Discontinuous Galerkin (DG) methods for hyperbolic equations

Basic theory and parallel implementations

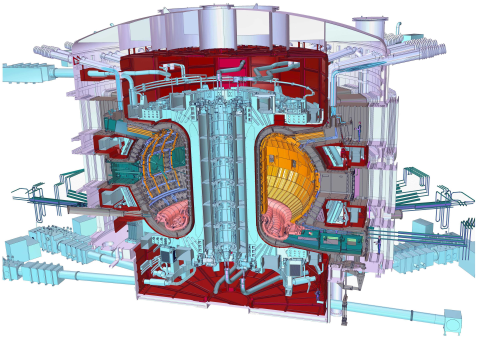
Philippe Helluy

Math. Institute Strasbourg & Inria ToNuS, France

November 20, 2017

Forewords: ITER project

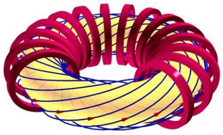
International Thermonuclear Experimental Reactor, ITER project: thermonuclear fusion in a hot hydrogen plasma (more than 100 million degrees °C). Clean energy of the future.



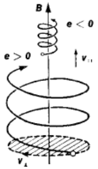
The plasma is confined in a doughnut-shaped device: a tokamak

Tokamak

Tokamak: magnetic plasma confinement in a torus. Poloidal coils $\Rightarrow$ toroidal field. Plasma current $\Rightarrow$ poloidal field $\Rightarrow$ plasma stability. Tamm-Sakharov in the 50's. Another solution: stellarator.



Strong magnetic field, few collisions: gyrokinetic turbulence.



Vlasov transport equation

- Unknown: the distribution function $f(x,v,t)$. Number of ions at point x and time t having velocity v. The problem is time-dependent in a six-dimensional phase space.
- The magnetic field B is given. Poisson equation for the electric field E and potential Φ.

$$\nabla \cdot E = \rho - \rho_0, \quad E = -\nabla \Phi.$$

  The charge ρ is given by

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

- Vlasov equation

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

  It is a **transport equation**.
- Boundary conditions.

Outlines

Discontinuous Galerkin methods

Transport equation

The simplest hyperbolic Partial Differential Equation (PDE) is the 1D transport equation

$$\partial_t w + c\partial_x w = 0.$$

- The unknown is $w(x,t) \in \mathbb{R}$. The space position $x \in \mathbb{R}$ and the time $t \in [0,T]$.
- $\partial_t = \frac{\partial}{\partial t}$, $\partial_x = \frac{\partial}{\partial x}$.
- For simplicity, we assume that the transport velocity c is constant.
- The solution has the form

  $$w(x,t) = w_0(x - ct),$$

  where $w_0(x)$ is the initial condition at time $t = 0$.
- The graph of w_0 is transported at the velocity c.

Boundary conditions

Generally one wishes to solve the transport equation for x in a finite domain $\Omega =]a, b[$.

If $c > 0$ we have to provide boundary values at $x = a$ and no condition at $x = b$

$$w(a, t) = \beta(a, t).$$

If $c < 0$ we have to provide boundary values at $x = b$ and no condition at $x = a$

$$w(b, t) = \beta(b, t).$$

Boundary conditions

In 1D the boundary of $\Omega =]a,b[$ is $\partial\Omega = \{a,b\}$.
We denote by n the unit normal vector on $\partial\Omega$ directed toward the exterior of Ω

$$n(a) = -1, \quad n(b) = 1.$$

We also use the notations $x^+ = \max(x,0)$ and $x^- = \min(x,0)$
The boundary condition can then be written

$$(c \cdot n)^- w = (c \cdot n)^- \beta$$

(it reduces to $0 = 0$ when $c \cdot n \geq 0$).

Conservation

- Conservation property

$$\frac{d}{dt}\int_{x=a}^{b} w(x,t)dx = -\int_{x=a}^{b} c\partial_x w(x,t)dx = cw(a,t) - cw(b,t).$$

- Other terminology: hyperbolic conservation law
- "Hyperbolic" ? will be explained later

Higher Dimensions

In general dimension $D \geq 1$. The transport equation reads

$$\partial_t w + c \cdot \nabla_x w = 0$$

- $x = (x^1, \ldots, x^D) \in \Omega \subset \mathbb{R}^D$, $c = (c^1, \ldots, c^D) \in \mathbb{R}^D$
- $c \cdot \nabla_x w = \sum_{i=1}^{D} c^i \partial_i w$, $\partial_i = \frac{\partial}{\partial x^i}$
- Exact solution in $\Omega = \mathbb{R}^D$: $w(x,t) = w_0(x - tc)$ (fundamental formula for Lagrangian and semi-Lagrangian solvers: LBM, PIC).
- For bounded Ω: unit outward vector $n(x)$ at $x \in \partial\Omega$. Inflow boundary: $\Gamma^- = \{x \in \partial\Omega, n(x) \cdot c < 0\}$
- Boundary conditions must be given only on the inflow boundary
- Can be written $w(n \cdot c)^- = \beta(n \cdot c)^-$ with $(n \cdot c)^- = \min(n \cdot c, 0)$.

Numerical approximation

End of the story ? at slide 12 ???
Of course not:

- Lagrangian or semi-Lagrangian methods exist, but have some drawbacks.
- How to deal with non-constant $c(x,t)$?
- Transport on complex geometries.
- High order schemes on complex geometries.
- Solving other PDEs, coupling with other PDEs.
- How to ensure exact conservation ?
- Solvers with better data locality.
- Parallel algorithms, *etc.*

Continuous Galerkin method c>0

We start with the dimension $D = 1$ and $c > 0$. Take the transport equation, multiply by a *continuous* test function $\varphi(x)$ and integrate between a and b

$$\frac{d}{dt}\int_a^b w\varphi + \int_a^b c\partial_x w\varphi = 0.$$

Integration by parts

$$\frac{d}{dt}\int_a^b w\varphi - \int_a^b cw\partial_x\varphi + cw(b,t)\varphi(b) - cw(a,t)\varphi(a) = 0.$$

Boundary condition (only at $x = a$)

$$\frac{d}{dt}\int_a^b w\varphi - \int_a^b cw\partial_x\varphi + cw(b,t)\varphi(b) - c\beta(a,t)\varphi(a) = 0.$$

We also have (with another integration by parts)

$$\frac{d}{dt}\int_a^b w\varphi + \int_a^b c\partial_x w\varphi + c(w(a,t) - \beta(a,t))\varphi(a) = 0.$$

Continuous Galerkin method

For general c and $\Omega =]a,b[$ we obtain (flux formulation)

$$\frac{d}{dt}\int_\Omega w\varphi - \int_\Omega w\partial_x\varphi + \int_{\partial\Omega}\varphi\left\{(c\cdot n)^+ w_L + (c\cdot n)^- w_R\right\} = 0$$

where, on $\partial\Omega$, w_L denotes the value of w inside Ω and w_R the outside value (given by the boundary condition).
In other words, if $x = a$ or $x = b$

$$\begin{array}{rcl} w_L(x,t) & = & \lim\limits_{\varepsilon\to 0,\varepsilon<0} w(x+\varepsilon n(x),t) = w(x,t), \\ w_R(x,t) & = & \beta(x,t). \end{array}$$

We also have (fluctuation formulation)

$$\frac{d}{dt}\int_\Omega w\varphi + \int_\Omega \partial_x w\varphi + \int_{\partial\Omega}\varphi(c\cdot n)^-(w_R - w_L) = 0$$

The two formulations can be used now for $D > 1$.

Conservation laws

Many equations in physics are systems of conservation laws:

$$\partial_t w + \partial_i F^i(w) = 0.$$

- $w = w(x,t) \in \mathbb{R}^m$: vector of conserved quantities;
- $x = (x^1 \dots x^D)$: space variable, D: space dimension, t: time;
- $\partial_t = \frac{\partial}{\partial t}$, $\partial_i = \frac{\partial}{\partial x_i}$;
- sum on repeated indices: $\partial_i F^i$ means $\sum_{i=1}^{D} \partial_i F^i$;
- $F^i(w)$: flux vector (contains the physics).

