A generic Discontinuous Galerkin solver based on OpenCL task graph. Application to electromagnetic compatibility.

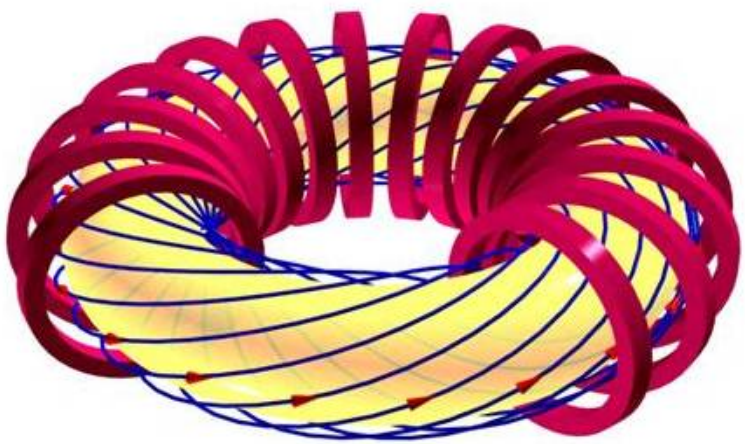
Philippe HELLUY [1,2]

[1]IRMA, Université de Strasbourg, [2]Inria Tonus, France

CEMRACS, July 2015, Luminy

ITER

Context: International Thermonuclear Experimental Reactor (ITER project). Thermonuclear fusion in a hot hydrogen plasma (more than 100 million degrees °C). Clean energy of the future. Tokamak: magnetic plasma confinement in a torus.



Plasma physics modeling: Vlasov-Maxwell

- Unknowns: distribution function $f(x, v, t)$ =number of ions at point x and time t having velocity v; electromagnetic field (E, B).
- Vlasov equation (6D (x, v) phase space)

$$\partial_t f + v \cdot \nabla_x f + (E + v \times B) \cdot \nabla_v f = 0.$$

- Maxwell equations (3D x space)

$$\partial_t E - \nabla \times B = j, \quad \partial_t B + \nabla \times E = 0.$$

The current j couples Vlasov and Maxwell

$$j(x, t) = \int_v f(x, v, t) v \, dv.$$

- General framework: **conservation laws**.

Conservation laws

Many equations in physics are systems of conservation laws:

$$\partial_t W + \sum_{i=1}^{d} \partial_i F^i(W) = 0.$$

- $W = W(x,t) \in \mathbb{R}^m$: vector of conserved quantities;
- $x = (x^1 \dots x^d)$: space variable, d: space dimension, t: time;
- $\partial_t = \frac{\partial}{\partial t}$, $\partial_i = \frac{\partial}{\partial x_i}$;
- $W = \begin{bmatrix} E \\ B \end{bmatrix}$ (Maxwell) or $W = \begin{bmatrix} \vdots \\ \int_v f v^k dv \\ \vdots \end{bmatrix}$ (Vlasov);
- $F^i(W)$: flux vector (contains the physics).

SCHNAPS

- Factorize software developments: design of a generic, non-linear conservation laws solver.
- Optimizations for addressing hybrid CPU/GPU clusters.
- Fundamental and industrial applications.

SCHNAPS: “Solveur Conservatif Hyperbolique Non-linéaire Appliqué aux PlasmaS”.

- OpenCL: SIMD fine grain parallelism (GPU).
- MPI for dealing with MIMD coarse grain parallelism.
- Task graph programming model.

Outlines

1. Simple approach:
   - 2D structured grids;
   - Finite Difference (FD) + Strang directional splitting;
   - OpenCL;
   - Synchronous OpenCL/MPI numerical simulations.
2. More general approach:
   - 3D unstructured grids;
   - Discontinuous Galerkin (DG);
   - Asynchronous OpenCL/MPI numerical simulations.

1) 2D Structured grid

- Grid step: Δx, time step $\Delta t \leq \Delta x / V_{\max}$, grid directions $n_1 = (1, 0)$, $n_2 = (0, 1)$.
- Approximation $W_{i,j}^p \simeq W(i\Delta x, j\Delta x, p\Delta t)$.
- Finite Difference (FD) method + Strang splitting:

$$\frac{W_{i,j}^* - W_{i,j}^p}{\Delta t} + \frac{F(W_{i,j}, W_{i+1,j}, n_1) - F(W_{i-1,j}, W_{i,j}, n_1)}{\Delta x} = 0,$$

$$\frac{W_{i,j}^{p+1} - W_{i,j}^*}{\Delta t} + \frac{F(W_{i,j}, W_{i,j+1}, n_2) - F(W_{i,j-1}, W_{i,j}, n_2)}{\Delta x} = 0.$$

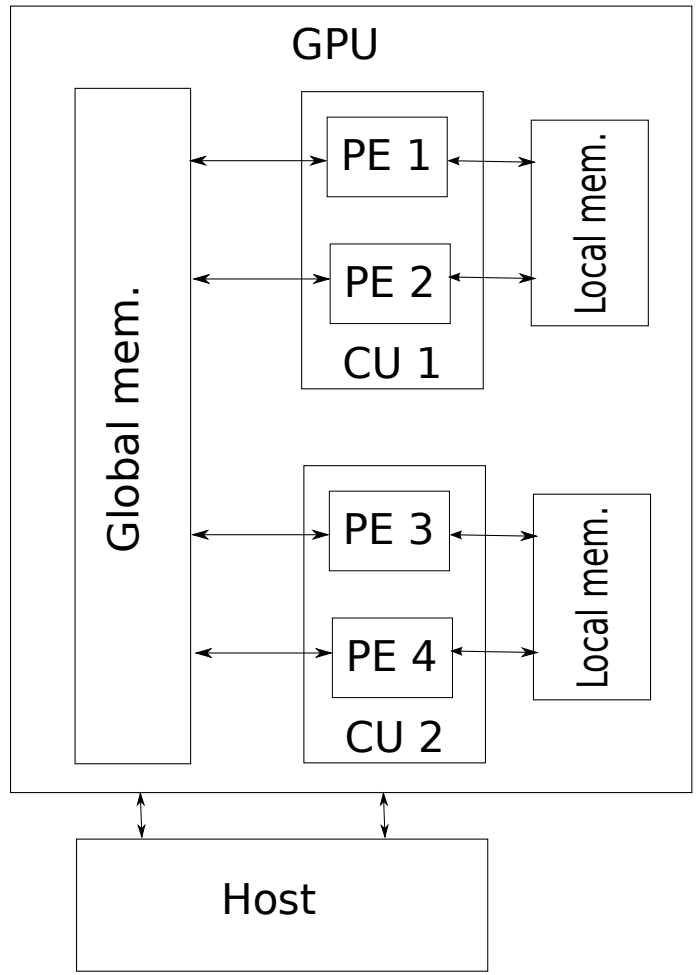
- Numerical flux: $F(W_L, W_R, n), \quad F(W, W, n) = F(W) \cdot n.$

OpenCL

- OpenCL: "Open Computing Language". Library of C functions for driving the GPU (or any multicore accelerator). Similar to but more general than CUDA. SYCL: C++ templated version.
- API managed by the Khronos Group (in charge also of OpenGL) `https://www.khronos.org/`
- Industry standard: the very same program can really run on many accelerators. Drivers exist for: NVIDIA GPUs, AMD CPUs and GPUs, Intel CPUs and GPUs, MIC, ARM (CPU+GPU), IBM, etc.
- An OpenCL program can access accelerators of different vendors at the same time (kernels compilation at runtime).
- Also "Meta" drivers: SOCL (StarPU), SnuCL, etc.
- Python bindings: PyOpenCL `https://github.com/pyopencl/pyopencl`

OpenCL abstraction

- An accelerator is made of compute units ("work-groups") of several processors ("work-items") sharing a small local cache memory.
- All the processors have access to the global memory.
- The same program (a "kernel") can be executed by all the work-items at the same time.
- OpenCL manages the asynchronous distribution of the work-items on the actual processors.



