ESAIM: PROCEEDINGS AND SURVEYS, September 2014, Vol. 45, p. 485-492
J.-S. Dhersin, Editor

NUMERICAL SIMULATION FOR THE MHD SYSTEM IN 2D USING OPENCL *

Michel Massaro[1], Philippe Helluy[2] and Vincent Loechner[3]

Abstract. In this work we compute the MHD equations with divergence cleaning on GPU. The method is based on the finite volume approach and Strang dimensional splitting. The simplicity of the approach makes it a good candidate for a GPU implementation with OpenCL. With adequate memory optimization access, we achieve very high speedups, compared to a classical sequential implementation.

Résumé. Dans ce travail, nous résolvons les équations de la MHD avec correction de divergence sur carte graphique. La méthode est basée sur les volumes finis et le splitting directionnel de Strang. La simplicité de l'algorithme en fait un bon candidat pour la programmation sur carte graphique sous OpenCL. Avec de bonnes optimisations des accès mémoire, nous obtenons de très bonnes accélérations, comparé à une programmation séquentielle classique.

Introduction

The aim of this work is to propose an efficient algorithm for solving the two-dimensional MHD equations. The Magneto-Hydro-Dynamics (MHD) system is a model to describe the behavior of conducting fluids inside a magnetic field. It is useful for instance for simulating the edge plasma in tokamak simulations or astrophysics plasmas. For one-dimensional computations, we can assume that the magnetic field in the x-direction is constant. In this case the divergence free condition on the magnetic field B, which reads $\nabla \cdot B = 0$, is automatically satisfied. In higher dimensions, it is important to impose the divergence free condition in order to ensure a good precision of the scheme for long time simulations. We impose the condition through a divergence cleaning technique described in [3, 5].

Our numerical method is based on a the finite volume approach. We have implemented two differents fluxes for solving the problem: the Rusavov flux, the VFRoe flux [2].

One objective of our work is to compute plasma reconnection in astrophysics. Such computations require very fine meshes and thus lead to long computations. Therefore, we have implemented our algorithm on GPU using the OpenCL environment. We use several techniques in order to achieve very high memory bandwitdth: optimized transposition algorithm, cache prefetch, *etc.* We test our algorithms on one-dimensional and two-dimensional well-known test cases: a Riemann problem with a strong shock and the Orzag-Tang vortex.

* *This work has been published under the framework of the LABEX ANR-11-LABX-0055-IRMIA and benefits from a funding from the state managed by the French National Research Agency as part of the Investments for the future program.*

[1] IRMA, 7 rue René Descartes, 67084 Strasbourg, michel.massaro@unistra.fr

[2] IRMA, 7 rue René Descartes, 67084 Strasbourg & Inria Tonus, helluy@unistra.fr

[3] ICPS - LSIIT, Boulevard Sébastien Brant, 67400 Illkirch-Graffenstaden, loechner@icps.u-strasbg.fr

Article published online by EDP Sciences and available at http://www.esaim-proc.org or http://dx.doi.org/10.1051/proc/201445050

1. Model

The Magneto-Hydro-Dynamics (MHD) system a useful model for describing the behavior of astrophysical plasmas, or conducting fluids in a magnetic fields. The unknowns are the density ρ, the velocity $\mathbf{u} \in \mathbb{R}^3$, the internal energy e, the pressure p and the magnetic field $\mathbf{B} \in \mathbb{R}^3$. All of these unknowns depend on the time t and on the space variable $x \in \mathbb{R}^3$. For numerically enforcing the zero divergence condition on the magnetic field, it was proposed [6] to modify the original MHD equations by considering an additional variable ψ. With this "divergence cleaning" the equations read:

$$\partial_t \begin{pmatrix} \rho \\ \rho\mathbf{u} \\ E \\ \mathbf{B} \\ \psi \end{pmatrix} + \nabla \cdot \begin{pmatrix} \rho\mathbf{u} \\ \rho\mathbf{u}\otimes\mathbf{u} + (p + \frac{\mathbf{B}\cdot\mathbf{B}}{2})\mathbf{I} - \mathbf{B}\otimes\mathbf{B} \\ (E + p + \frac{\mathbf{B}\cdot\mathbf{B}}{2})\mathbf{u} - (\mathbf{B}\cdot\mathbf{u})\mathbf{B} \\ \mathbf{u}\otimes\mathbf{B} - \mathbf{B}\otimes\mathbf{u} - \psi\mathbf{I} \\ c_h^2\mathbf{B} \end{pmatrix} = 0, \tag{1}$$

$$E = \rho e + \frac{\rho\mathbf{u}\cdot\mathbf{u}}{2} + \frac{\mathbf{B}\cdot\mathbf{B}}{2}. \tag{2}$$

$$p = P(\rho, e) = (\gamma - 1)\rho e, \quad \gamma > 1. \tag{3}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{4}$$