Applications: fluid mechanics, electromagnetics, and … transport equation !

$$m = 1, \quad F^i(w) = c^i w.$$

Hyperbolicity

The model is mathematically stable under a hyperbolicity condition:
Consider the jacobian matrix

$$A^i(w) = \nabla_w F^i(w).$$

It is an $m \times m$ matrix defined by

$$(A^i)^k_\ell(w) = \frac{\partial}{\partial w^\ell}\left(F^i(w)\right)^k, \quad 1 \le k \le m, \quad 1 \le \ell \le m.$$

We also define

$$A(w,n) = A^i n_i = \nabla_w F^i(w) n_i.$$

The system is hyperbolic iff $A(w,n)$ is diagonalizable with real eigenvalues $\lambda_1 \le \lambda_2 \ldots \le \lambda_m$.
Transport equation: $m = 1$ and $\lambda_1 = c$.

Generalization of the CG method

We can still use the CG formulation for a general system of conservation laws

$$\frac{d}{dt}\int_{\Omega} w\varphi - \int_{\Omega} F^i(w)\partial_i\varphi + \int_{\partial\Omega} \varphi F(w_L, w_R, n) = 0$$

$F(w_L, w_R, n)$ is the numerical flux. It satisfies

- $F(w, w, n) = F^i(w)n_i$
- $F(w_L, w_R, n) = -F(w_R, w_L, -n)$.

For transport we have the so-called upwind numerical flux:

$$F(w_L, w_R, n) = (c\cdot n)^+ w_L + (c\cdot n)^- w_R$$

We indeed check that:

- $F(w, w, n) = ((c\cdot n)^+ + (c\cdot n)^-)w = c\cdot n w$
- $F(w_R, w_L, -n) = (-c\cdot n)^+ w_R + (-c\cdot n)^- w_L = F(w_L, w_R, n)$

Finite elements mesh

Go back to $D = 1$. We consider the subdivision

$$a = x_0 < x_1 \ldots < x_{N_e} = b.$$

For $0 \leq i < N_e$, Element L corresponds to the interval

$$L = [x_i, x_{i+1}] \quad \text{(Notation: } L = i\text{)}.$$

We also consider the reference element

$$\hat{L} = [-1, 1]$$

and a linear mapping from the reference element $\hat{L}$ to element L

$$\tau_L(\hat{x}) = x(\hat{x}) = x_i + \frac{\hat{x} + 1}{2}(x_{i+1} - x_i).$$

This mapping has a constant jacobian

$$\tau' = \frac{dx}{d\hat{x}} = \frac{x_{i+1} - x_i}{2}$$

and can be reversed

$$\hat{x} = 2\frac{x - x_i}{x_{i+1} - x_i} - 1.$$

Gauss-Lobatto points

On the reference element, we can consider the order d Gauss-Lobatto points and associated weights

$$\xi_q, \quad \omega_q, \quad q = 0 \ldots d$$

(d (order) should not be confused with D (space dimension))
Those points are such that $\xi_0 = -1$, $\xi_d = 1$ and

$$\int_{-1}^{1} P(t)dt = \sum_{q=0}^{d} \omega_q P(\xi_q)$$

when P is a polynomial of degree $\leq 2d-1$
We could also use Gauss-Legendre points (exact for polynomial of order $\leq 2d+1$ but then $\xi_0 > -1$ and $\xi_d < 1$.)

Numerical integration

For instance if $d = 1$ (trapezoidal rule)

$$\xi_0 = -1, \quad \xi_1 = 1, \quad \omega_0 = 1, \quad \omega_1 = 1;$$

for $d = 2$ (Simpson's rule)

$$\xi_0 = -1, \quad \xi_1 = 0, \quad \xi_2 = 1, \quad \omega_0 = 1/3, \quad \omega_1 = 4/3 \quad \omega_2 = 1/3;$$

for $d = 3$ (First non equally spaced Gauss Lobatto rule)

$$\xi_0 = -1, \quad \xi_1 = -1/\sqrt{5}, \quad \xi_2 = 1/\sqrt{5}, \quad \xi_3 = 1,$$

$$\omega_0 = 1/6, \quad \omega_1 = 5/6 \quad \omega_2 = 5/6, \quad \omega_3 = 1/6.$$

etc.

Lagrange interpolation

To the Gauss-Lobatto points we can associate Lagrange interpolation polynomials

$$l_j(\hat{x}) = \prod_{\substack{0 \le k \le d \\ k \ne j}} \frac{\hat{x} - \xi_k}{\xi_j - \xi_k}, \quad j = 0 \ldots d.$$

The Lagrange polynomials are of order d and satisfy the interpolation property

$$l_j(\xi_j) = \delta_{ij} = \begin{cases} 1 & \text{if} \quad i = j, \\ 0 & \quad \text{otherwise.} \end{cases}$$

Mesh nodes

The nodes of the finite element mesh are obtained by mapping the Gauss-Lobatto points with τ_L. The number of nodes is $N_n = dN_e + 1$. The nodes are denoted y_i, $i = 0 \dots N_n - 1$. We also use a local numbering in each finite element L. In the end, we have the relation

$$y_k = \tau_L(\xi_i),$$

with

$$k = \mathrm{connec}(L, i) = dL + i.$$

The table connec is the finite element connectivity array.

Nodal Approximation space

We now construct the continuous basis functions φ_k. They are associated to the finite element nodes y_k and satisfy the interpolation property

$$\varphi_i(y_j) = \delta_{ij}.$$

They are defined as follows: let $x \in [a,b]$ then x necessarily belongs to one (or more) element L. If node k is not a node of L then

$$\varphi_k(x) = 0,$$

otherwise, there is a local node ξ_i of L such that

$$k = \text{connec}(L, i)$$

and then

$$\varphi_k(x) = l_i(\tau_L^{-1}(x)).$$

(see Maple demo.)

Continuous Galerkin (CG) scheme

Now we construct the Galerkin approximation. We expand w on the basis function

$$w(x,t) \simeq \sum_j w_j(t)\varphi_j(x)$$

and insert the expansion in the CG formulation with $\varphi = \varphi_i$. We obtain

$$\sum_j \int_\Omega \varphi_i \varphi_j \partial_t w_j - \int_\Omega F(\sum_j w_j \varphi_j)\varphi_i' + \int_{\partial\Omega} F(\sum_j w_j \varphi_j, \beta, n)\varphi_i = 0.$$

If we use a Gauss-Lobatto quadrature and the interpolation property, we obtain

$$\omega_L^i \partial_t w_i - \sum_k \omega_L^k F(w_k)\varphi_i'(y_k) + \\ F(w_1, \beta, -1)\delta_{1,i} + F(w_{N_n}, \beta, 1)\delta_{N_n,i} = 0$$

where ω_L^k are the mapped quadrature weights

$$\omega_L^k = \omega_\ell \frac{\tau_L'}{d\hat{x}} \quad k = \text{connec}(L,\ell)$$

System of differential equation

Define the big vector

$$W(t) = \begin{bmatrix} w_1(t) \\ \vdots \\ w_{N_n}(t) \end{bmatrix}$$

We end up with a system of differential equations

$$W'(t) = G(W(t))$$

Time integration

Any method for time integration could be used. Time step Δt.
Discrete times $t_p = p\Delta t,\ p = 0, 1, 2, \ldots$

$$W^p \simeq W(p\Delta t)$$

Taylor expansion

$$W(t+\Delta t) = W(t) + \Delta t W'(t) + o(\Delta t)$$

This lead to the explicit Euler scheme

$$W^{p+1} = W^p + \Delta t G(W^p)$$

Never stable !
Implicit scheme

$$W^{p+1} = W^p + \Delta t G(W^{p+1})$$

Unconditionaly stable but requires solving a linear system at each time-step

Higher order, CFL condition

Other possibilities. For instance second order explicit Runge-Kutta scheme (RK2):

$$W^* = W^p + \frac{\Delta t}{2} G(W^p), \quad W^{p+1} = W^p + \Delta t G(W^*)$$

Also: explicit RK4, Adams-Bashforth, *etc.*
The explicit schemes require a stability CFL condition

$$\Delta t \leq C \frac{\Delta x}{\lambda_{\max}}$$

where Δx is the maximal size of the elements, $\lambda_{\max}$ is the maximal wave speed and C is a constant (depending on the method, d, *etc.*).

Numerical results

- Small demo (implicit, various d, first order in time and then second order in time)
- Convergence rate for smooth solution (and Δt very small) $\|\text{error}\|_{L^2(\Omega)} \simeq C(w)\Delta x^d$.
- Oscillations with discontinuous solutions (demo).

