

Towards an ultra efficient kinetic scheme : High-Performance Computing

G. Dimarco¹ J. Narski² R. Loubère²

¹Department of Mathematics and Computer Science, University of Ferrara, Italy.

²Institut de Mathématiques de Toulouse, Université de Toulouse 31062 Toulouse, France.

SHARK-FV Ofir 2014

Motivations

- Modeling of non equilibrium gas flows (plasma, hypersonic flow)
- Kinetic equations extremely difficult to solve numerically (7 dimensions)

FKS

- G. DIMARCO, R. LOUBÈRE, *Towards an ultra efficient kinetic scheme. Part I: basics on the BGK equation*, J. Comput. Phys., Vol. 255, 2013, pp 680-698.
- G. DIMARCO, R. LOUBÈRE, *Towards an ultra efficient kinetic scheme. Part II: the high order case*, J. Comput. Phys., Vol. 255, 2013, pp 699-719.

Goal

- Parallelize FKS

Problem formulation

Boltzmann-BGK equation

$$\partial_t f + \mathbf{V} \cdot \nabla_{\mathbf{X}} f = \frac{1}{\tau} (M_f - f),$$

$f = f(\mathbf{X}, \mathbf{V}, t)$ distribution of particles, τ relaxation time.

Collisions modeled by relaxation towards the local thermodynamical equilibrium defined by the Maxwellian distribution

$$M_f = M_f[\rho, \mathbf{U}, T](\mathbf{V}) = \frac{\rho}{(2\pi\theta)^{3/2}} \exp\left(-\frac{\|\mathbf{U} - \mathbf{V}\|^2}{2\theta}\right),$$

$\theta = TR$, ρ, \mathbf{U}, T, R — density, mean velocity, temperature and gas constant

Macroscopic moments

$$\rho = \int_{\mathbb{R}^3} f \, d\mathbf{V}, \quad \mathbf{U} = \frac{1}{\rho} \int_{\mathbb{R}^3} \mathbf{V} f \, d\mathbf{V}, \quad \theta = \frac{1}{3\rho} \int_{\mathbb{R}^3} \|\mathbf{V} - \mathbf{U}\|^2 f \, d\mathbf{V},$$

Total energy

$$E = \frac{1}{2} \int_{\mathbb{R}^3} \|\mathbf{V}\|^2 f \, d\mathbf{V} = \frac{1}{2} \rho \|\mathbf{U}\|^2 + \frac{3}{2} \rho \theta$$

The limit of $\tau \rightarrow 0$

If number of collisions goes to infinity, then $\tau \rightarrow 0$ and $f \rightarrow M_f$. One retrieves compressible Euler equations

$$\frac{\partial \rho}{\partial t} + \nabla_{\mathbf{x}} \cdot (\rho \mathbf{U}) = 0,$$

$$\frac{\partial(\rho \mathbf{U})}{\partial t} + \nabla_{\mathbf{x}} \cdot (\rho \mathbf{U} \otimes \mathbf{U} + pI) = 0,$$

$$\frac{\partial E}{\partial t} + \nabla_{\mathbf{x}} \cdot ((E + p)\mathbf{U}) = 0,$$

$$p = \rho\theta, \quad E = \frac{3}{2}\rho\theta + \frac{1}{2}\rho\|\mathbf{U}\|^2,$$

where I is the identity and p the pressure given by a perfect equation of state with gas constant $\gamma = 5/3$ in 3D.

This is the fluid/macroscopic model.

Fast Kinetic Scheme

Semi-Lagrangian scheme for Discrete Velocity Model (DVM)
approximation of Boltzmann-BGK equation.

DVM

Let \mathcal{K} be a bounded set of N_v^3 multi-indices of \mathbb{N}^3 . Let \mathcal{V} be a Cartesian grid given by

$$\mathcal{V} = \{\mathbf{v}_k = k\Delta v + \mathbf{W}, k \in \mathcal{K}\},$$

where Δv is the grid step in the velocity space. The generic cell in the velocity space is $\omega_{k+1/2} = [\mathbf{v}_k; \mathbf{v}_{k+1}]$. We denote the discrete collision invariants on \mathcal{V} by

$$m_k = \left(1, \mathbf{v}_k, \frac{1}{2}\|\mathbf{v}_k\|^2\right)^t$$

Fast Kinetic Scheme

Semi-Lagrangian scheme for Discrete Velocity Model (DVM)
approximation of Boltzmann-BGK equation.