OpenCL specificities

- The local (cache) memory is small but fast.
- The global memory is bigger but slower.
- Accessing the global memory of the GPU is faster if neighboring processors access neighboring locations ("coalescent" access).
- Accessing the host memory is very slow.
- Branching may be costly (SIMD parallelism).
- Kernel compilation at runtime: increase verbosity, but very interesting for **metaprogramming and performance portability**.
- **OpenCL manages events and a task graph for asynchronous kernel launching.**

OpenCL implementation

The data are arranged in a (i, j) matrix. 1 work-item = 1 cell (i, j).
1 work-group = 1 row i.
For each time step p:

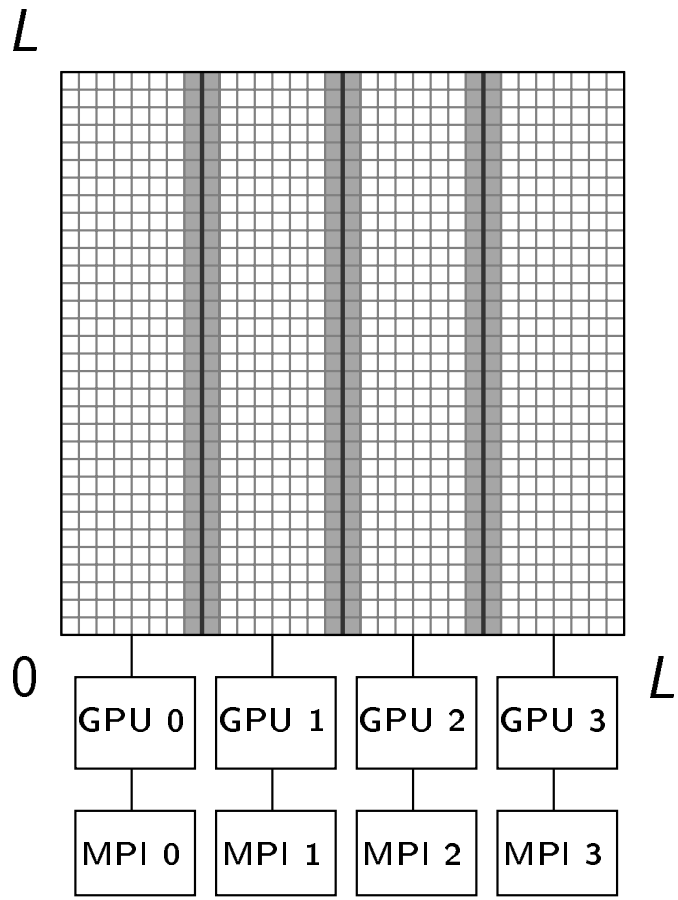
- compute the fluxes balance in the x^1-direction for each cell of each row of the grid.
- transpose the matrix (exchange i and j) in a **coalescent** way.
- compute the fluxes balance in the x^2-direction for each row of the transposed grid.
- transpose again the matrix.

Kernel code

```
__kernel void flux_balance(int m, float dt_over_dx,
                           __global float wnow[][_NY][_NX],
                           __global float wnext[][_NY][_NX])
{
  int i = get_group_id(0);
  int j = get_local_id(0);

  float w1[m], w2[m], fR[m], fL[m];

  for(int ii = 0; ii < m; ++ii)
    {
      w1[ii] = wnow[ii][i][j];
      w2[ii] = wnow[ii][i][j+1];
    }

  numflux(w1,w2,fR);

  for(int ii = 0; ii < m; ++ii)
    w2[ii] = wnow[ii][i][j-1];

  numflux(w2,w1,fL);

  for(int ii = 0; ii < m; ++ii)
    wnext[ii][i][j] -= dt_over_dx * (fR[ii] - fL[ii]);
}
```

OpenCL + synchronous MPI

- Use of several GPUs;
- Subdomain decomposition compatible with the transposition algorithm;
- 1 GPU = 1 subdomain = 1 MPI node;
- MPI for exchanging data between GPUs (greyed cells layers).



Comparisons (M. Massaro)

On large grids ($> 1024 \times 1024$). We compare:

- a naive C implementation (compiled with -O3)
- an optimized (tiling) OpenMP implementation of the FD scheme on 2x6-core CPUs;
- the OpenCL implementation running on 2x6-core CPUs, NVidia or AMD GPU;
- the OpenCL+MPI implementation running on 4 GPUs.

Implementation	Time	Speedup
Naive C & -O3		$\simeq 1/20$
OpenMP (CPU Intel 2x6 cores)	717 s	1
OpenCL (CPU Intel 2x6 cores)	996 s	0.7
OpenCL (NVidia Tesla K20)	45 s	16
OpenCL (AMD Radeon HD 7970)	38 s	19
OpenCL + MPI (4 x NVIDIA K20)	12 s	58

The GPU performance depends essentially on the transposition kernel... We achieve approximately 800 GFLOP/s/GPU.

Shock-bubble interaction (J. Jung)

$$W = (\rho, \rho u^1, \rho u^2, \rho Q, \rho\varphi)^T, \quad Q = e + |u|^2/2, \quad p = p(\rho, e, \varphi),$$