Discontinuous Galerkin

We can also solve the transport equations with a DG solver. The unknown numbering is modified in the following way. First

$$k = \mathrm{connec}(L, i) = (d+1)L + i$$

the number of nodes is thus

$$N_n = (d+1)N_e.$$

The basis functions are also defined in a different way. Now they are associated to a given element L and a local node $i = 0 \ldots d$ and

$$\varphi_k(x) = \varphi_{L,i}(x) = l_i(\tau_L^{-1}(x)) \text{ if } x \in L, \quad \varphi_{L,i}(x) = 0 \text{ otherwise.}$$

The basis functions are discontinous at the element boundaries x_i. (see Maple demo.)

Discontinuous Galerkin (DG) formulation

We expand w in this discontinuous basis function in each element L

$$w(x,t) \simeq \sum_{j=0}^{d} w_{L,j}(t)\varphi_{L,j}(x), \quad x \in L. \tag{1}$$

The DG formulation is simply obtained **by using the CG formulation independently on each element**

$$\int_L \partial_t w \varphi_{L,i} - \int_L F(w)\partial_x \varphi_{L,i} + \\ F(w_{L,d}, w_{L+1,0}, 1)\varphi_{L,i}(x_{L+1}) - F(w_{L-1,d}, w_{L,0}, 1)\varphi_{L,i}(x_L) = 0,$$

where $F(w_L, w_R, n)$ is the numerical flux.
In addition, in the DG formulation we have

$$\varphi_{L,i}(x_{L+1}) = \delta_{i,d}, \quad \varphi_{L,i}(x_L) = \delta_{i,0},$$

This shows that coupling only occurs at the element boundary nodes (this would not be true anymore for Gauss-Legendre integration).

Bibliography

- [Reed and Hill, 1973]
- [Lesaint and Raviart, 1974]
- [Johnson et al., 1984]
- [Bourdel et al., 1992]
- [Cockburn and Shu, 1998]
- [Barth, 2006]
- [Hesthaven and Warburton, 2007]
- [Zingan et al., 2013]

Numerical results

- As for the CG method we have to solve a system of differential equations
$$W'(t) = G_d(W(t)).$$
Same kind of CFL conditions for explicit time solver.
- Small demo (implicit, various d, first order in time and then second order in time)
- Convergence rate for smooth solution and regular grids (Δt very small) $\|\text{error}\|_{L^2(\Omega)} \simeq C(w)\Delta x^{d+1}$.
- Less oscillations with discontinuous solutions than for the CG method (demo).
- Nice feature of the upwind flux for transport: the implicit Euler scheme
$$W^{p+1} = W^p + \Delta t G(W^{p+1})$$
can be solved without inverting a global linear system.

Upwind implicit DG solver (flux version)

We assume $c > 0$ and we use the notation $w = w^p$ (p: time index)

$$\int_L \frac{w - w^{p-1}}{\Delta t}\varphi_{L,i} - \int_L cw\partial_x\varphi_{L,i} + cw_{L,d}\delta_{i,d} - cw_{L-1,d}\delta_{i,0} = 0.$$

Applying Gauss-Lobatto integration, we obtain

$$\tau'\omega_i \frac{w_{L,i} - w_{L,i}^{p-1}}{\Delta t} - \sum_{k=0}^{d} cw_{L,k}\omega_k l_i'(\xi_k) + cw_{L,d}\delta_{i,d} - cw_{L-1,d}\delta_{i,0} = 0.$$

We observe that if we know $w_{L-1,d}$ then we can compute directly all the values of $w_{L,i}$ for $i = 0 \ldots d$ by solving a small $(d+1)\times(d+1)$ linear system.

Upwind DG solver for transport

It is also possible to write the same system, but without an integration by parts. We obtain

$$\int_L \partial_t w \varphi_{L,i} + \int_L c\partial_x w \varphi_{L,i} + (cw_{L,0} - cw_{L-1,d})\varphi_{L,i}(x_L) = 0.$$

Then we apply Gauss-Lobatto integration and time discretisation

$$\omega_i \tau' \frac{w_{L,i} - w_{L,i}^{p-1}}{\Delta t} + c\sum_{j=0}^{d} w_{L,j}\partial_x \varphi_j(\tau_L(\xi_i))\omega_i \tau' + (cw_{L,0} - cw_{L-1,d})\delta_{i,0} = 0.$$

$$w_{L,i} + \frac{c\Delta t}{\tau'}\left(\sum_{j=0}^{d} l_j'(\xi_i) w_{L,j} + \frac{1}{\omega_0} w_{L,0}\delta_{i,0}\right) = w_{L,i}^{p-1} + \frac{c\Delta t}{\omega_0 \tau'} w_{L-1,d}\delta_{i,0}$$

Local linear system

We define the $(d+1)\times(d+1)$ matrix Λ by

$$\Lambda_{ij} = l_j'(\xi_i) + \frac{\delta_{0,j}\delta_{0,i}}{\omega_0}.$$

In element L we consider the right hand side vector

$$r = (w_{L,0}^{p-1} + \frac{c\Delta t}{\omega_0 \tau'} w_{L-1,d}, w_{L,1}^{p-1}, \dots, w_{L,d}^{p-1})^T$$

(if element $L-1$ does not exist we replace $w_{L-1,d}$ by the (upwind) boundary condition). Then the vector

$$w = (w_{L,0}, \dots, w_{L,d})^T$$

is solution of

$$\left(I + \frac{c\Delta t}{\tau'}\Lambda\right) w = r.$$

Solving this small linear system step by step from $L=0$ to $L = N_e - 1$ gives the solution of the implicit time step.

Nodal DG on GPU with OpenCL

Conservation laws

Many equations in physics are systems of conservation laws:

$$\partial_t w + \partial_i F^i(w) = 0.$$

- $w = w(x,t) \in \mathbb{R}^m$: vector of conserved quantities;
- $x = (x^1 \dots x^D)$: space variable, D: space dimension, t: time;
- $\partial_t = \frac{\partial}{\partial t}$, $\partial_i = \frac{\partial}{\partial x_i}$;
- sum on repeated indices: $\partial_i F^i$ means $\sum_{i=1}^{D} \partial_i F^i$;
- $F^i(w)$: flux vector (contains the physics).

Applications: fluid mechanics, electromagnetics, and … transport equation !

$$m = 1, \quad F^i(w) = c^i w.$$

FDTD on a 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:

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

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

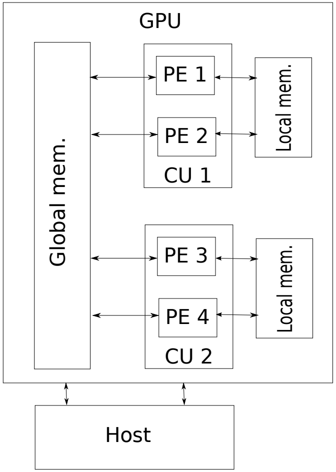
- Numerical flux: $F(w_L, w_R, n)$, $\quad F(w, w, n) = F(w) \cdot n$.

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

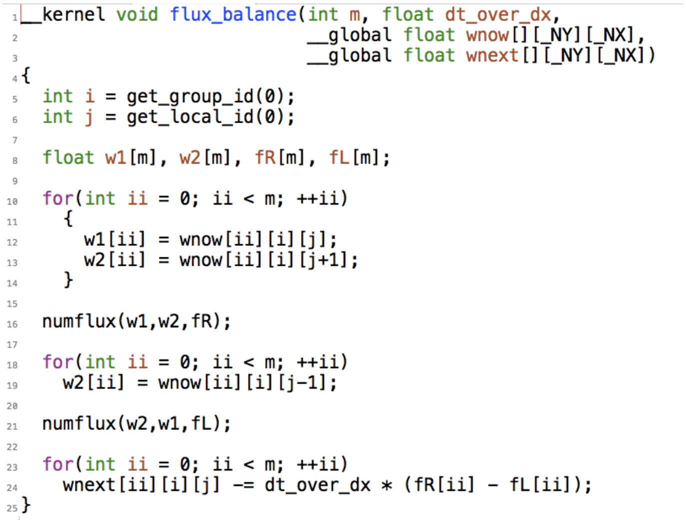
- The local (cache) memory is small but fast.
- The global memory is bigger but slower.
- Accessing the global memory of the GPU is faster if neighboring processors access neighboring locations (“coalescent” access).
- Accessing the host memory is very slow.
- Branching may be costly (SIMD parallelism).
- 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.