DVM

The continuous distribution function f is replaced by a vector

$$f_{\mathcal{K}}(\mathbf{X}, t) = (f_k(\mathbf{X}, t))_k, \quad f_k(\mathbf{X}, t) \approx f(\mathbf{X}, \mathbf{V}_k, t).$$

Fluid quantities:

$$F(\mathbf{X}, t) = \sum_{k \in \mathcal{K}} m_k f_k(\mathbf{X}, t) \Delta v.$$

Fast Kinetic Scheme

Set of N_v^3 evolution equations

$$\partial_t f_k + \mathbf{V}_k \cdot \nabla_{\mathbf{x}} f_k = \frac{1}{\tau} (\mathcal{E}_k[F] - f_k)$$

Space and time discretization

Cartesian uniform grid $\mathcal{X} = \{\mathbf{X}_j = j\Delta\mathbf{x} + \mathbf{Y}, j \in \mathcal{J}\}$, $\Delta\mathbf{x}$ is the grid step, \mathbf{Y} is a vector in \mathbb{R}^3 and \mathcal{J} is a subset of \mathbb{N}^3 .

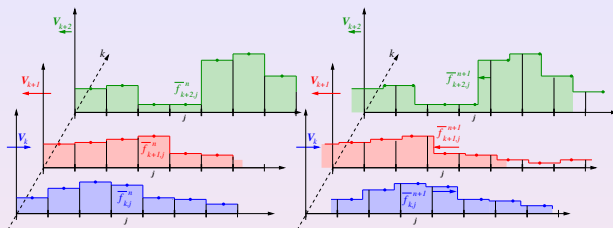
$t^{n+1} = t^n + \Delta t$, time step Δt defined by a CFL condition.

Splitting

$$\text{Transport stage} \quad \longrightarrow \quad \partial_t f_k + \mathbf{V}_k \cdot \nabla_{\mathbf{x}} f_k = 0,$$

$$\text{Relaxation stage} \quad \longrightarrow \quad \partial_t f_k = \frac{1}{\tau} (\mathcal{E}_k[F] - f_k).$$

Fast Kinetic Scheme



Let $f_{j,k}^n$ be the pointwise approximation at discrete time t^n of the distribution f : $f_{j,k}^n = f(\mathbf{X}_j, \mathbf{V}_k, t^n)$ and $\mathcal{E}_{j,k}^n[F]$ be the equilibrium distribution approximation of $M_{j,k}^n = M_f(\mathbf{X}_j, \mathbf{V}_k, t^n)$ defined at any point \mathbf{X}_j of space at discrete time $t = t^n$. Let \bar{f}_k^n be a piecewise constant function associated with velocity \mathbf{V}_k at time t^n defined at each space cell by

$$\bar{f}_{k,j}^n = \frac{1}{|\Omega_j|} \int_{\Omega_j} f(\mathbf{X}, \mathbf{V}_k, t^n) d\mathbf{X}$$

Exact transport during Δt :

$$\bar{f}_k^{*,n+1} = \bar{f}_k^n(\mathbf{X} - \mathbf{V}_k \Delta t), \quad \forall \mathbf{X} \in \Omega$$

Relaxation step

$$\partial_t f_{j,k} = \frac{1}{\tau} (\mathcal{E}_{j,k}[F] - f_{j,k})$$

Initial data is given by the result of the transport step

$$f_{j,k}^{*,n+1} = \bar{f}_k^{*,n+1}(\mathbf{X}_j).$$

Maxwellian computed using macroscopic quantities

$$F_j^{n+1} = F_j^{*,n+1} = \sum_{k \in \mathcal{K}} m_k f_{j,k}^{*,n+1} \Delta v$$

Preservation of macroscopic quantities: moments before ($F_j^{*,n+1}$) and after (F_j^{n+1}) unchanged. Then

$$f_{j,k}^{n+1} = \exp(-\Delta t/\tau) f_{j,k}^{*,n+1} + (1 - \exp(-\Delta t/\tau)) \mathcal{E}_k[F_j^{n+1}],$$

New value of f^{n+1} only in the cell centers, we need f^{n+1} in whole domain for the transport step.

Relaxation step

$$\partial_t f_{j,k} = \frac{1}{\tau} (\mathcal{E}_{j,k}[F] - f_{j,k})$$

Define \mathcal{E}_k as the equilibrium function with the discontinuities located in the same positions as f_k

$$\bar{\mathcal{E}}_k^{n+1}(\mathbf{X})[F] = \mathcal{E}_{j,k}^{n+1}[F], \quad \forall \mathbf{X} \text{ such that } \bar{f}_k^{*,n+1}(\mathbf{X}) = \bar{f}_k^{*,n+1}(\mathbf{X}_j)$$

Finally

$$\bar{f}_k^{n+1}(\mathbf{X}) = \bar{f}_k(\mathbf{X}, t^n + \Delta t) = \exp(-\Delta t/\tau) \bar{f}_k^*(\mathbf{X}) + (1 - \exp(-\Delta t/\tau)) \bar{\mathcal{E}}_k^{n+1}(\mathbf{X})[F]$$

HOFKS - high order extension

Second order in time

Time splitting with Strang splitting strategy. CFL:

$$\Delta t \max_K \frac{\|\mathbf{V}_k\|}{L_c} \leq 1$$

- performs well in collisionless regimes
- scheme stable for $\Delta t > CFL$
- projection over the equilibrium of first order \rightarrow loss of accuracy close to fluid regime

HOFKS - high order extension

$$\bar{f}_k^{n+1}(\mathbf{X}) = \exp(-\Delta t/\tau)\bar{f}_k^*(\mathbf{X}) + (1 - \exp(-\Delta t/\tau))\bar{\mathcal{E}}_k^{n+1}(\mathbf{X})[F]$$

Solve the equilibrium part with of the distribution function with a macroscopic scheme instead of kinetic.