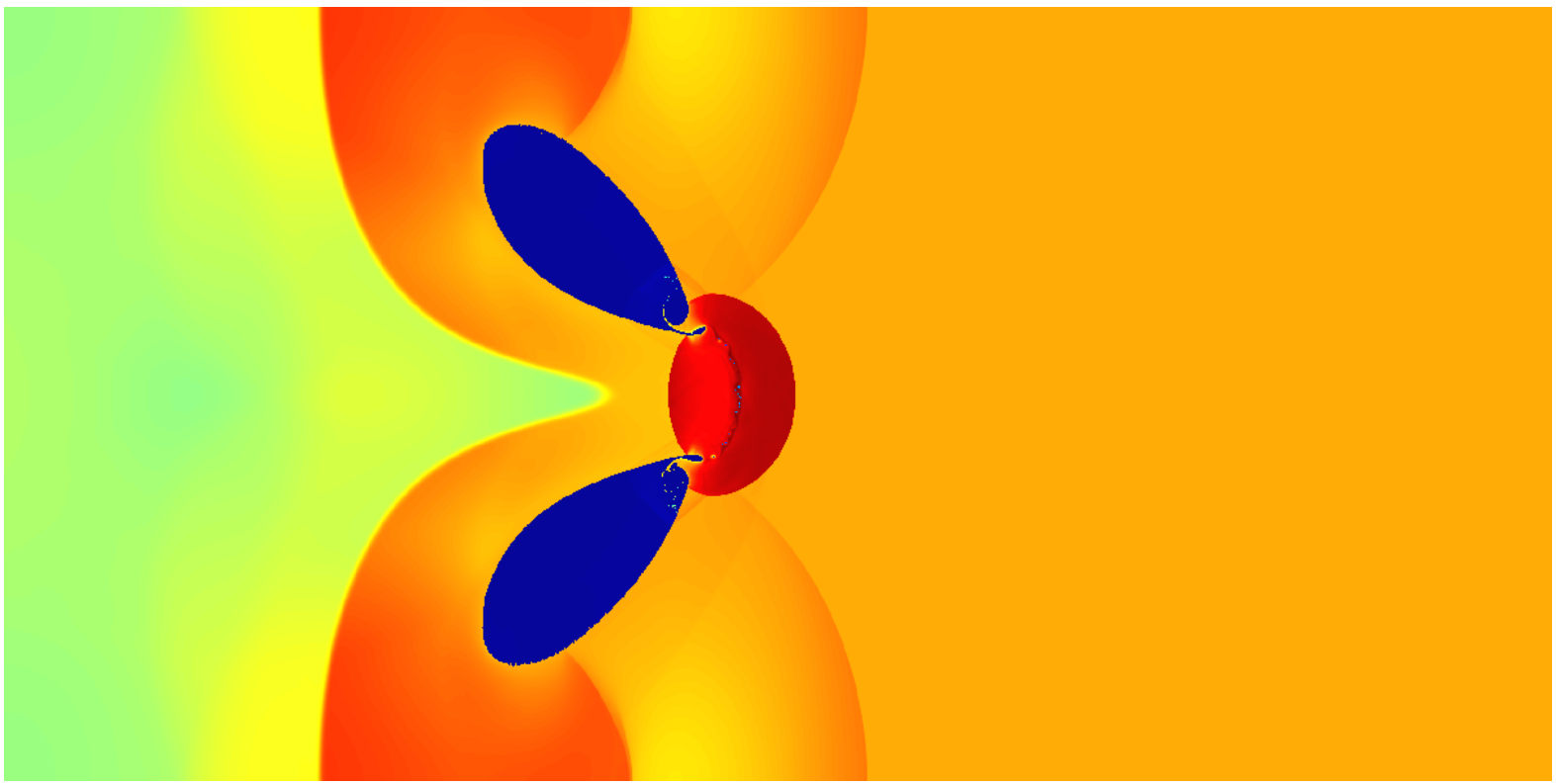
$$F(w) \cdot n = (\rho u \cdot n, \rho(u \cdot n)u^T + pn^T, (\rho Q + p)u \cdot n, \rho\varphi u \cdot n)^T.$$

- Simulation of a compressible two-fluid flow: interaction of a shock wave in a liquid with a gas bubble
- Coarse mesh OpenCL simulation on an AMD HD 5850
- OpenGL/OpenCL interop + video capture.

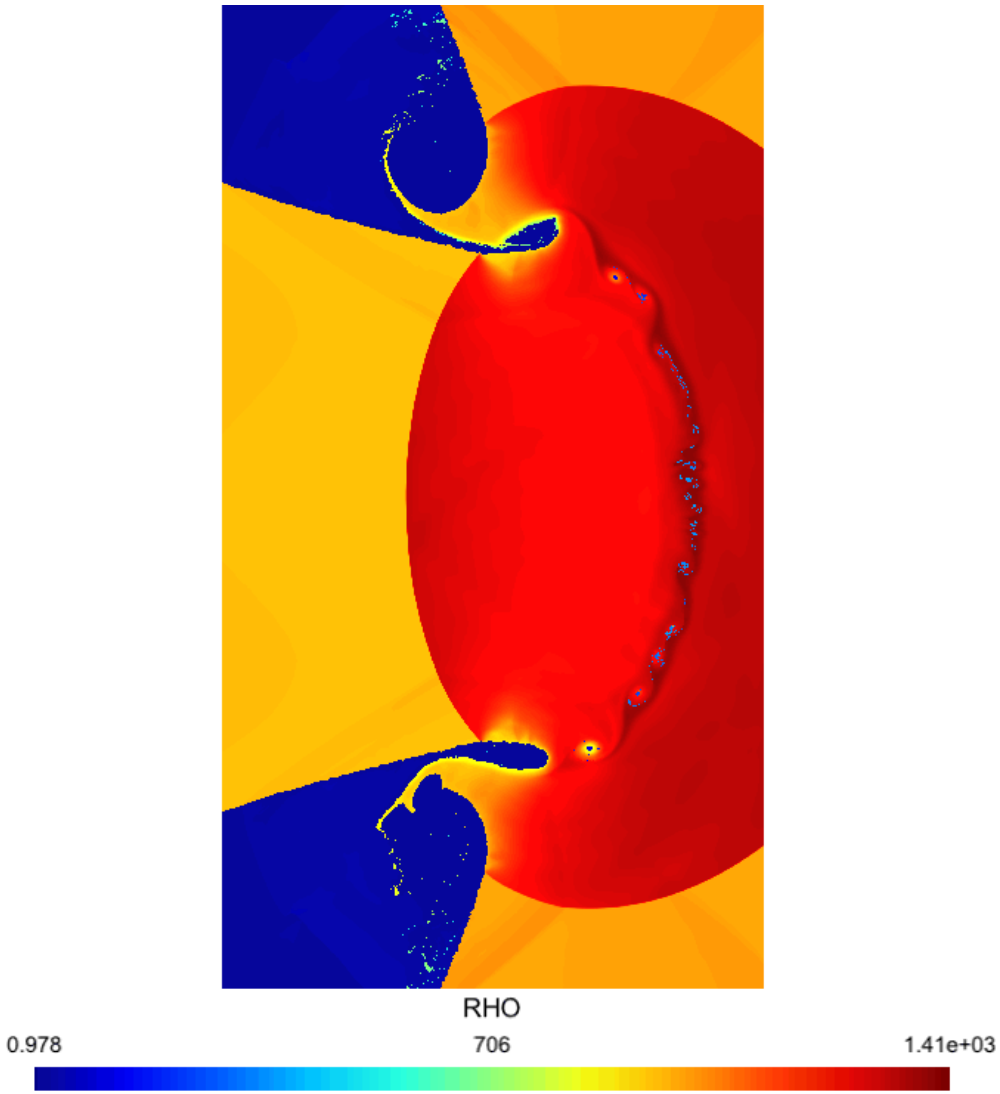
`https://www.youtube.com/watch?v=c8hcqihJzbw`