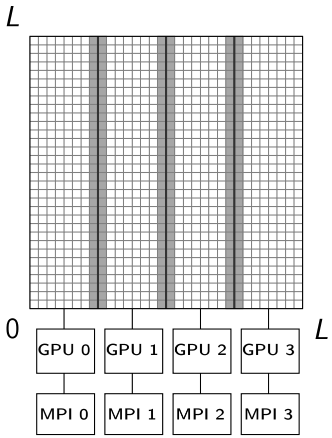
Kernel code



```
__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

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

- 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
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

[Helluy and Jung, 2014]

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

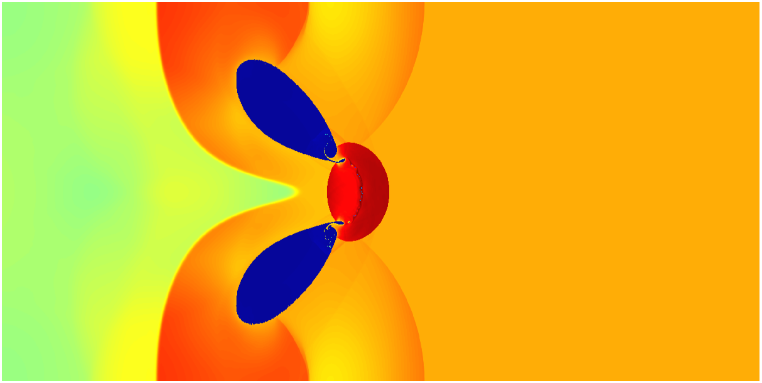
$$F(w) \cdot n = (\rho u \cdot n, \rho (u \cdot n) u^T + p n^T, (\rho Q + p) u \cdot n, \rho \varphi u \cdot n)^T.$$

- 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.

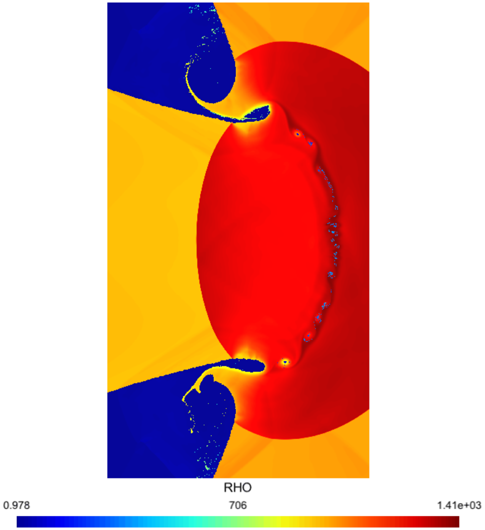
`https://www.youtube.com/watch?v=c8hcqihJzbw`

Very fine mesh

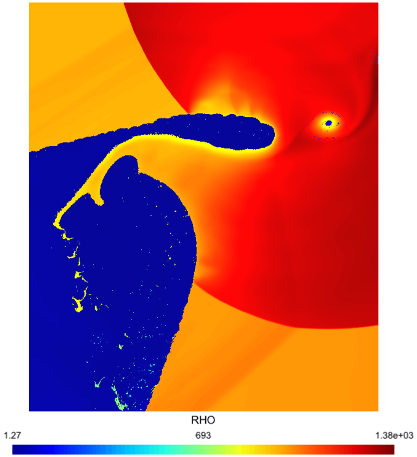
- Very fine mesh OpenCL + MPI simulation, 40,000x20,000 grid. 4 billions unknowns per time step
- 10xNVIDIA K20 GPUs, 30 hours
- Red=high density (compressed liquid); blue=low density (gas).

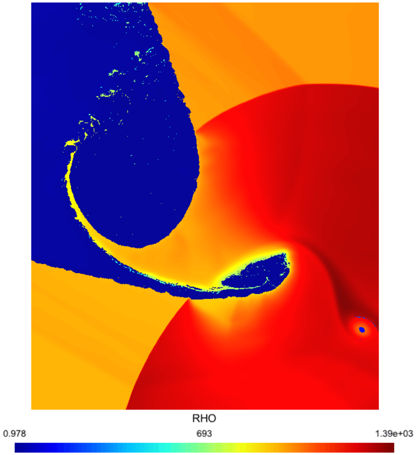


Zoom 1



Zoom 2





Discontinous Galerkin

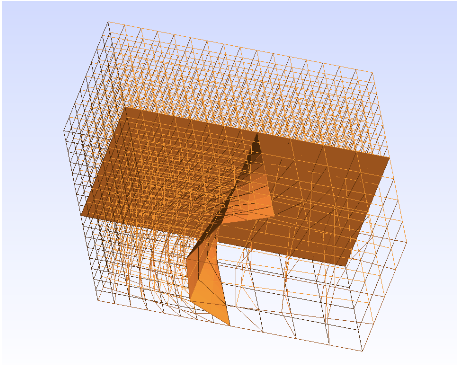
[Strub, 2015], joint work with AxesSim company.

- Unstructured hexahedrons 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.

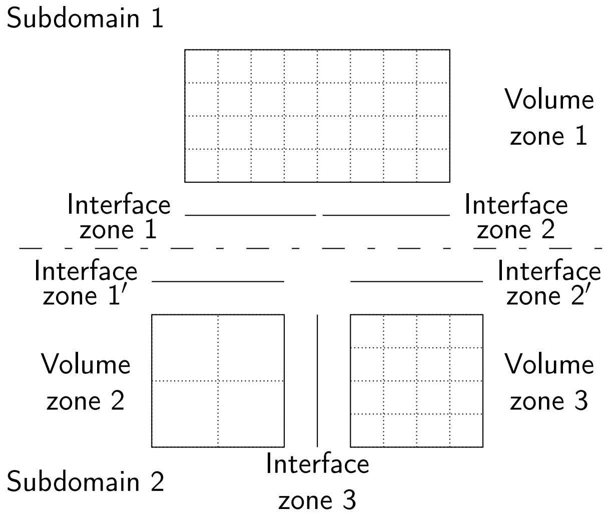
Mesh example

A non-conforming mesh with two subdomains, three volume zones and three interface zones (created with gmsh `http://gmsh.info`).

- 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

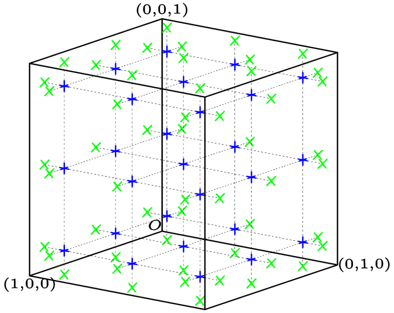


Discontinuous Galerkin (DG) approximation

DG method in a 3D space.
In each cell L of the mesh, the conserved quantities are expanded on 3D mapped Lagrange polynomial basis functions

$$w(x,t) = w_L^j(t)\varphi_j^L(x), \quad x \in L.$$

- L is a (possibly stretched) hexahedron. $\hat{L}$ is a cube.
- w is determined by its values at blue volume Gauss points
- w is discontinuous at green faces Gauss points.

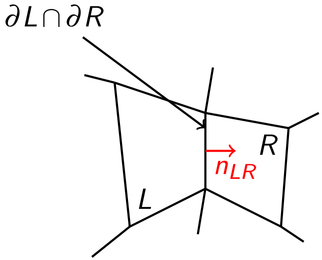


DG formulation

The numerical solution satisfies the DG approximation scheme

$$\forall L, \forall i \quad \int_L \partial_t w \varphi_i^L - \int_L F(w, w, \nabla \varphi_i^L) + \int_{\partial L} F(w_L, w_R, n_{LR}) \varphi_i^L = 0.$$

- 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.
- $F(w, w, n) = F^k(w) n_k$.

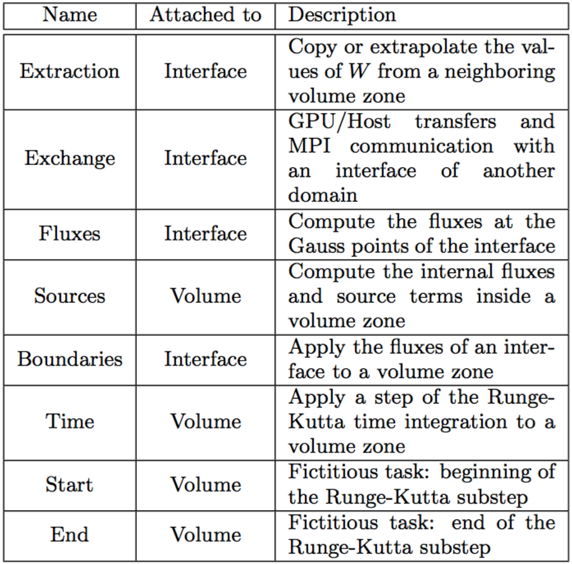


Time integration of a system of ordinary differential equations.

Tasks

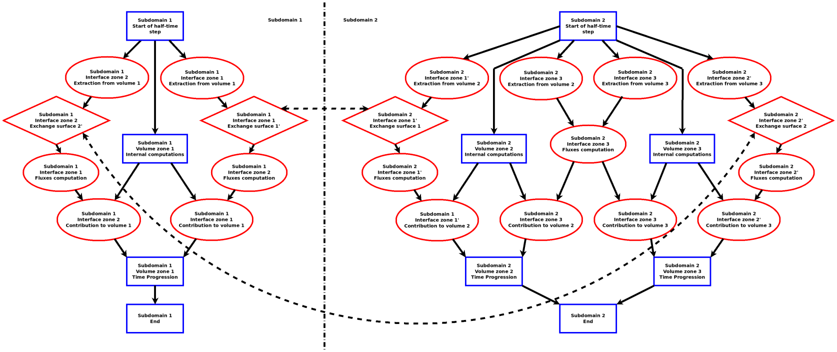
- Elementary tasks attached to volume or interface zones
- A task is associated to a computational OpenCL kernel or to memory operations (GPU$\leftrightarrow$CPU and MPI transfers).
- The optimized design of the computational kernels is much more tricky than for the FD method...
  - Hexahedra mesh optimizations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

Tasks

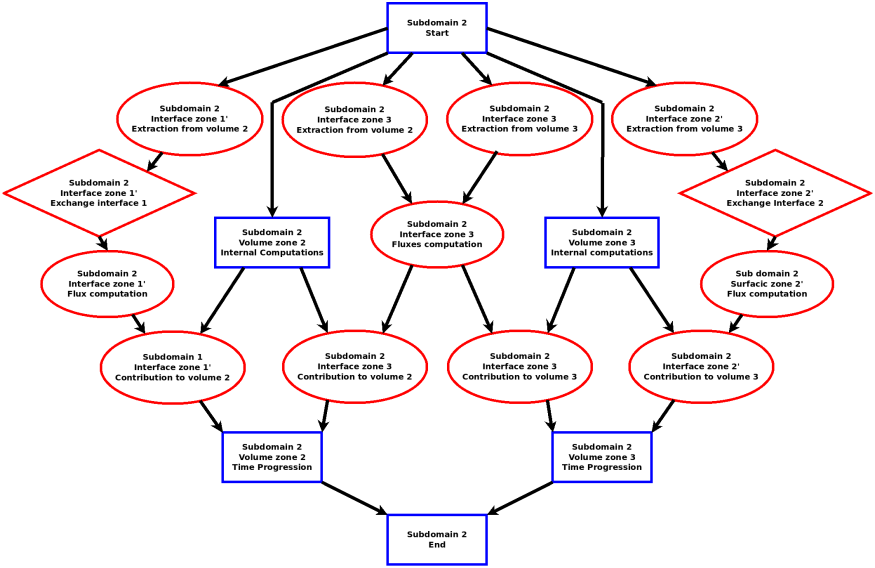


Name	Attached to	Description
Extraction	Interface	Copy or extrapolate the values of W from a neighboring volume zone
Exchange	Interface	GPU/Host transfers and MPI communication with an interface of another domain
Fluxes	Interface	Compute the fluxes at the Gauss points of the interface
Sources	Volume	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


Subdomain 1
Subdomain 2
Subdomain 1 Start of half-time step
Subdomain 1 Interface zone 2 Extraction from volume 1
Subdomain 1 Interface zone 1 Extraction from volume 1
Subdomain 1 Interface zone 2 Exchange surface 2'
Subdomain 1 Volume zone 1 Internal computations
Subdomain 1 Interface zone 1 Exchange surface 1'
Subdomain 1 Interface zone 1 Fluxes computation
Subdomain 1 Interface zone 2 Fluxes computation
Subdomain 1 Interface zone 2 Contribution to volume 1
Subdomain 1 Interface zone 1 Contribution to volume 1
Subdomain 1 Volume zone 1 Time Progression
Subdomain 1 End
Subdomain 2 Start of half-time step
Subdomain 2 Interface zone 1' Extraction from volume 2
Subdomain 2 Interface zone 3 Extraction from volume 2
Subdomain 2 Interface zone 3 Extraction from volume 3
Subdomain 2 Interface zone 2' Extraction from volume 3
Subdomain 2 Interface zone 1' Exchange surface 1
Subdomain 2 Volume zone 2 Internal computations
Subdomain 2 Interface zone 3 Fluxes computation
Subdomain 2 Volume zone 3 Internal computations
Subdomain 2 Interface zone 2' Exchange surface 2
Subdomain 2 Interface zone 1' Fluxes computation
Subdomain 2 Interface zone 2' Fluxes computation
Subdomain 1 Interface zone 1' Contribution to volume 2
Subdomain 2 Interface zone 3 Contribution to volume 2