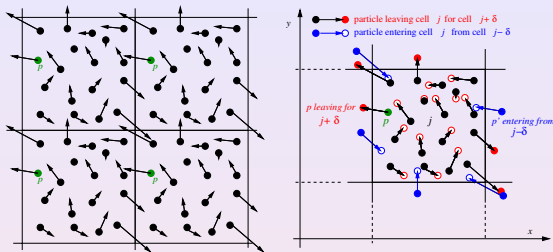
Moments from the transport stage are replaced by a solution of Euler equations. We use MUSCL scheme. Stability condition:

$$\Delta t < \frac{1}{2} \frac{\Delta x}{\alpha_{\max}}$$

In the limit of $\tau \rightarrow 0$ scheme corresponds to a high order shock capturing scheme.

Second order close to the fluid limit.

Efficient implementation



Particle implementation: Initially N_V^3 particles are positioned at the cell center

$$\mathbf{X}_p^0 = (\Delta x/2, \Delta y/2, \Delta z/2)^t$$

each particle has a unique constant velocity \mathbf{V}_p from the velocity space, $p = 1, \dots, N_V^3$. The transport of these particles during Δt follows

$$\tilde{\mathbf{X}}_p^{n+1} = \mathbf{X}_p^n + \Delta t \mathbf{V}_p.$$

Same set of particles in every space cell, only positions and velocities of particles in generic cell kept in memory.

Particle mass

Each particle p in cell j carries its “mass” which is updated defined at t^n thanks to the previous mass $m_{j,p}^{n-1}$ and updated moments $\rho_j^n, \mathbf{U}_j^n, \theta_j^n$ as

$$\begin{aligned} m_{j,p}^n &= \exp(-\Delta t/\tau) m_{j,p}^{n-1} + (1 - \exp(-\Delta t/\tau)) M_f[\rho_j^n, \mathbf{U}_j^n, \theta_j^n](\mathbf{V}_p) \\ &= \exp(-\Delta t/\tau) m_{j,p}^{n-1} + (1 - \exp(-\Delta t/\tau)) \frac{(\Delta v)^3 \rho_j^n}{(2\pi\theta_j^n)^{3/2}} \exp\left(\frac{-\|\mathbf{V}_p - \mathbf{U}_j^n\|^2}{2\theta_j^n}\right) \end{aligned}$$

Because the fluid quantities are obtained through discrete summations on particles in cell j :

$$F_j^n = \sum_{p=1}^{N_v^3} m_{j,p}^n \Delta v$$

the updated fluid quantities are therefore obtained after the transport step following

$$F_j^{n+1} = F_j^n - \underbrace{\sum_{p, \tilde{\mathbf{X}}_p^{n+1} \notin \Omega_j} m_{j,p}^n \Delta v}_{\text{Leaving particles}} + \underbrace{\sum_{p', \tilde{\mathbf{X}}_{p'}^{n+1} \in \Omega_j} m_{j-\delta,p'}^n \Delta v}_{\text{Entering particles}}.$$

Generic algorithm

- 1 *Relaxation step.* Compute masses of N_v^3 particles, store them in an array of the size $N_v^3 \times N^3$
- 2 *Transport of particles.* Displace N_v^3 particles, produce a list of N_{out} particles escaping the generic cell and store the δ determining the destination and provenance of associated sister particles.
- 3 *Update conservative variables* F_j^{n+1}
- 4 *Update primitive variables*

OpenMP algorithm

- 1 *Relaxation step.* Compute in parallel masses of N_v^3 particles with, parallelization is performed on the external loop over N_v^3 particles.
- 2 *Transport of particles.* Move in parallel N_v^3 particles
- 3 *Update conservative variables.* Test in a parallel loop over N_v^3 particles if a particle has escaped from the generic cell. If so, add a contribution to F_j^{n+1} for every space cell. Update the particle position and exchange particle mass with the associated sister particle.
- 4 *Update primitive variables*

GPU algorithm

- 1 *Copy from CPU to GPU.* Copy to the GPU memory all primitive and conservative variables.
- 2 *Loop over particles*
 - 1 *Transport step* Move every particle and test if it has escaped the generic cell. If so, store the provenance of the sister cell.
 - 2 *Relaxation step* Compute relaxed masses of particles for every space cell using CUDA. Store the result on GPU.
 - 3 *Update conservative variables.* If the particle has escaped the generic cell, add contribution to conservative variables and assign mass to be one of the incoming sister particle.
 - 4 *Copy from GPU to CPU.* Copy the resulting mass array from the GPU memory to the CPU memory.
- 3 *Update primitive variables* in parallel on GPU.
- 4 *Copy from GPU to CPU.* Write to the CPU memory the updated conservative variables.