Very fine mesh

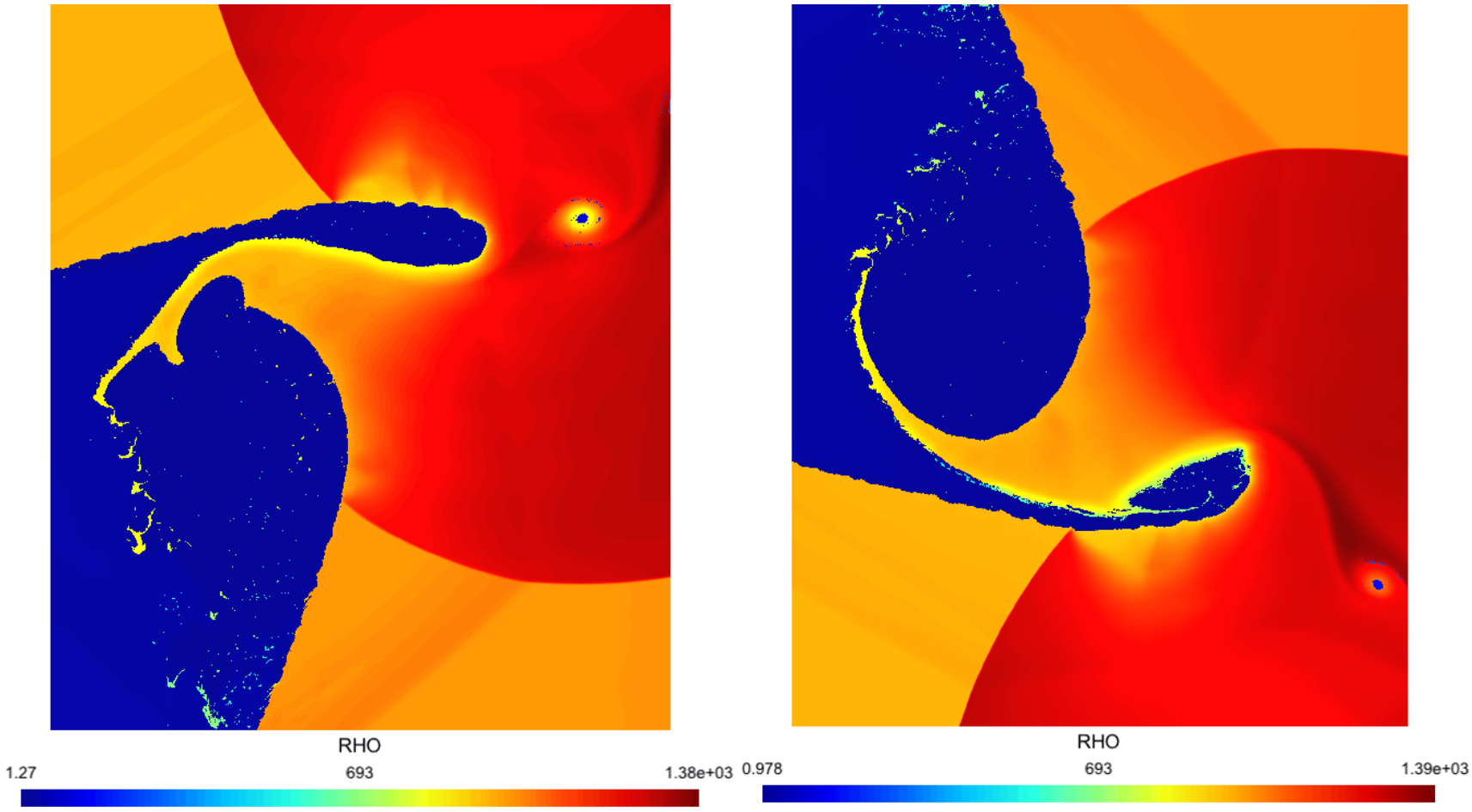
- Very fine mesh OpenCL + MPI simulation, 40,000x20,000 grid. 4 billions unknowns per time step
- 10xNVIDIA K20 GPUs, 30 hours
- Red=high density (compressed liquid); blue=low density (gas).



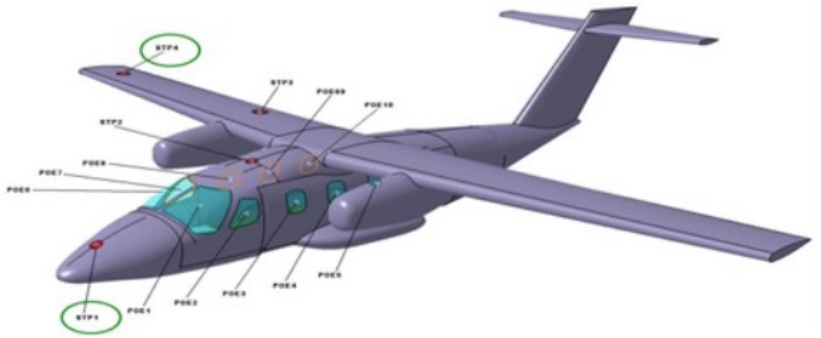
Zoom 1

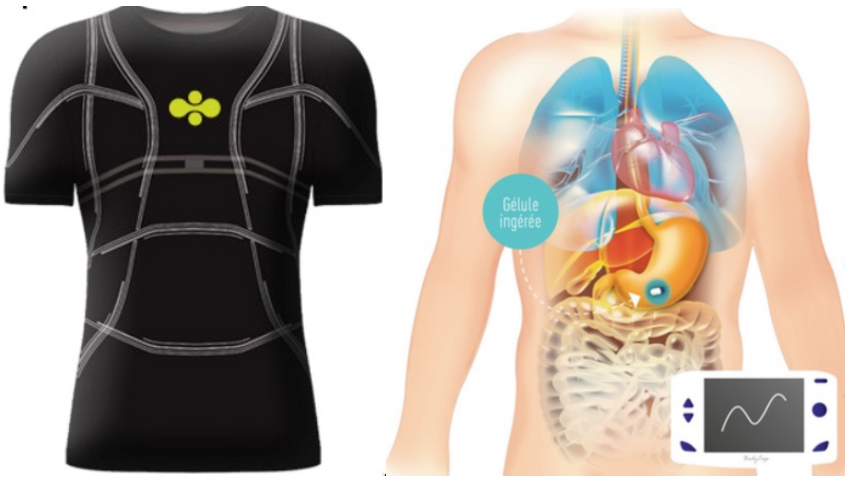


Zoom 2



Unstructured grids





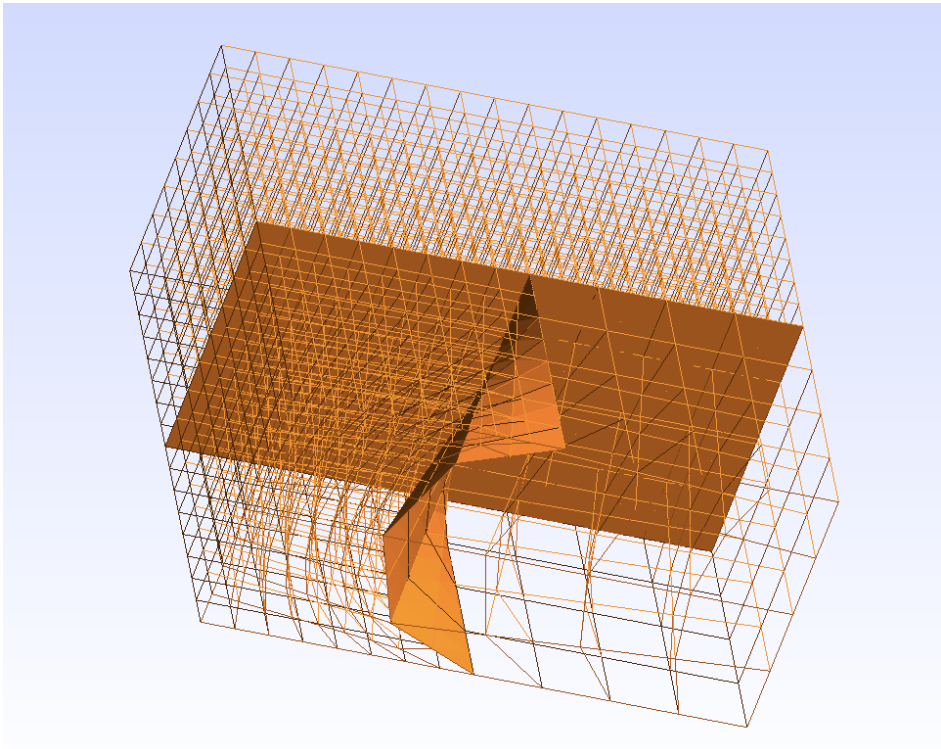
(Project with Thales, AxesSim, Body Cap, Citizens Sciences)

- Unstructured hexahedra mesh for representing complex geometries.
- Subdomain decomposition. 1 domain = 1 MPI node = 1 OpenCL device.
- Zone decomposition. Each subdomain is split into volume zones and interface zones.
- Non-conformity between zones is allowed.