Subdomain 2 Interface zone 3 Contribution to volume 3
Subdomain 2 Interface zone 2' Contribution to volume 3
Subdomain 2 Volume zone 2 Time Progression
Subdomain 2 Volume zone 3 Time Progression
Subdomain 2 End

Tasks graph: one domain


Subdomain 2 Start
Subdomain 2 Interface zone 1' Extraction from volume 2
Subdomain 2 Interface zone 3 Extraction from volume 2
Subdomain 2 Interface zone 3 Extraction from volume 3
Subdomain 2 Interface zone 2' Extraction from volume 3
Subdomain 2 Interface zone 1' Exchange interface 1
Subdomain 2 Volume zone 2 Internal Computations
Subdomain 2 Interface zone 3 Fluxes computation
Subdomain 2 Volume zone 3 Internal computations
Subdomain 2 Interface zone 2' Exchange Interface 2
Subdomain 2 Interface zone 1' Flux computation
Sub domain 2 Surfacic zone 2' Flux computation
Subdomain 1 Interface zone 1' Contribution to volume 2
Subdomain 2 Interface zone 3 Contribution to volume 2
Subdomain 2 Interface zone 3 Contribution to volume 3
Subdomain 2 Interface zone 2' Contribution to volume 3
Subdomain 2 Volume zone 2 Time Progression
Subdomain 2 Volume zone 3 Time Progression
Subdomain 2 End

MPI/OpenCL events management

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 for the poor !

Sync./Async. comparison

Big mesh, polynomial order $d = 3$, NVIDIA K20 GPUs, infiniband network.

		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

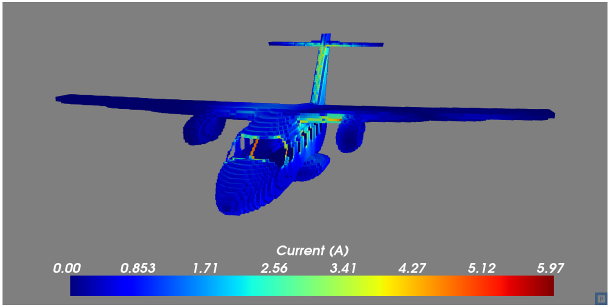
We achieve $\simeq 30\%$ of the peak performance.s

$$w = \left(E^1, E^2, E^3, H^1, H^2, H^3\right)^T, \quad m = 6, \quad D = 3.$$

$$F(w) \cdot n = \begin{pmatrix} -n \times H \\ n \times E \end{pmatrix}.$$

Application

- Electromagnetic wave interaction with an aircraft (Maxwell equations).
- Aircraft geometry described with $3,337,875$ hexaedrons ($\simeq$1 billion unknowns per time step): mesh of the interior and exterior of the aircraft.
- We use 8 GPUs to perform the computation. The simulation does not fit into a single GPU memory.



ADER approaches

Local time-stepping

Explicit Discontinuous Galerkin (DG) are constrained by an annoying CFL condition. Empirical stability condition

$$\Delta t \leq \frac{\Delta x}{2D(2d+1)V_{\max}}$$

with:

- Δx: cell size.
- D: space dimension
- d : polynomial degree
- $V_{\max}$: maximal speed
- Can be worse...

Is it possible to use different time steps on element of different sizes?

DG scheme

The approximate solution is solution of the DG scheme: for all element $L =]x_k, x_{k+1}[$ and all basis function φ_i^L

$$\begin{aligned} \int_L \partial_t w \varphi_i^L - \int_L A w \partial_x \varphi_i^L + \\ (A^+ w(x_{k+1}^-, t) + A^- w(x_{k+1}^+, t))\varphi_i^L(x_{k+1}^-) + \\ (-A^- w(x_k^+, t) - A^+ w(x_k^-, t))\varphi_i^L(x_k^+) &= 0. \end{aligned}$$

$$\begin{aligned} \int_L \frac{d}{dt} w_j^k \varphi_j^L \varphi_i^L - \int_L A w_j^k \varphi_j^L \partial_x \varphi_i^L + \\ (A^+ w_d^k + A^- w_0^{k+1})\delta_{i,d} + \\ (-A^- w_0^k - A^+ w_d^{k-1})\delta_{i,0} &= 0. \end{aligned} \tag{2}$$

Uncoupling predictor

In order to remove the coupling between elements $]x_{k-1},x_k[$, $]x_k,x_{k+1}[$, $]x_{k+1},x_{k+2}[$ we consider an approximate differential equation (also called "predictor")

$$\begin{aligned}\int_L \frac{d}{dt} v_j^k \varphi_j^L \varphi_i^L - \int_L A v_j^k \varphi_j^L \partial_x \varphi_i^L + \\ (A^+ v_d^k + A^- v_d^k)\delta_{i,d} + \\ (-A^- v_0^k - A^+ v_0^k)\delta_{i,0} &= 0.\end{aligned}$$

Space-time DG

Suppose that we know $w_j^k(t=\alpha)$. For evaluating $w_j^k(t=\beta)$ we integrate (2) from $t=\alpha$ to $t=\beta$

$$\begin{aligned}\left(\int_L \varphi_j^L \varphi_i^L\right)\left(w_j^k(\beta)-w_j^k(\alpha)\right)-\int_{t=\alpha}^{\beta}\int_L A v_j^k \varphi_j^L \partial_x \varphi_i^L + \\ \int_{t=\alpha}^{\beta}(A^+ v_d^k + A^- v_0^{k+1})\delta_{i,d} + \\ \int_{t=\alpha}^{\beta}(-A^- v_0^k - A^+ v_d^{k-1})\delta_{i,0} &= 0.\end{aligned} \tag{3}$$

Comments

- We can now advance each element with its own time step: simply use the most recent predictors of the neighbour elements.
- Allows to perform smaller time steps on smaller elements. Interesting when we have geometrical mesh constraints.
- Interesting even in the case of a common time step: better data locality in the predictor step.
- Many different possibilities for the predictor: local RK, Adams-Bashforth, Cauchy-Kowaleska procedure, nilpotent space-time discretisation *etc.*
- Large gain in CPU time when the mesh contains very few small elements. A polyhedral paradise ?