This system is a little bit different from the classical MHD system. It contains an additionnal variable ψ and a constant parameter c_h in order to impose the condition $\nabla \cdot \mathbf{B} = 0$ numerically. The parameter c_h must be an upper bound of the characteristic velocities of the system.

For all vector $\mathbf{n} = (n_1, n_2, n_3)^T \in \mathbb{R}^3$ and $\mathbf{W} = (\rho, \rho\mathbf{u}^T, E, \mathbf{B}^T, \psi)^T$ the vector of conservatives variables, we define the flux

$$\mathcal{F}(\mathbf{W}, \mathbf{n}) = \begin{pmatrix} \rho\mathbf{u}\cdot\mathbf{n} \\ \rho(\mathbf{u}\cdot\mathbf{n})\mathbf{u} + (p + \frac{\mathbf{B}\cdot\mathbf{B}}{2})\mathbf{n} - (\mathbf{B}\cdot\mathbf{n})\mathbf{B} \\ (E + p + \frac{\mathbf{B}\cdot\mathbf{B}}{2})\mathbf{u}\cdot\mathbf{n} - (\mathbf{B}\cdot\mathbf{u})(\mathbf{B}\cdot\mathbf{n}) \\ (\mathbf{u}\cdot\mathbf{n})\mathbf{B} - (\mathbf{B}\cdot\mathbf{n})\mathbf{u} + \psi\mathbf{n} \\ c_h^2\mathbf{B}\cdot\mathbf{n} \end{pmatrix}. \tag{5}$$

The system is hyperbolic and the eigenvalues can be computed explicitely. We note a the speed of sound, c_A the Alfven's velocity, c_f and c_s the fast and slow magneto-acoustic velocity and λ_i the eigenvectors sorted in ascending order.

$$\begin{aligned}
&b = \mathbf{B}\cdot\mathbf{n} \\
&c_A = \frac{b}{\sqrt{\rho}}, \\
&a = \sqrt{\frac{\gamma p}{\rho}}, \\
&c_{f,s} = \sqrt{\frac{1}{2}\left(\frac{b^2 + B^2}{\rho} + a^2\right) \pm \sqrt{\frac{1}{4}\left(\frac{b^2 + B^2}{\rho} + a^2\right)^2 - \frac{a^2 b^2}{\rho}}}, \\
&\lambda_1 = -c_h \\
&\lambda_2 = \mathbf{u}\cdot\mathbf{n} - c_f \quad \lambda_3 = \mathbf{u}\cdot\mathbf{n} - c_A \quad \lambda_4 = \mathbf{u}\cdot\mathbf{n} - c_s \\
&\lambda_5 = \mathbf{u}\cdot\mathbf{n} \\
&\lambda_6 = \mathbf{u}\cdot\mathbf{n} + c_s \quad \lambda_7 = \mathbf{u}\cdot\mathbf{n} + c_A \quad \lambda_8 = \mathbf{u}\cdot\mathbf{n} + c_f \\
&\lambda_9 = c_h
\end{aligned} \tag{6}$$

2. Finite volume approximation

2.1. Numerical scheme

In this work, we solve the two-dimensional MHD equations which read:

$$\partial_t \mathbf{W} + \partial_x \mathcal{F}(\mathbf{W}, \mathbf{n}_x) + \partial_y \mathcal{F}(\mathbf{W}, \mathbf{n}_y) = 0,$$

with $\mathbf{n}_x = (1,0,0)^T$ and $\mathbf{n}_y = (0,1,0)^T$.