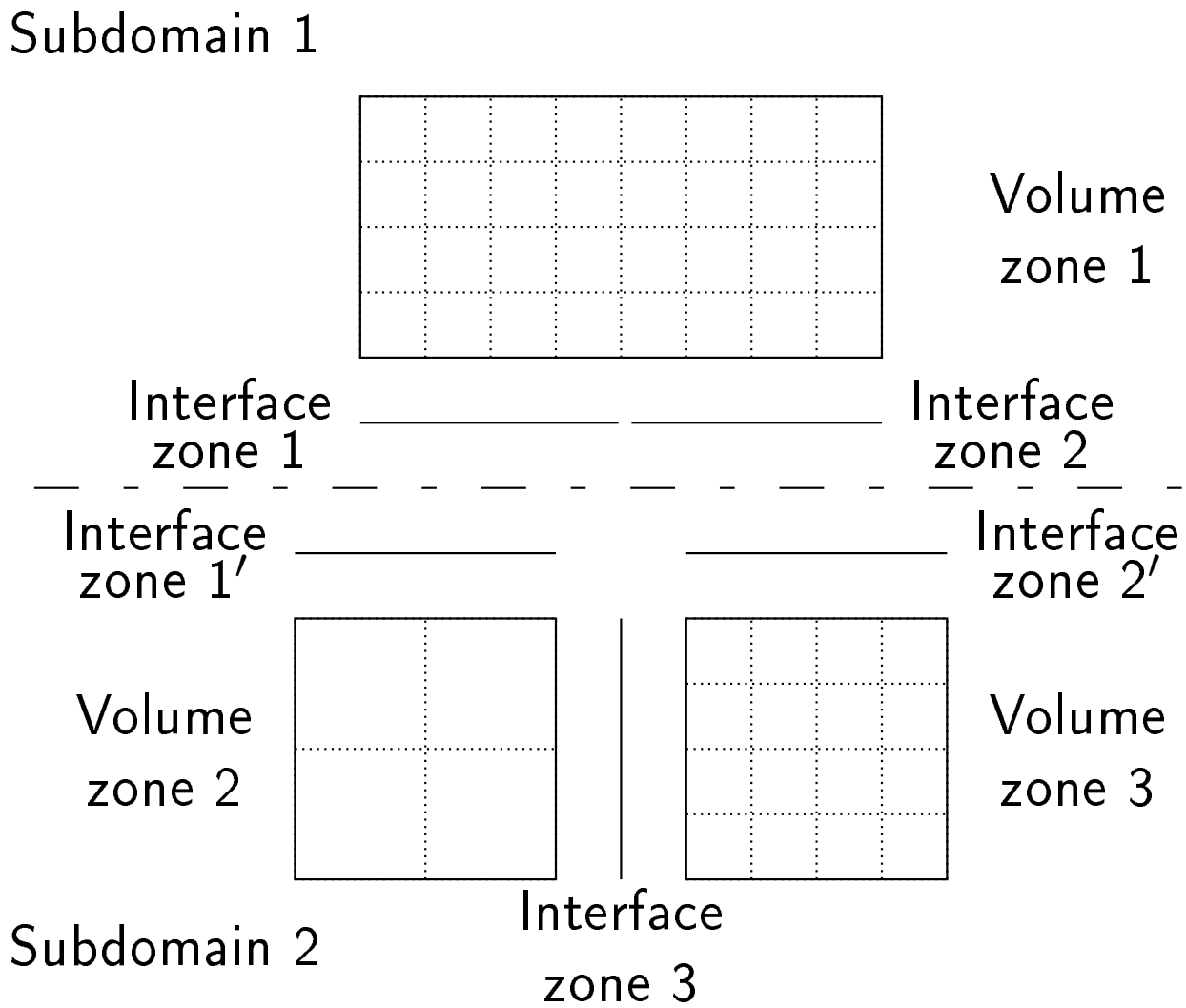
Mesh example

A non-conforming mesh with two subdomains, three volume zones and three interface zones.

- Subdomain 1: only one big refined volume zone. Two interface zones.
- Subdomain 2: two small volume zones (coarse and refined). Three interface zones.



Mesh structure

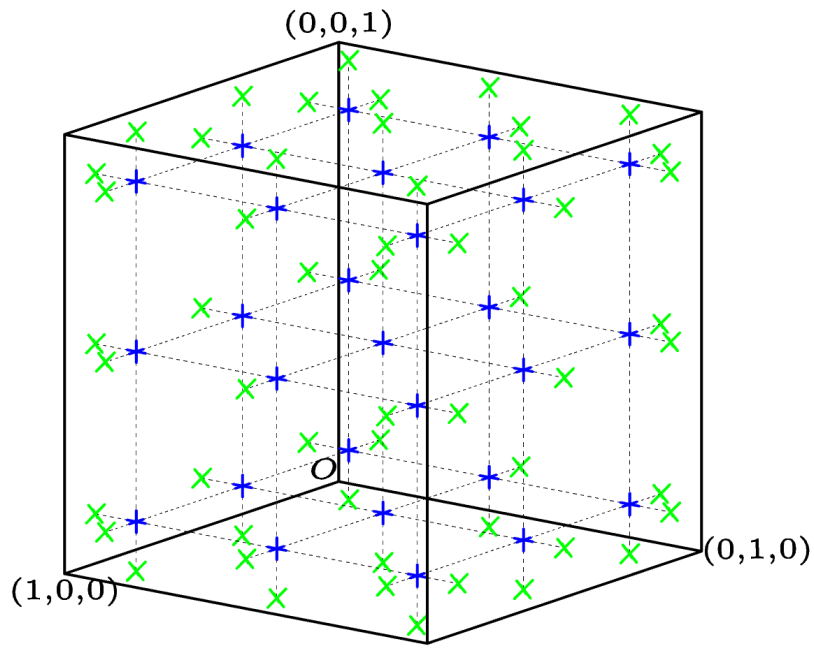


Discontinuous Galerkin (DG) approximation [4]

In each cell L of the mesh, the conserved quantities are expanded on Lagrange polynomial basis functions

$$W(x,t) = \sum_j W_L^j(t)\psi_j^L(x), \quad x \in L.$$

- L is a (possibly stretched) hexahedron
- W is determined by its values at blue (volume) Gauss points
- W is discontinuous at green (faces) Gauss points.

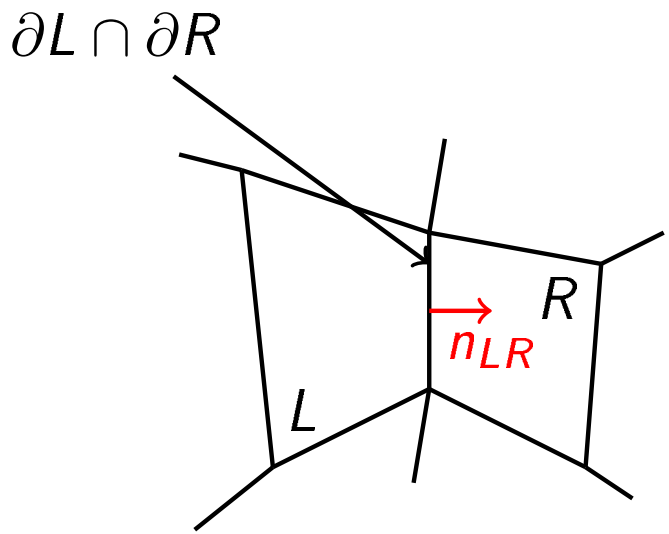


DG formulation [7, 8]

The numerical solution satisfies the DG approximation scheme

$$\forall L, \forall i \quad \int_L \partial_t W_L \psi_i^L - \int_L F(W_L, W_L, \nabla \psi_i^L) + \int_{\partial L} F(W_L, W_R, n_{LR}) \psi_i^L = 0.$$

- R denotes the neighbor cells along ∂L.
- n_{LR} is the unit normal vector on ∂L oriented from L to R.
- $F(W_L, W_R, n)$: numerical flux.
- $F(W, W, n) = \sum_k F^k(W) n_k$.