(small demo, figures ?)

Parallel transport solver

Implicit DG solver for transport

Explicit Discontinuous Galerkin (DG) are constrained by an annoying CFL condition. Empirical stability condition Explicit Discontinuous Galerkin (DG) are constrained by an annoying CFL condition. Empirical stability condition

$$\Delta t \leq \frac{\Delta x}{2D(2d+1)V_{\max}}$$

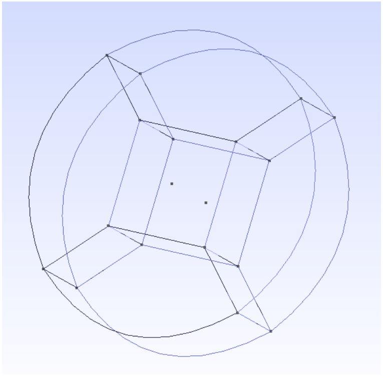
with:

- Δx: cell size.
- D: space dimension
- d : polynomial degree
- $V_{\max}$: maximal speed
- Can be worse...

3D mesh

We want to solve $\partial_t w + c \cdot \nabla w = 0$
We consider a coarse mesh made of hexahedral curved macrocells

- Each macrocell is itself split into smaller subcells of diameter $\simeq \Delta x$.
- In each subcell L we consider polynomial basis functions φ_i^L of degree d.
- Possible non-conformity in "Δx" and "d".
- We need a conservative scheme. We want to avoid CFL condition.

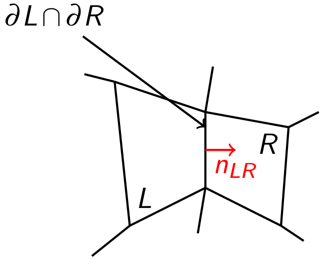


DG approximation of the transport equation

Implicit scheme for going from time $p-1$ to time p:
$\forall L, \forall i$

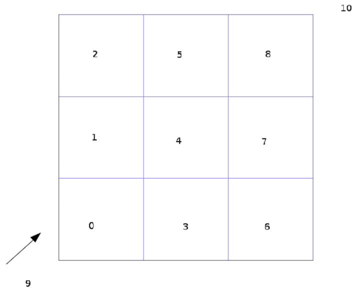
$$\int_L \frac{w^p - w^{p-1}}{\Delta t}\varphi_i^L - \int_L w^p c \cdot \nabla \varphi_i^L + \int_{\partial L} \left(c \cdot n^+ w_L^p + c \cdot n^- w_R^p\right) \varphi_i^L = 0.$$

- R denotes the neighbor cells along ∂L.
- $c \cdot n^+ = \max(c \cdot n, 0)$,
  $c \cdot n^- = \min(c \cdot n, 0)$.
- n_{LR} is the unit normal vector on ∂L oriented from L to R.



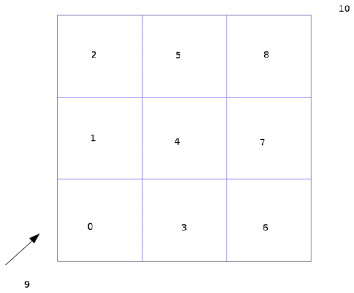
Upwind numbering

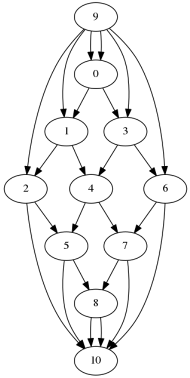
- L is *upwind* with respect to R if $c \cdot n_{LR} > 0$ on $\partial L \cap \partial R$.
- In a macrocell, the solution depends only on the values of w in the upwind macrocells.
- No assembly and factorization of the global system. Only local-to-macrocell linear systems.
- Example with a simple mesh (curved meshes are also allowed):



Dependency graph

For a given velocity c we can build a dependency graph. Vertices are associated to macrocells and edges to macrocells interfaces or boundaries. We consider two fictitious additional vertices: the "upwind" vertex and the "downwind" vertex.





Algorithm

- Topological ordering of the dependency graph.
- First time step: Assembly and LU decomposition of the macrocells local matrices.
- For each macrocell (in topological order):
    - Compute volume terms.
    - Compute upwind fluxes.
    - Solve the local linear system.
    - Extract the results to the downwind macrocells.

Bibliography: [Duff and Reid, 1978, Johnson et al., 1984, Wang and Xu, 1999, Natvig and Lie, 2008]
Parallelization ?

StarPU parallelization

- StarPU is a library developed at Inria Bordeaux [Augonnet et al., 2012]: `http://starpu.gforge.inria.fr`
- Task-based parallelism.
- Task description: codelets, inputs (R), outputs (W or RW).
- The user submits tasks in a correct sequential order.
- StarPU schedules the tasks in parallel if possible.

StarPU implementation

- We start from a working sequential code `http://schnaps.gforge.inria.fr`
- StarPU implementation was smooth: incremental migrations task by task.
- Several implementations of the same task are possible (CPU, optimized CPU, GPU I, GPU II, MIC, etc.)

Preliminary results

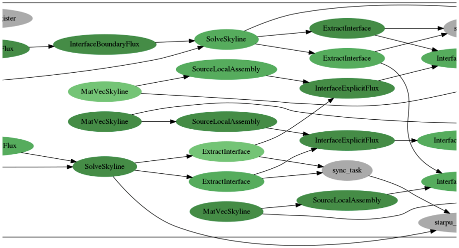
We compare a global direct solver to the upwind StarPU solver with several meshes.
Weak scaling. “dmda” scheduler. AMD Opteron 16 cores, 2.8 Ghz.
Timing in seconds for 200 iterations.

nb cores	0	1	2	4	8	16
$10 \times 10 \times 8 \times 8$ direct	30	144	-	-	-	-
$10 \times 10 \times 8 \times 8$ upwind	-	32	19	12	7	6
$20 \times 20 \times 4 \times 4$ upwind	-	41	26	17	12	17
$20 \times 20 \times 8 \times 8$ upwind	-	120	72	40	28	20

It is more efficient to apply the previous strategy to larger groups of cells.

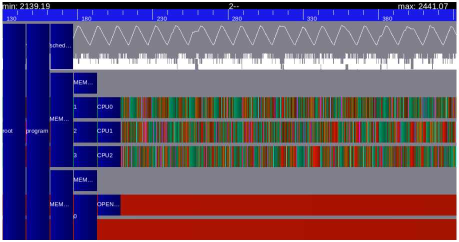
Task graph

Zoom of the task graph generated by StarPU



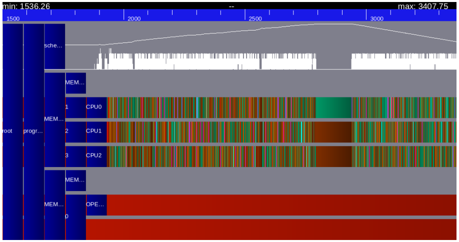
Gantt diagram

Gantt diagram generated by StarPU: sync point at the end of each time step



Gantt diagram

Gantt diagram generated by StarPU: without sync point at the end of each time step



Kinetic schemes

Kinetic framework

- Distribution function: $f(x, v, t)$, $x \in \mathbb{R}^d$, $v \in \mathbb{R}^d$, $t \in [0, T]$.
- Microscopic "collision vector" $K(v) \in \mathbb{R}^m$. Macroscopic conserved data