Easily extendable to multi GPU architectures

HOFKS and HOFKS-OMP

Serial version implemented in C++ compiled with gcc 4.7.2 and -Ofast optimization flag

Computational server with 4 Intel(R) Xeon(TM) E5-4650 processors running at 2.7 GHz (giving a total of 32 physical cores and 64 logical) with 512GB of RAM running under Debian Wheezy

HOFKS-GPU

GPU version implemented in CUDA 5.5 and gcc 4.7.2 and -Ofast optimization flag

Computational server with dual Intel(R) Xeon(TM) E5-2650 processor running at 2.0 GHz (16 physical and 32 virtual cores) with 128GB and 2 Nvidia GTX 780 units (3GB of memory, 2304 CUDA cores at 900MHz each) running under Debian Wheezy

Decent card for gaming, not designed for professional applications (lack of memory error correction, double precision, worse copy engine than Tesla/Quadro)

CUDA architecture

- Massively parallel : 12 multiprocessors consisting of 192 CUDA cores
- Functions executed on GPU (kernels) are executed in warps involving 32 threads
- Parallelization strategy : replace every loop over space cells by a call to suitable CUDA kernel
- Slow CPU \leftrightarrow GPU memory transfer (8Gb/s at most)
- Example: $200^3 \times 15^3$ particles equals to 100Gb of data (mass vector) — 25s lost on transfer from and to GPU
- Sometimes better to recompute some values than to copy them from CPU memory
- Not really possible in this case
- Possibility to use two concurrent copy engine (Tesla/Quadro) : mass array of 1 particle is copied to GPU, second particle is processed by GPU and the updated masses of third particle are transferred back to CPU at the same time \Rightarrow time lost on transferred reduced to 0.

Parallelization test only

- $\Omega = [0, 2]^3$, ball centered at $(1, 1, 1)$, radius $r = 0.2$, number of space cells $N^3 = 25, 50, 100, 200$, velocity space $[-15, 15]^3$ discretized with $N_V = 15^3$ points
- The relaxation parameter $\tau = 10^{-4}$
- Δt fixed at maximal time step

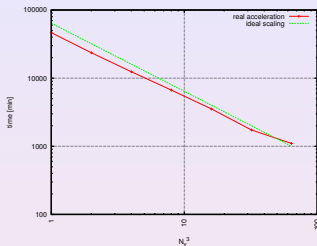
Convergence tests in

- G. DIMARCO, R. LOUBÈRE, *Towards an ultra efficient kinetic scheme. Part I: basics on the BGK equation*, J. Comput. Phys., Vol. 255, 2013, pp 680-698.
- G. DIMARCO, R. LOUBÈRE, *Towards an ultra efficient kinetic scheme. Part II: the high order case*, J. Comput. Phys., Vol. 255, 2013, pp 699-719.

Cell # $N_c \times N_v^3$	Cycle N_{cycle}	Time T (s)	T/cycle T_{cycle} (s)	T/cell T_{cell} (s)	Mem (GB)
$25^3 \times 15^3$ $= 52.7 \times 10^6$	13	204s (3.5mn) 6.77s 6.1s	15.69 0.52 0.47	1×10^{-3} 33×10^{-6} 30×10^{-6}	0.23
$50^3 \times 15^3$ $= 421.9 \times 10^6$	25	3244s (54mn) 86.6s (1.43mn) 46s	129.76 3.46 1.84	1×10^{-3} 27.7×10^{-6} 14.7×10^{-6}	1.6
$100^3 \times 15^3$ $= 3.4 \times 10^9$	50	46408s (13h) 1102s (18.4mn) 486s (8.1mn)	928 22.04 9.7	0.9×10^{-3} 22.04×10^{-6} 9.7×10^{-6}	12
$200^3 \times 15^3$ $= 27 \times 10^9$	98	784×10^3 s (9d) 17036s (4.73h) 9353s (2.6h)	8000 174 95	1×10^{-3} 21.7×10^{-6} 12×10^{-6}	101

