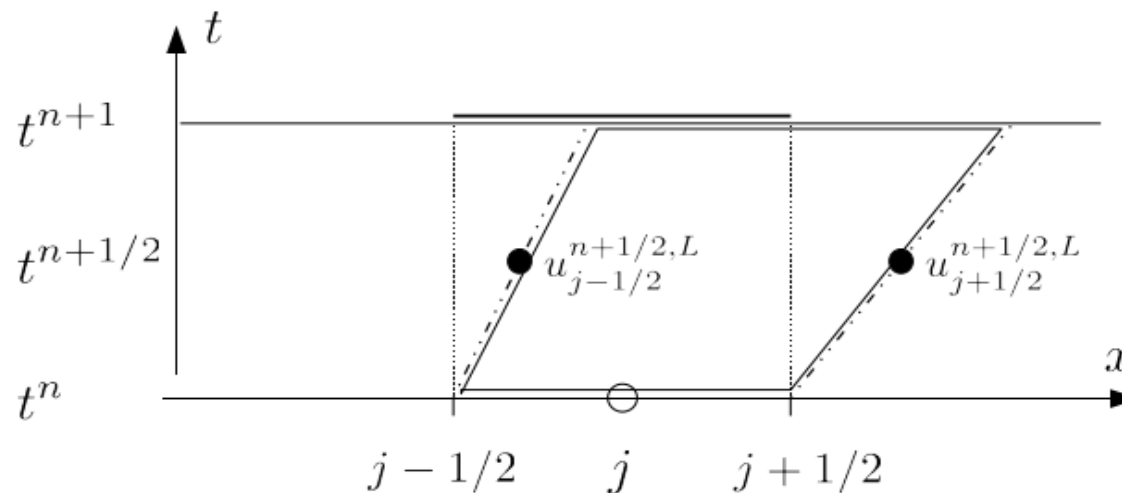
Conservative reformulation (2)

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:

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



Conservative reformulation (3)

Mass balance rewriting : 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

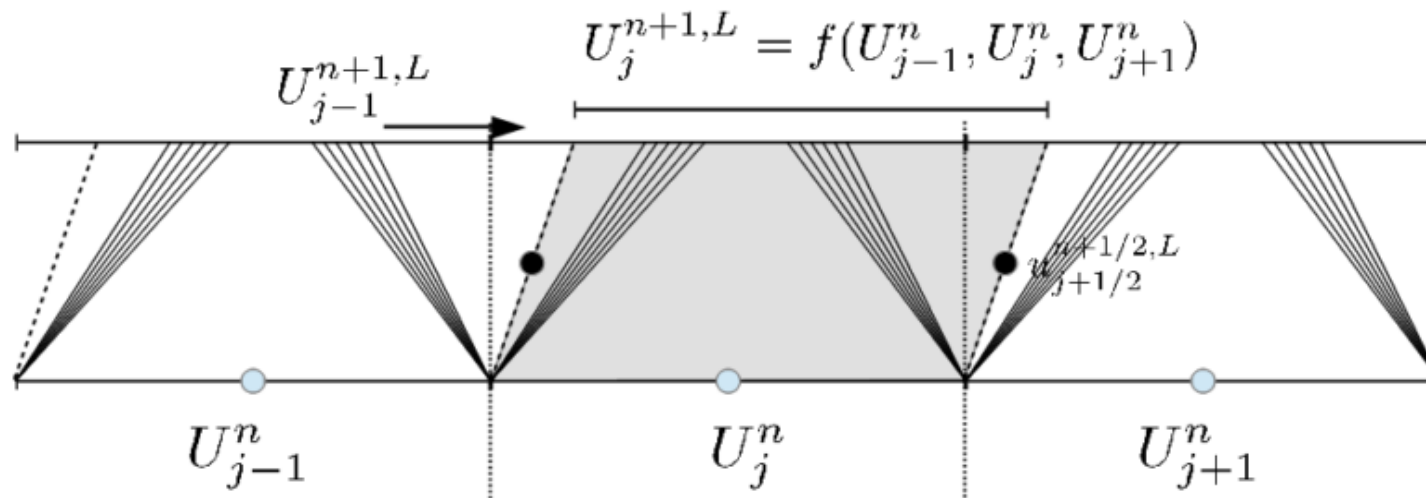
$$\rho_j^{n+1} = \rho_j^n - \frac{\Delta t}{h} \left(\Phi_{m,j+1/2}^{n+1/2,n+1} - \Phi_{m,j-1/2}^{n+1/2,n+1} \right),$$