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

- Microscopic entropy s and associated Maxwellian $M_w(v)$:

$$\int_v M_w K = w, \quad \int_v s(M_w) = \max_{\int_v fK = w} \left\{ \int_v s(f) \right\}.$$

- Kinetic-BGK equation ($a = a(x, t)$ is the acceleration):

$$\partial_t f + v \cdot \nabla_x f + a \cdot \nabla_v f = \frac{1}{\tau} (M_w - f).$$

Kinetic schemes

When the relaxation time τ is small, the kinetic equation provides an approximation of the hyperbolic conservative system

$$\partial_t w + \nabla \cdot F(w) + \Pi(w) = 0,$$

with

$$F^i(w) = \int_v v^i M_w(v) K(v) dv.$$

$$\Pi(w) = a \cdot \int_v \nabla_v M_w(v) K(v) = -a \cdot \int_v M_w(v) \nabla_v K(v).$$

Main idea: numerical solvers for the linear scalar transport equation lead to natural solvers for the non-linear hyberbolic system [Deshpande, 1986]. Micro or macro approach.

Toy problem: Broadwell model

- Lattice "D1Q3": $v \in V = \{-1, 0, 1\}$. $K(v) = (1, v)^T$.
- $w = (\rho, \rho u)^T$.
- $f(x, v, t)$ is solution of

$$\partial_t f + v \partial_x f = 1/\tau (M_w - f).$$

- $M_w(\pm 1) = \rho u(u \pm 1)/2 + c^2 \rho/2$,
  $M_w(0) = \rho(1 - u^2 - c^2)$.
- $\rho = \int_{v \in V} f = f(\cdot, -1, \cdot) + f(\cdot, 0, \cdot) + f(\cdot, 1, \cdot)$,
  $\rho u = \int_{v \in V} fv = f(\cdot, 1, \cdot) - f(\cdot, -1, \cdot)$,
  $\rho u^2 + c^2 \rho = \int_{v \in V} fv^2 = f(\cdot, 1, \cdot) + f(\cdot, -1, \cdot)$.
- Then $w = (\rho, \rho u)^T$ is solution of the isothermal Euler equations

$$\partial_t \rho + \partial_x(\rho u) = 0,$$

$$\partial_t(\rho u) + \partial_x(\rho u^2 + c^2 \rho) = 0.$$

First order splitting algorithm

For each time step of duration Δt,

- free transport: solve (for $v = -1, 0, 1$)

$$\partial_t f + v \partial_x f = 0;$$

- relaxation: compute $w = (\rho, u)^T$ and return to

$$f(x, v, t) = M_{w(x,t)}(v).$$

The resulting scheme is $O(\Delta t)$ with high numerical viscosity

Implicit upwind approximation

1D DG approximation with order $d = 0$ for the transport equation (finite volume method)

- space step Δx and time step Δt.
- constant approximation of f in each cell $C_i =]i\Delta x, (i+1)\Delta x[$, $i \in \mathbb{Z}$.
- If $x \in C_i$ and $t \in [p\Delta t, (p+1)\Delta t[$ then $f(x,v,t) \simeq f_i^p(v)$.
- Upwind scheme

$$v > 0, \quad \frac{f_i^p - f_i^{p-1}}{\Delta t} + v\frac{f_i^p - f_{i-1}^p}{\Delta x} = 0$$

$$v < 0, \quad \frac{f_i^p - f_i^{p-1}}{\Delta t} + v\frac{f_{i+1}^p - f_i^p}{\Delta x} = 0$$

Comments

- Unconditionally stable. CFL number$=\Delta t / v\Delta x$ can be arbitrary.
- Implicit scheme but upper or lower triangular linear system.
- Parallelism: uncoupled transport equations, local relaxation.
- Higher order in x: Discontinuous Galerkin (DG) method (polynomial approximation of f in cells C_i)
- Issue: higher order in time ?

Second order extension

Spatial approximation leads to a differential equation

$$v'(t) = g(v(t)), \quad v(0) = v_0.$$

First order numerical method

$$\Phi(\Delta t)v_0 = v_0 + g(v_0)\Delta t + E(v_0)\Delta t^2 + O(\Delta t^3).$$

More precise method $\Psi(\Delta t) = \Phi(\alpha\Delta t)\Phi(\beta\Delta t)$.
Second order is attained iff

$$\alpha + \beta = 1, \quad \alpha\beta = \frac{1}{2}.$$

We find $\alpha = \frac{1+i}{2}, \quad \beta = \frac{1-i}{2}$.

Comments and possible generalization

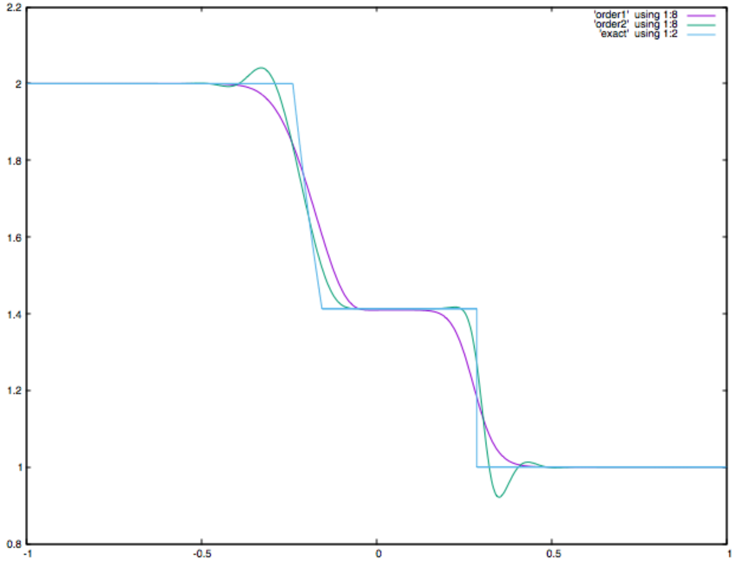
- Bibliography: [Fung, 1998, McLachlan and Quispel, 2002]
- Very easy to program: consider complex time-step Δt. At the end of the time-step change Δt to $\overline{\Delta t}$.
- Transport: each sub-step is unstable. The global step is stable.
- Complexity: storage$\times 2$, CPU time approximately$\times 2.5$.
- Possible generalization to higher orders. Rely on the BCH formula in the Lie algebra of vector fields

$$e^Z = e^X e^Y, \quad Z = X + Y + \frac{1}{2}[X, Y] + \cdots$$

Numerical results

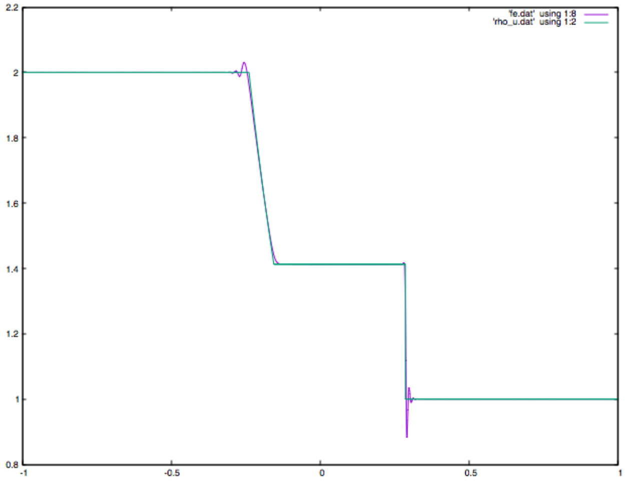
- Riemann problem (discontinuous initial data made of two constant states): $\rho_L = 2$, $\rho_R = 1$, $u_L = u_R = 0$.
- 200 cells (800 nodes). $x \in [-1, 1]$
- Compare first and second order time integration at time $t = 0.4$.

density, CFL=2



finer mesh

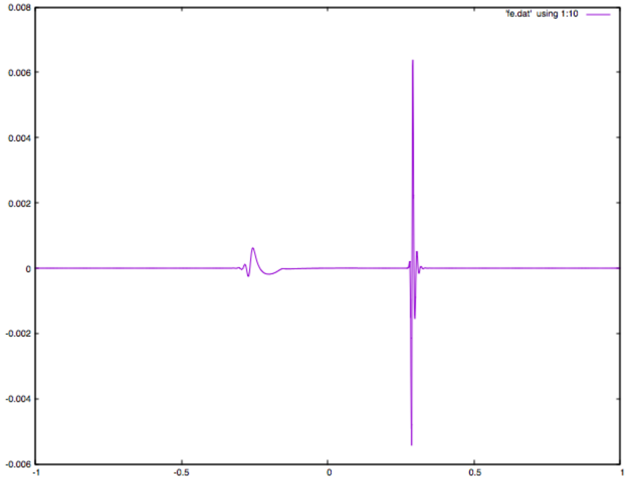
600 cells, CFL=0.5



Oscillations but nice stability and convergence

Imaginary part

600 cells, CFL=0.5

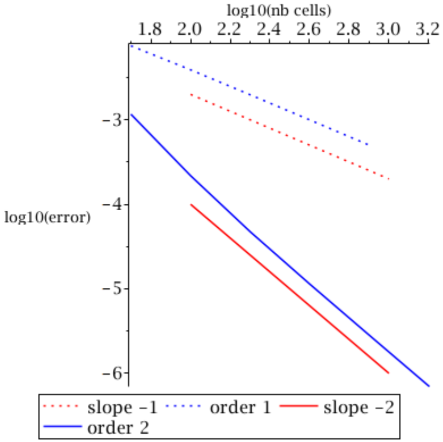


shock detector ?

Smooth solution

- Smooth initial data. No shock wave before final time.