OpenMP scalability

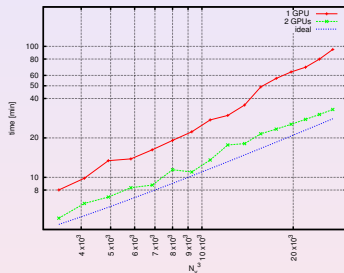
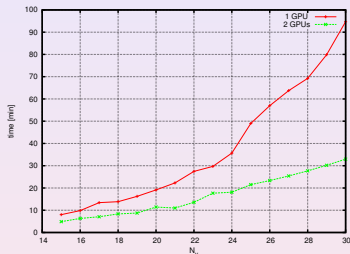
# of cores	Time T (s)	Time/cycle T_{cycle} (s)	Time/cell T_{cell} (s)	Speed up
1	46408	928	928×10^{-6}	1
2	23573	471	471×10^{-6}	1.96
4	12395	248	248×10^{-6}	3.74
8	6674	133.5	133.5×10^{-6}	6.95
16	3536	70.7	70.7×10^{-6}	13.12
32	1735	34.7	34.7×10^{-6}	26.74
64	1102	22.04	22.04×10^{-6}	42.11



- Smaller speed-up when no. of threads exceeds no. of physical cores
- The HOFKS is “embarrassingly parallel”

GPU scalability

SOD test again, $N = 100^3$, N_v varies from 15^3 to 30^3



- Computational time grows linearly with number of particles
- 2 GPUs almost twice as fast as single GPU

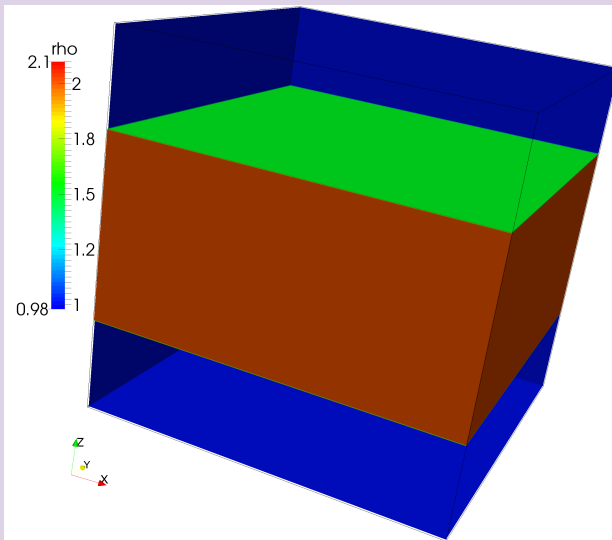
Kelvin-Helmholtz instabilities

- Cubic domain $\Omega = [0, 1]^3$, $N^3 = 200^3$, $N_v^3 = 10^3$
- Three layers : $\Omega_1 = \{\mathbf{X} \in \Omega, 0 \leq z < 0.25\}$,
 $\Omega_2 = \{\mathbf{X} \in \Omega, 0.25 \leq z \leq 0.75\}$, and $\Omega_3 = \Omega \setminus \{\Omega_1 \cup \Omega_2\}$
- Initial conditions

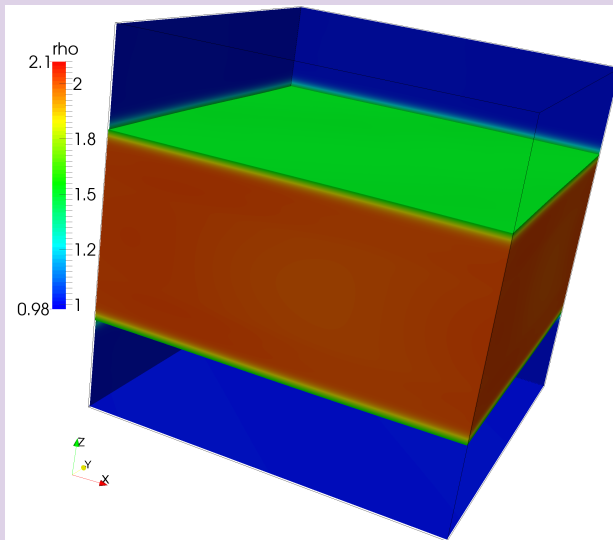
	Ω_1 and Ω_3	Ω_2
ρ^0	1	2
p^0	2.5	2.5
\mathbf{u}^0	$\begin{pmatrix} \frac{1}{2} \\ 0 \\ 0 \end{pmatrix} + a(x, y) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} -\frac{1}{2} \\ 0 \\ 0 \end{pmatrix} + a(x, y) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

with $a(x, y) = 0.01 \sin(2\pi x) \sin(2\pi y)$ and $\gamma = 1.4$.