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

[De Vuyst, Fochesato, Loubère, Saas, Motte, Ghidaglia, CRAS Meca, submitted 2012]

Remark

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

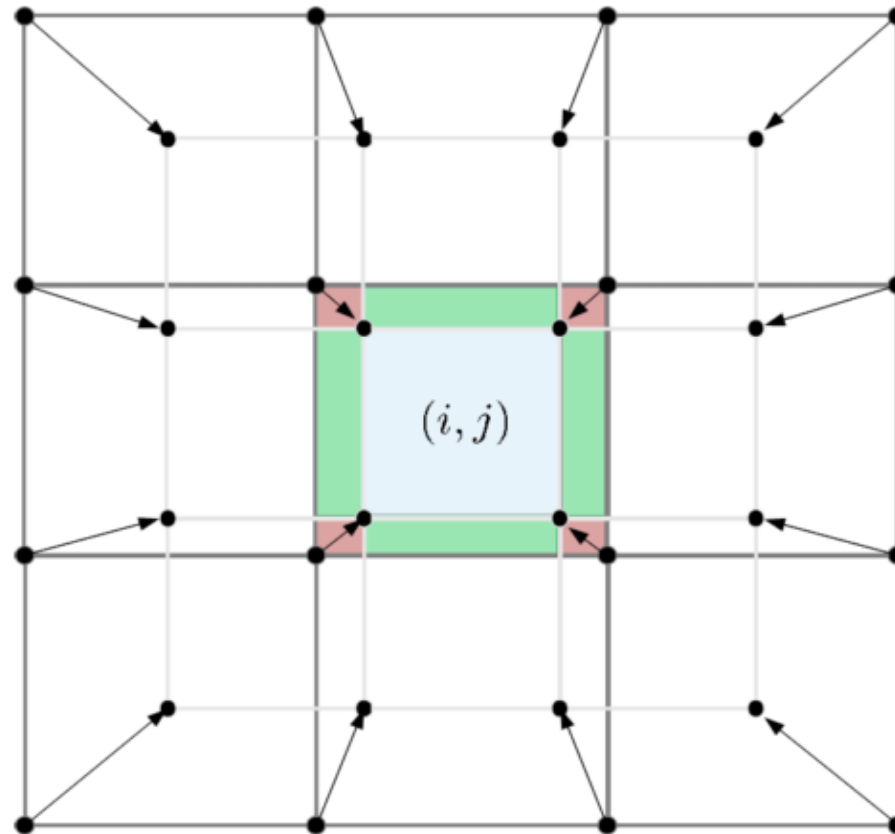


$$U_j^{n+1} = U_j^n - \frac{\Delta t}{h} \left(\Phi_{j+1/2}^{n,n+1} - \Phi_{j-1/2}^{n,n+1} \right),$$