Time integration of a system of ordinary differential equations.

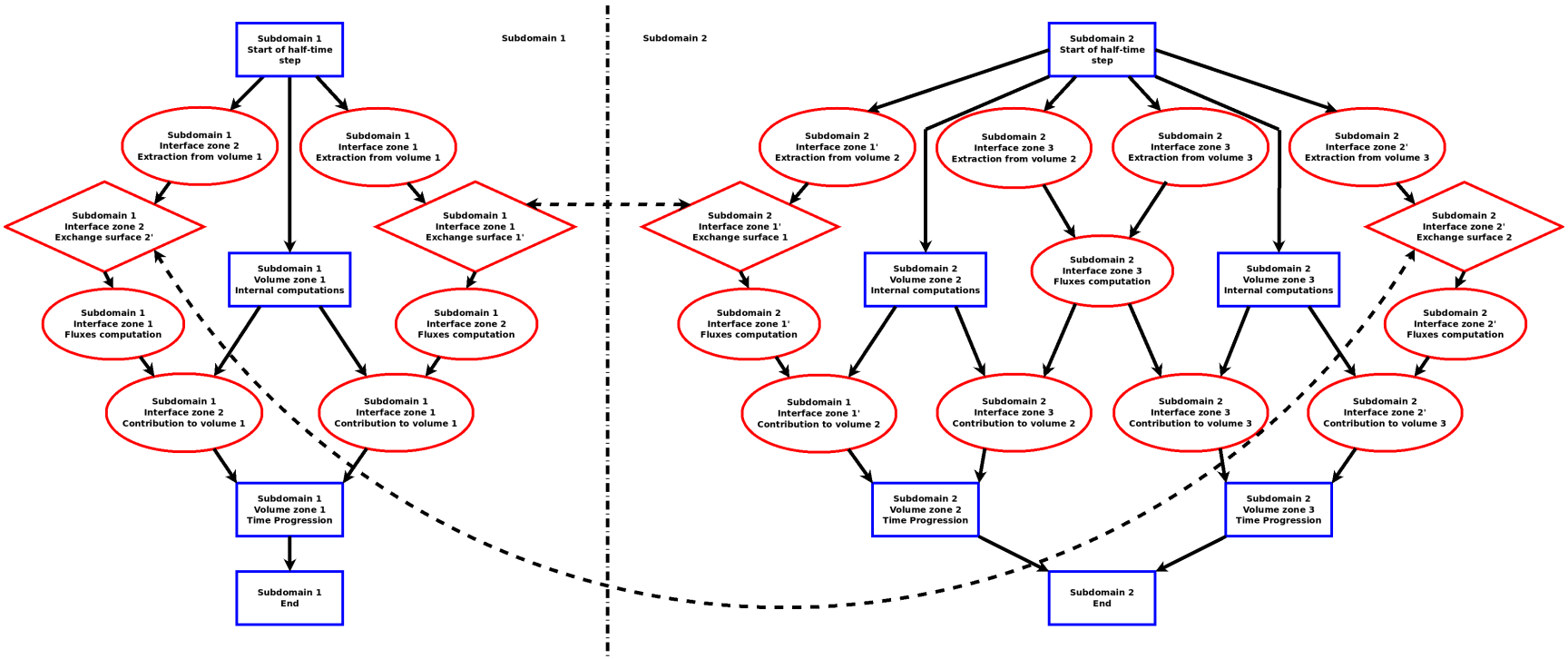
Tasks

- Elementary tasks attached to volume or interface zones
- A task is associated to a computational OpenCL kernel or to memory operations (GPU$\leftrightarrow$CPU and MPI transfers).
- The optimized design of the computational kernels is tricky...
  - Hexahedra mesh optimizations ($(D+1)^3 \rightarrow 3(D+1)$ complexity).
  - Idling work-item strategy for avoiding cache misses.
  - Our FLOPs are good FLOPs !
- See `https://hal.archives-ouvertes.fr/hal-01134222v2`

Tasks

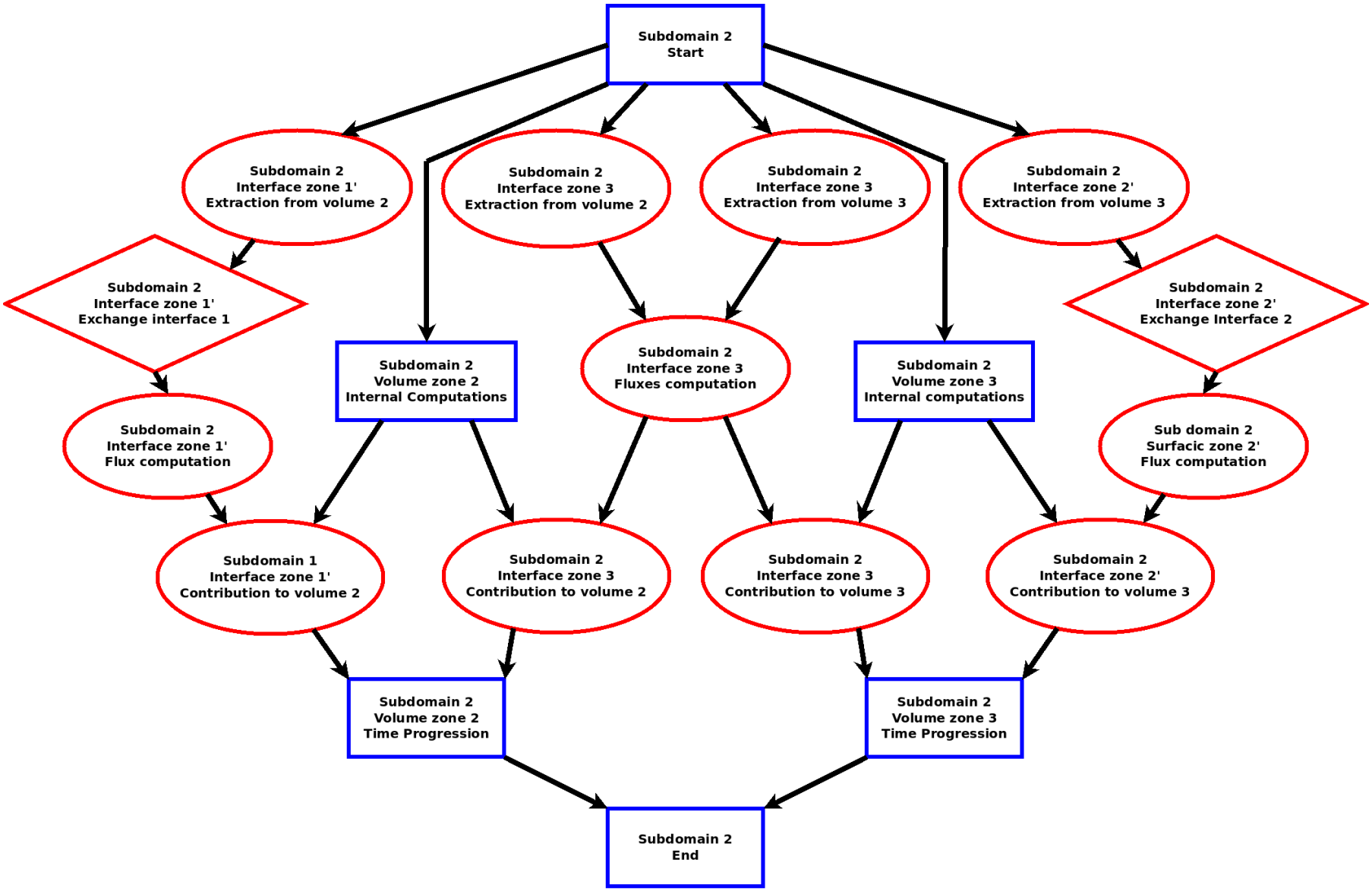
Name	Attached to	Description
Extraction	Interface	Copy or extrapolate the values of W from a neighboring volume zone
Exchange	Interface	GPU/Host transfers and MPI communication with an interface of another domain
Fluxes	Interface	Compute the fluxes at the Gauss points of the interface
Sources	Volume	Compute the internal fluxes and source terms inside a volume zone
Boundaries	Interface	Apply the fluxes of an interface to a volume zone
Time	Volume	Apply a step of the Runge-Kutta time integration to a volume zone
Start	Volume	Fictitious task: beginning of the Runge-Kutta substep
End	Volume	Fictitious task: end of the Runge-Kutta substep