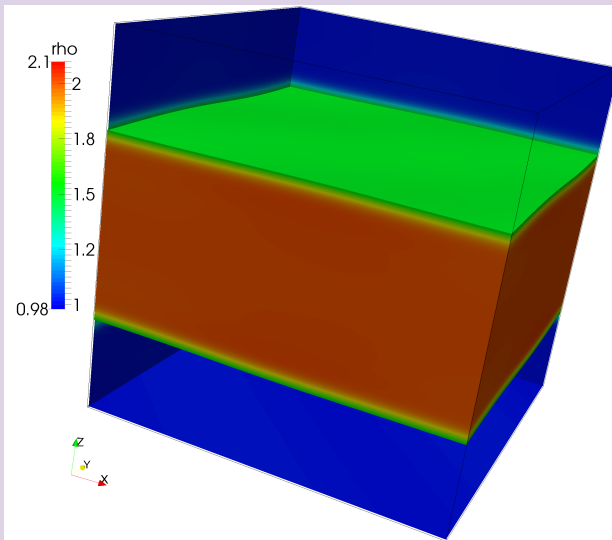
Time step : 0 CPU time : 0h



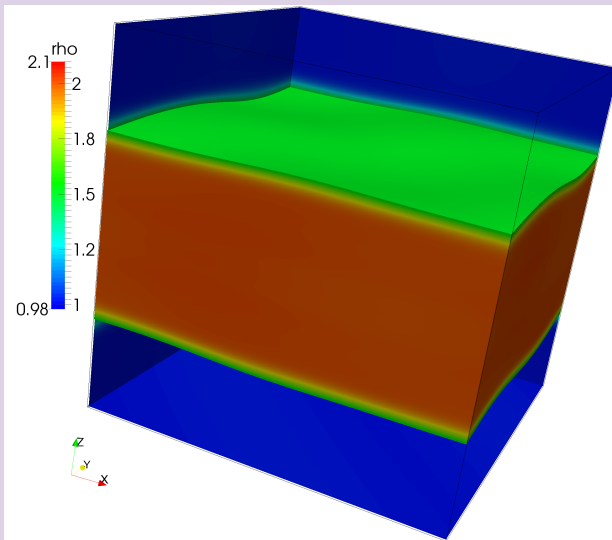
Time step : 1000 CPU time : 4h



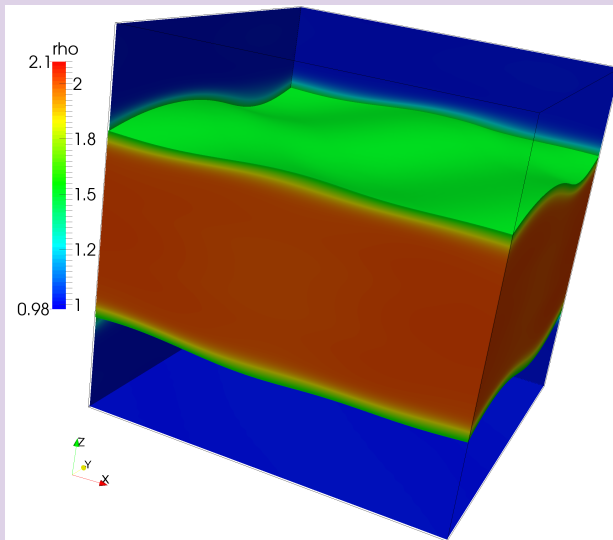
Time step : 2000 CPU time : 8h



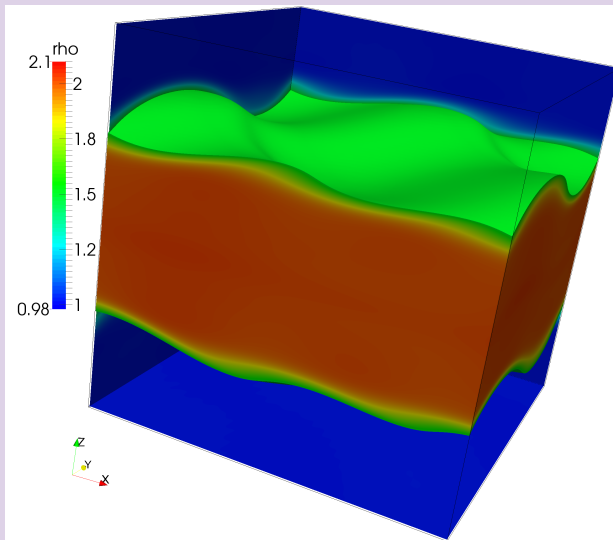
Time step : 3000 CPU time : 12h



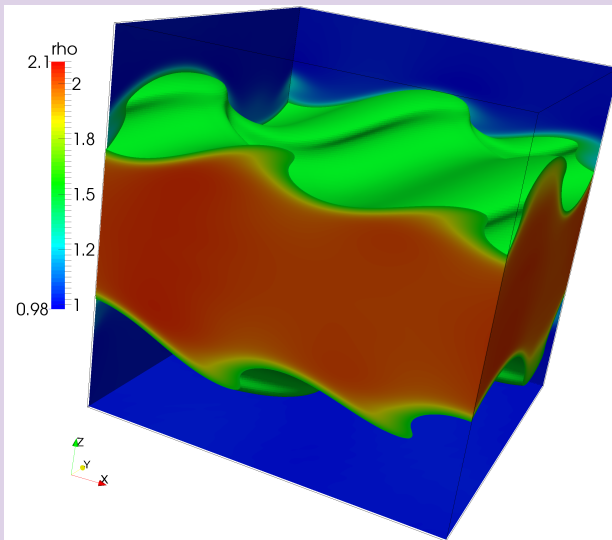
Time step : 4000 CPU time : 16h