We propose to use a dimensional Strang splitting. For advancing a time step Δt we first solve the 1D problem along x-direction

$$\begin{array}{rcl} \partial_t \mathbf{V} + \partial_x \mathcal{F}(\mathbf{V}, \mathbf{n}_x) & = & 0, \\ \mathbf{V}(t=0) & = & \mathbf{W}_0, \end{array}$$

then along y-direction

$$\begin{array}{rcl} \partial_t \mathbf{W} + \partial_y \mathcal{F}(\mathbf{W}, \mathbf{n}_y) & = & 0, \\ \mathbf{W}(t=0) & = & \mathbf{V}(t=\Delta t). \end{array}$$

For solving the one-dimensional problems, we use a finite volume scheme:

$$\mathbf{W}_i^{n+1} - \mathbf{W}_i^n + \frac{\Delta t}{h_i}(\mathcal{F}(\mathbf{W}_i^n, \mathbf{W}_{i+1}^n, \mathbf{n}) - \mathcal{F}(\mathbf{W}_{i-1}^n, \mathbf{W}_i^n, \mathbf{n})) = 0 \tag{7}$$

where $\mathbf{W}_i^n$ is a constant approximation of $\mathbf{W}$ on the cells $[x_{i-1/2}, x_{i+1/2}]$ at time t_n, and $h_i = x_{i+1/2} - x_{i-1/2}$ and where $\mathcal{F}(\mathbf{W}_i^n, \mathbf{W}_{i+1}^n, \mathbf{n})$ represents the numerical flux.

2.2. Numerical flux

The vector of primitive variables is defined by $\mathbf{Y} = (\rho, u_x, u_y, u_z, B_y, B_z, p)^T$. For smooth solutions, the one-dimensional MHD system can also be written under the non-conservative form

$$\partial_t \mathbf{Y} + \mathbf{A}(\mathbf{Y}, \mathbf{n}_x)\partial_x \mathbf{Y} = 0.$$

Let $\mathbf{W_L}$ and $\mathbf{W_R}$ be the left and right states at an interface between two cells. Let $\mathbf{Y}_L$ and $\mathbf{Y}_R$ be the left and right associated primitive variables. The VFRoe numerical flux [5] is given by

$$\mathcal{F}(\mathbf{W}_L, \mathbf{W}_R, \mathbf{n}) = \mathcal{F}(\mathcal{R}(\mathbf{Y}_R, \mathbf{Y}_L, 0, \mathbf{n})),$$

where $\mathcal{R}(\mathbf{Y}_L, \mathbf{Y}_R, \frac{x}{t}) = \mathbf{Y}(x,t)$ is the solution of the following linearized Riemann problem:

$$\begin{array}{r} \partial_t \mathbf{Y} + \mathbf{A}(\overline{\mathbf{Y}}, \mathbf{n})\partial_x \mathbf{Y} = 0 \\ \mathbf{Y}(x,0) = \mathbf{Y}_L \text{ if } x < 0 \\ \mathbf{Y}(x,0) = \mathbf{Y}_R \text{ if } x > 0 \\ \overline{\mathbf{Y}} = \dfrac{\mathbf{Y}_L + \mathbf{Y}_R}{2} \end{array}$$

The solution of the linearized Riemann problem is given by

$$\mathcal{R}(\mathbf{Y}_L, \mathbf{Y}_R, \frac{x}{t}) = \overline{\mathbf{Y}} - \frac{1}{2}\mathrm{sgn}(A(\overline{\mathbf{Y}}) \cdot \mathbf{n} - \frac{x}{t}\mathbf{I})(\mathbf{Y}_R - \mathbf{Y}_L).$$

2.3. **Entropy fix**

The VFRoe flux has the advantage to be more accurate than other more diffusive fluxes such as the HLL or HLLC fluxes. One drawback is that it causes the apparition of non-entropic shocks when the left and right eigenvalues (λ_L) and (λ_R) change of sign. To overcome this problem, we consider an entropy correction described in [2]. The previously defined numerical flux is replaced by

$$\mathcal{F}(\mathbf{W}_L, \mathbf{W}_R, \mathbf{n}) = \mathcal{F}(\mathcal{R}(\mathbf{Y}_R, \mathbf{Y}_L, 0, \mathbf{n})) - \frac{\min(|\lambda_L|, |\lambda_R|)(\mathbf{W}_R - \mathbf{W}_L)}{2}. \tag{8}$$

3. Programing on GPU

3.1. **Environment**

The directional splitting algorithm has the advantage of being very simple and well adapted to parallelization. We have implemented the algorithm with OpenCL for running the code on a graphics card.