Tasks graph: two domains



DAG: Direct Acyclic Graph

Tasks graph: one domain



MPI/OpenCL events management [6]

Problem: how to express the dependency between MPI and OpenCL operations ?

- We decided to rely only on the OpenCL events management.
- The beginning of a task depends on the completions of a list of OpenCL events. The task is itself associated to an OpenCL event.
- At an interface zone between two subdomains, an extraction task contains a GPU to host memory transfer, a MPI send/receive communication and a host to GPU transfer.
- we create an OpenCL user event, and launch a MPI blocking sendrecv in a thread. At the end of the communication, in the thread, the OpenCL event is marked as completed. Using threads avoids blocking the main program flow.

Simple runtime tasks management based only on well-established standards...

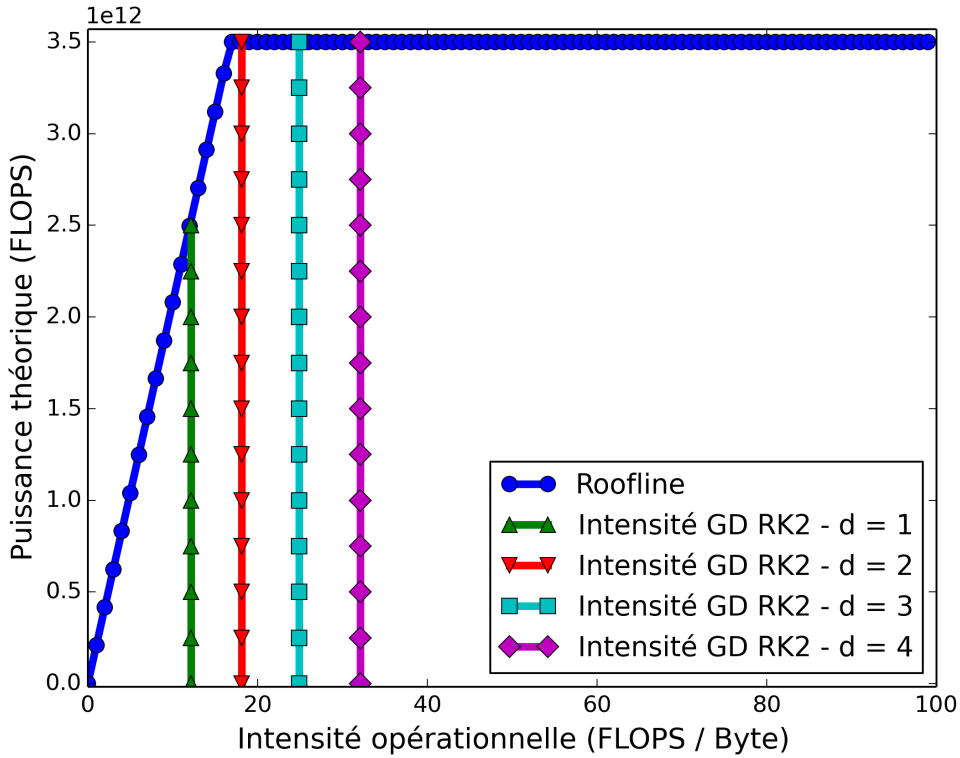
Roofline model

- Peak computation perfs: $P = 3.5\text{Tflops}$.
- Memory bandwidth: $B = 208\text{GB/s}$.
- Computational intensity of an algorithm $I = \frac{\text{number of computations}}{\text{number of memory transfers}}$.

Maximal perfs of one GPU:

$$P_A = \max(P, B \times I).$$

Roofline model



Sync./Async. comparison (T. Strub)

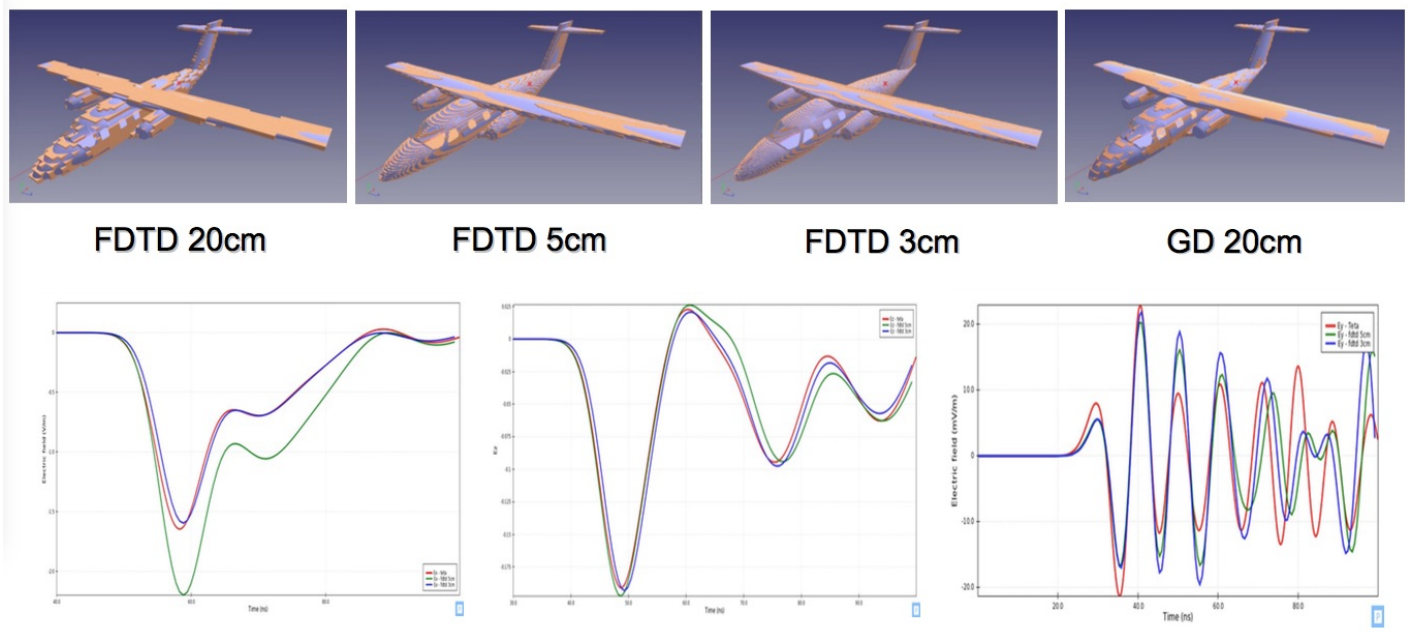
Big mesh, polynomial order $D = 3$, NVIDIA K20 GPUs, infiniband network, single-precision floats.