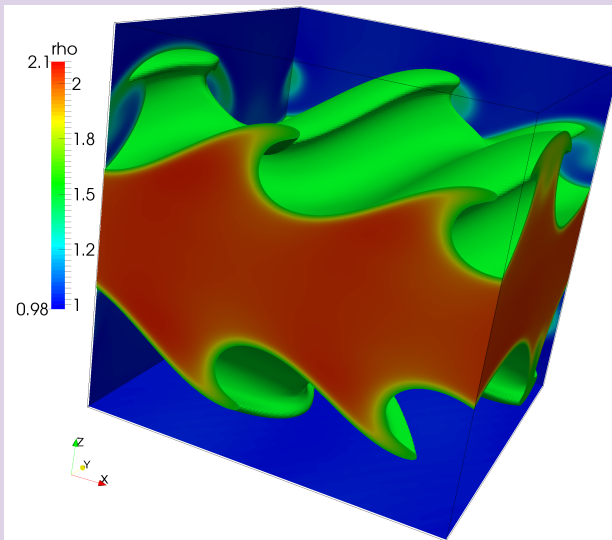
Time step : 5000 CPU time : 20h



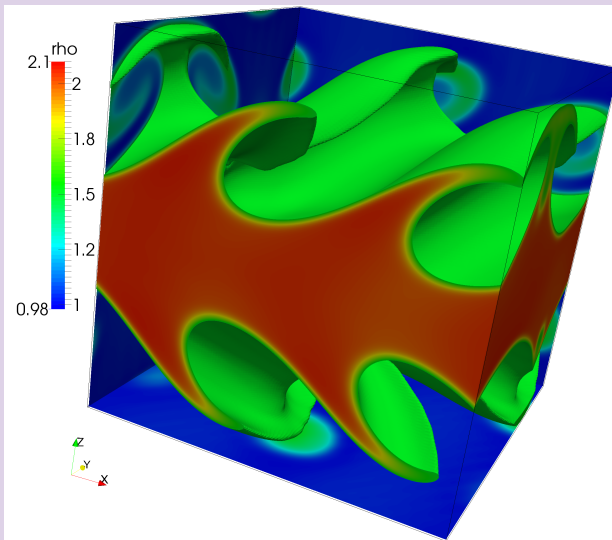
Time step : 6000 CPU time : 24h



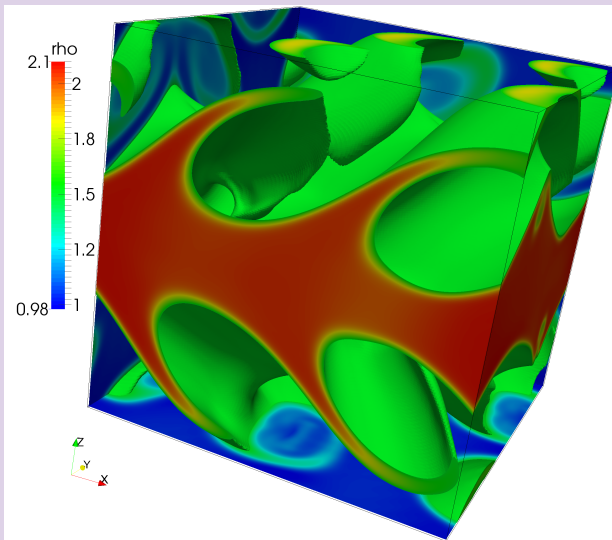
Time step : 7000 CPU time : 28h



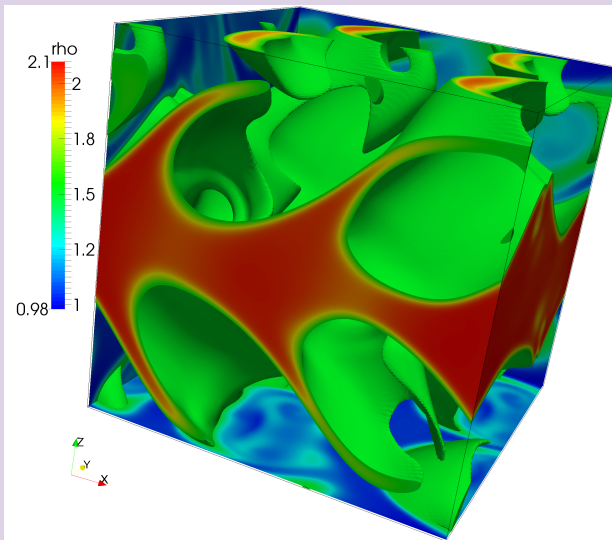
Time step : 8000 CPU time : 32h



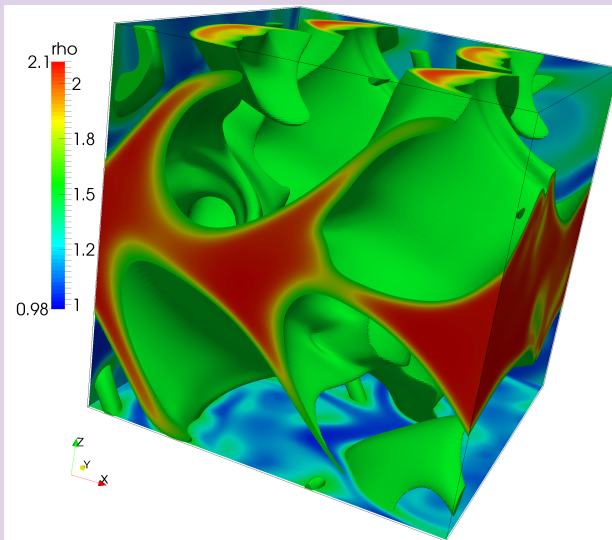