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

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

2D case



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

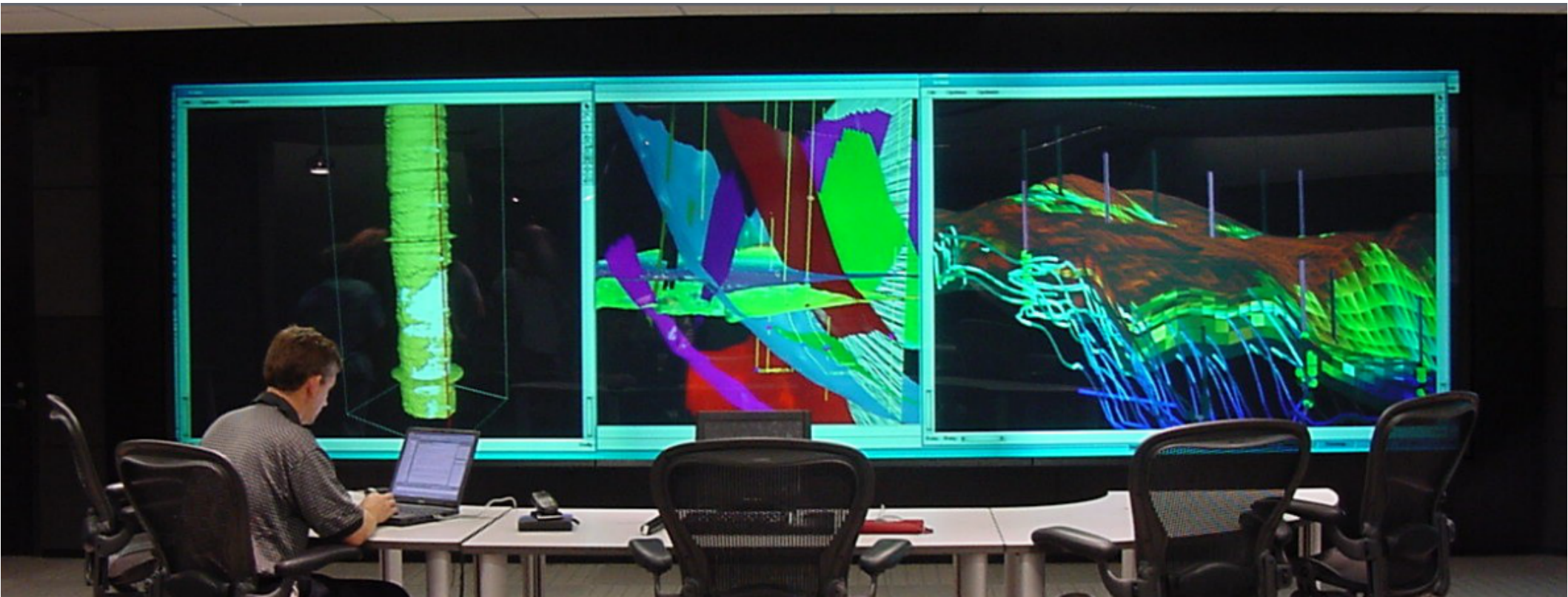
Recent results

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

→ Expected far better performance on GPU.

Issue : higher order schemes with slope reconstructions, multimaterial problems with interface reconstruction, ...

8. GPU & DIGISCOPE equipex (display wall project)



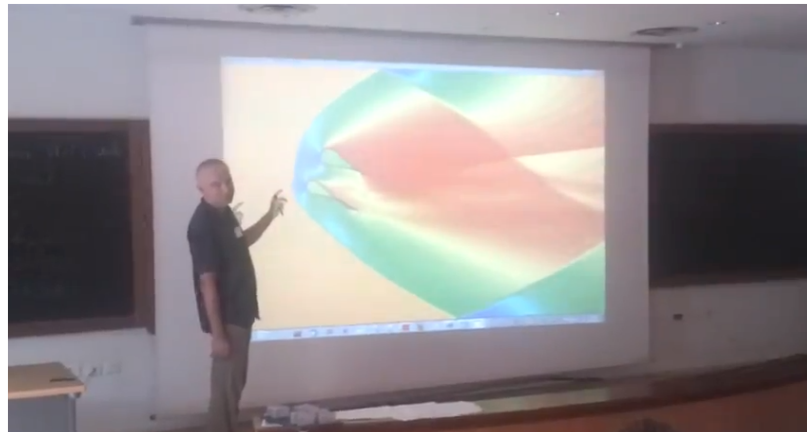
Courtesy : Barco

Expected use of GPU-accelerated CFD with DIGISCOPE

- [Industrial applications] Collaborative optimal placement of heat exchanger tubes into heat transport devices
- [Entertainment] Human embedding and interaction with scenes involving fluids (4-sensor tracking + kinect)
- [Life science] Real-time exploration into the parameter space of chaotic reaction-diffusion solutions (Gray-Scott equations)



time for a demo ...



Centre de mathématiques et de
leurs applications

CMLA UMR 8536



Cmla

Thank you for your attention