		1 GPU	2 GPUs	4 GPUs	8 GPUs
Sync.	TFLOPS/s	1.01	1.84	3.53	5.07
ASync.	TFLOPS/s	1.01	1.94	3.74	7.26

We achieve $\simeq 30\%$ of the peak performance.s

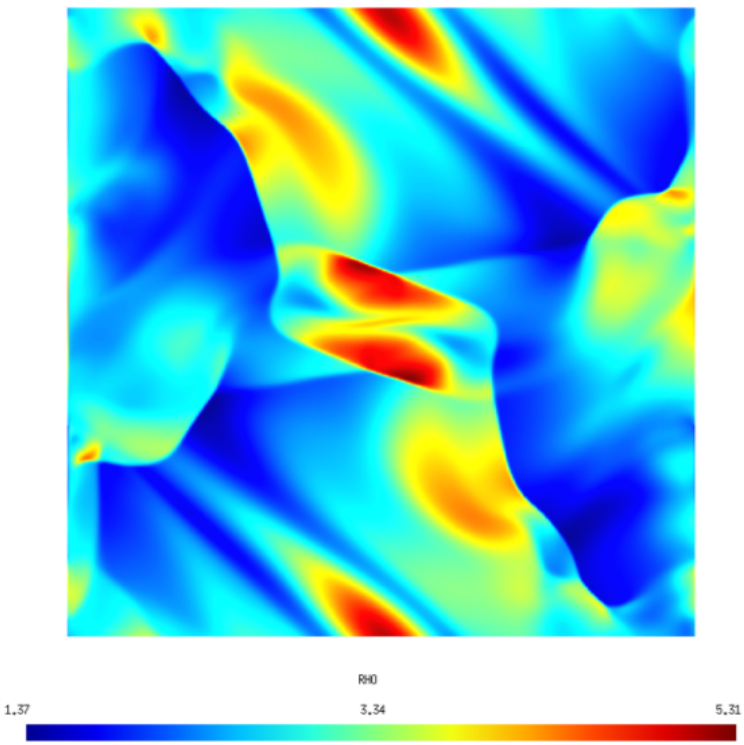
Electromagnetic compatibility application [3]

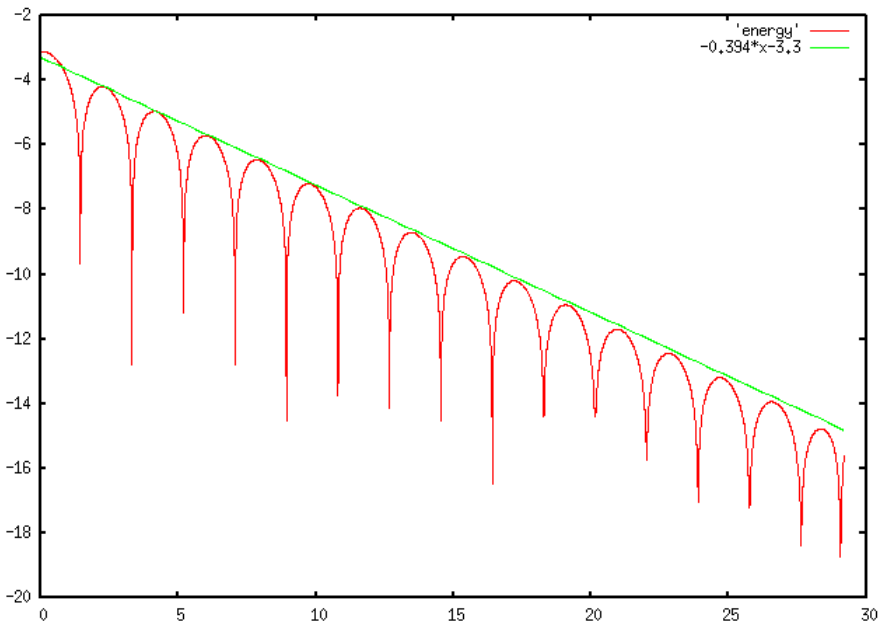
- Electromagnetic wave interaction with an aircraft.
- Aircraft geometry described with up to 3.5M hexaedrons ($\simeq$1 billion unknowns per time step): mesh of the interior and exterior of the aircraft. PML transparent boundary conditions.
- We use 8 GPUs to perform the computation. The biggest simulation does not fit into a single GPU memory.



Other applications [9, 5]

- MHD
- 4D Vlasov Landau damping





Conclusion

- Generally a huge computational power is lost in scientific software !
- Necessity of generic programming approaches: mathematicians and compter scientists collaborations
- task-graph programming model is probably the future.
- as of today lack of standard API (OpenMP ?):
  - technologies exist: StarPU [2, 1], PaRSEC, OmpSs, CHARM++, etc.
  - static or dynamic DAG, dynamic compilation, codelets, load balancing, data driven.
  - Data: automatic choice between saving or recomputing data, compression, coherency check, restart after failure.
  - Colleagues are often ready to rewrite their codes, but in a stable environment.

Bibliography

[1] Emmanuel Agullo, Luc Giraud, Abdou Guermouche, Stojce Nakov, and Jean Roman.
Task-based conjugate-gradient for multi-GPUs platforms.
2012.

[2] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier.
StarPU: a unified platform for task scheduling on heterogeneous multicore architectures.
Concurrency and Computation: Practice and Experience, 23(2):187–198, 2011.

[3] Tristan Cabel, Joseph Charles, and Stephane Lanteri.
Multi-GPU acceleration of a DGTD method for modeling human exposure to electromagnetic waves, 2011.

[4] Gary Cohen, Xavier Ferrieres, and Sébastien Pernet.
A spatial high-order hexahedral discontinuous Galerkin method to solve Maxwell's equations in time domain.
Journal of Computational Physics, 217(2):340–363, 2006.

[5] Philippe Helluy, Laurent Navoret, Nhung Pham, and Anaïs Crestetto.
Reduced Vlasov-Maxwell simulations.
Comptes Rendus Mécanique, 342(10-11):619–635, 2014.

[6] Philippe Helluy, Thomas Strub, Michel Massaro, and Malcolm Roberts.
Asynchronous OpenCL/MPI numerical simulations of conservation laws.

[7] A. Klöckner, T. Warburton, J. Bridge, and J. S. Hesthaven.
Nodal discontinuous Galerkin methods on graphics processors.
J. Comput. Phys., 228(21):7863–7882, 2009.

[8] A Kloeckner.
Hedge: Hybrid and Easy Discontinuous Galerkin Environment
`http://mathema.tician.de/software/hedge/`, 2010.

[9] Michel Massaro, Philippe Helluy, and Vincent Loechner.
Numerical simulation for the MHD system in 2D using OpenCL.
ESAIM: Proceedings and Surveys, 45:485–492, 2014.