Time step : 9000 CPU time : 36h



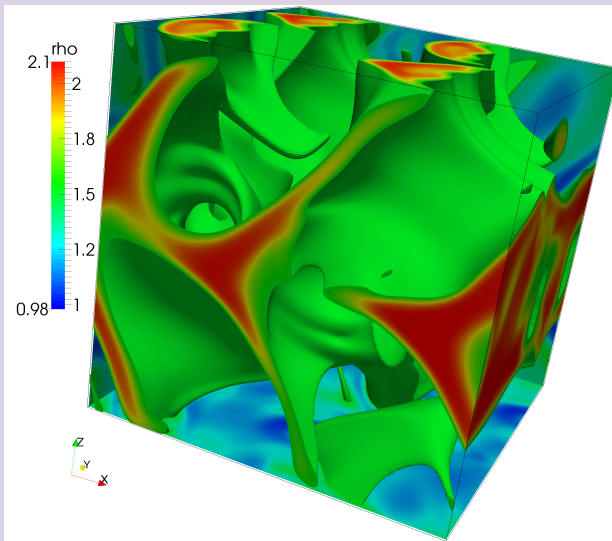
Time step : 10000 CPU time : 40h



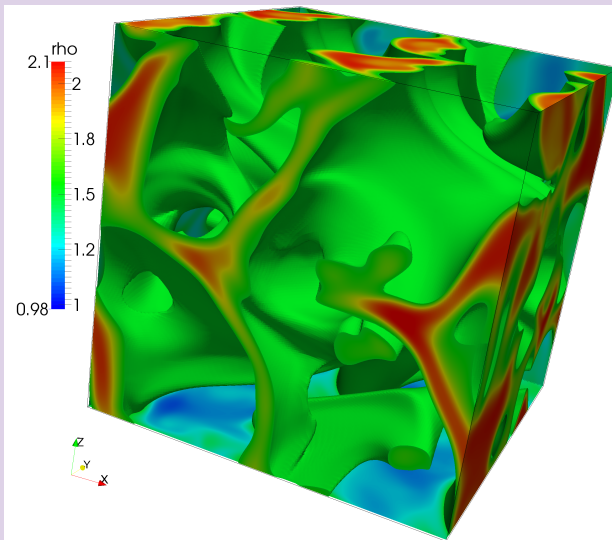
Time step : 11000 CPU time : 44h



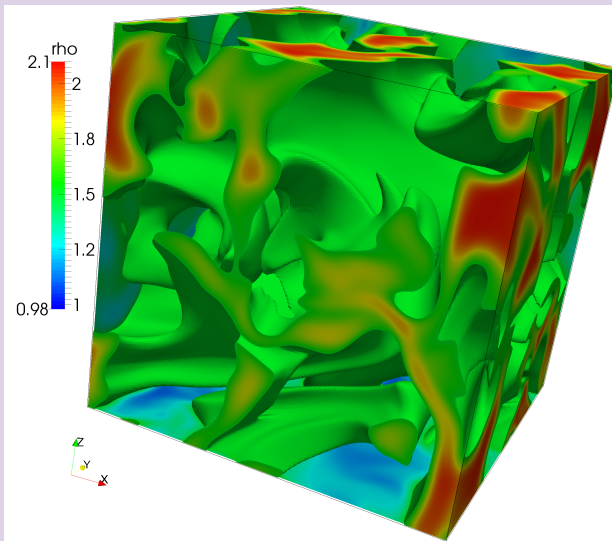
Time step : 12000 CPU time : 48h



Time step : 13000 CPU time : 52h



Time step : 14000 CPU time : 56h



Time step : 15000 CPU time : 60h