- Implicit Discontinuous Galerkin scheme with third order polynomials.
- 50 to 800 cells
- Convergence rate for first and second order time integration.

Convergence rate



- We obtain the expected convergence rate.
- Second order Strang splitting would degenerate to first order scheme when $\tau \to 0$ [Jin, 1995]

Conclusion

- DG method is an old but still alive method.
- Research on DG is very active.
- Well adapted to mesh adaptation, to parallelism.
- Many applications: transport, fluid mechanics, electromagnetism, *etc.*

Bibliography I

[Augonnet et al., 2012] Augonnet, C., Aumage, O., Furmento, N., Namyst, R., and Thibault, S. (2012).
StarPU-MPI: Task Programming over Clusters of Machines Enhanced with Accelerators.
In Jesper Larsson Träff, S. B. and Dongarra, J., editors, *EuroMPI 2012*, volume 7490 of *LNCS*. Springer.
Poster Session.

[Barth, 2006] Barth, T. (2006).
On the role of involutions in the discontinuous galerkin discretization of maxwell and magnetohydrodynamic systems.
In *Compatible spatial discretizations*, pages 69–88. Springer.

[Bourdel et al., 1992] Bourdel, F., Mazet, P.-A., and Helluy, P. (1992).
Resolution of the non-stationary or harmonic maxwell equations by a discontinuous finite element method. application to an emi (electromagnetic impulse) case.
In *10th international conference on computing methods in applied sciences and engineering on Computing methods in applied sciences and engineering*, pages 405–422. Nova Science Publishers, Inc. Commack, NY, USA.

[Cockburn and Shu, 1998] Cockburn, B. and Shu, C.-W. (1998).
The runge–kutta discontinuous galerkin method for conservation laws v: multidimensional systems.
Journal of Computational Physics, 141(2):199–224.

[Deshpande, 1986] Deshpande, S. (1986).
Kinetic theory based new upwind methods for inviscid compressible flows.
In *24th AIAA Aerospace Sciences Meeting*, volume 1.

[Duff and Reid, 1978] Duff, I. S. and Reid, J. K. (1978).
An implementation of tarjan's algorithm for the block triangularization of a matrix.
ACM Transactions on Mathematical Software (TOMS), 4(2):137–147.

Bibliography II

[Fung, 1998] Fung, T. (1998).
Complex-time-step newmark methods with controllable numerical dissipation.
International Journal for numerical methods in Engineering, 41(1):65–93.

[Helluy and Jung, 2014] Helluy, P. and Jung, J. (2014).
Interpolated pressure laws in two-fluid simulations and hyperbolicity.
In *Finite volumes for complex applications. VII. Methods and theoretical aspects*, volume 77 of *Springer Proceedings in Mathematics & Statistics*, pages 37–53. Springer.

[Hesthaven and Warburton, 2007] Hesthaven, J. S. and Warburton, T. (2007).
Nodal discontinuous Galerkin methods: algorithms, analysis, and applications.
Springer Science & Business Media.

[Jin, 1995] Jin, S. (1995).
Runge-kutta methods for hyperbolic conservation laws with stiff relaxation terms.
Journal of Computational Physics, 122(1):51–67.

[Johnson et al., 1984] Johnson, C., Nävert, U., and Pitkäranta, J. (1984).
Finite element methods for linear hyperbolic problems.
Computer methods in applied mechanics and engineering, 45(1):285–312.

[Lesaint and Raviart, 1974] Lesaint, P. and Raviart, P.-A. (1974).
On a finite element method for solving the neutron transport equation.
Mathematical aspects of finite elements in partial differential equations, (33):89–123.

[McLachlan and Quispel, 2002] McLachlan, R. I. and Quispel, G. R. W. (2002).
Splitting methods.
Acta Numerica, 11:341–434.

[Natvig and Lie, 2008] Natvig, J. R. and Lie, K.-A. (2008).
Fast computation of multiphase flow in porous media by implicit discontinuous galerkin schemes with optimal ordering of elements.
Journal of Computational Physics, 227(24):10108–10124.

Bibliography III

[Reed and Hill, 1973] Reed, W. H. and Hill, T. (1973).
Triangularmesh methodsfor the neutrontransportequation.
Los Alamos Report LA-UR-73-479.

[Strub, 2015] Strub, T. (2015).
Resolution of tridimensional instationary Maxwell's equations on massively multicore architecture.
Theses, Université de strasbourg.

[Wang and Xu, 1999] Wang, F. and Xu, J. (1999).
A crosswind block iterative method for convection-dominated problems.
SIAM Journal on Scientific Computing, 21(2):620–645.

[Zingan et al., 2013] Zingan, V., Guermond, J.-L., Morel, J., and Popov, B. (2013).
Implementation of the entropy viscosity method with the discontinuous galerkin method.
Computer Methods in Applied Mechanics and Engineering, 253:479–490.