As opposed to CPU with (as of 2014) 2,4,8 or 16 cores, a Graphical Processor Unit (GPU) has thousands of cores.

In the OpenCL abstraction a typical GPU has several Comput Units (CU), also known as work-groups. Each work-group has several Processing Elements (PE) called work-items and a fast-access cache memory unit, called local memory in the OpenCL terminology. The amount of cache memory of one work-group is of the order of some kilobytes and it can be accessed only by the work-items of this work-group. In a GPU there is also a global memory. It can be accessed by all the work-items and it is of the order of 1 or 2 Gigabytes. The global memory is slow-access memory compared to cache memory. See figure 1.

The OpenCL abstraction makes it possible, thanks to a queueing procedure, to manage as many virtual work-items as needed by the application. In addition, OpenCL is not bind to a particular GPU vendor. The code can be executed on GPUs or multicore CPUs of different brands.

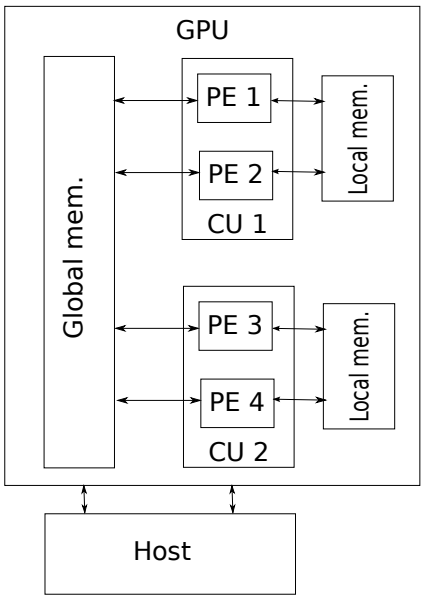

Figure 1. Architecture of a GPU

3.2. **Programing strategy**

A key points of the algorithm is the memory accesses.

The data are stored in a (x, y) grid. Each grid point is associated to one work-item. Thus during the computation of the fluxes balance in the x direction, two neighboring work-items access two neighboring memeory locations. This pattern is optimal for memory bandtwidth ("coalescent" memory accesses). However, when computing the flux balance in the y-direction two neighbor work-items access different rows of the grid and thus break coalescence. Non-coalescent accesses are dramatic for performance: computations appear to be approximately ten times slower. Hence before solving along the y-direction we perform an efficient transposition of the grid. The transposition algorithm is described in [8] .

In more details, a time step of the simulation is managed as follows. Each grid point is assigned to one work-item and one work-group manages one row (or a part of one row if the domain is too large). All the data are stored in the global memory.

(1) Computation along x. Each-work item fetches in its work-group local memory the vector of the conservative variables associated to its position.
(2) Each work-item computes the left and right fluxes. The neighboring data are stored in the local memory and thus the computations are efficient.
(3) Each work-item adds the contribution of its fluxes to the variables in the grid stored in the global memory.
(4) We perform the optimized transposition algorithm of [8]
(5) Computation along y. We follow the same steps as 1, 2 and 3
(6) We then perform another transposition before the next time-step.

4. Result

4.1. **Result in 1D**

On Figure 2, we present the profile of the density at time $t = 1$ with a strong shock test case (see Table 1 for the initial conditions). Computation is done on 2048 grid points. We can see that the VFRoe scheme is really more accurate than the Rusanov scheme. We also verify the importance of the entropy fix. We clearly observe the appearance of a wrong shock when the flux is not corrected.

Variables	Left states	Right states
ρ	3	1
p	3	1
u_x	1.3	1.3
u_y	0	0
u_z	0	0
B_x	1.5	1.5
B_y	1	cos(1.5)
B_z	1	sin(1.5)

Table 1. Initials states for the strong schock test case

4.2. **Result in 2D**

The Orzag-Tang Vortex is a well-known test case for MHD codes. The results of this two-dimensional test case are given at time $t = 1$ and $t = 3$ on Figures 3 and 4. The computation is done on a 4096×4096 grid points with the Rusanov flux. The domain is $[0, 2\pi] \times [0, 2\pi]$ and the boundaries conditions are periodic in x and y. The initial conditions are given in Table 2.

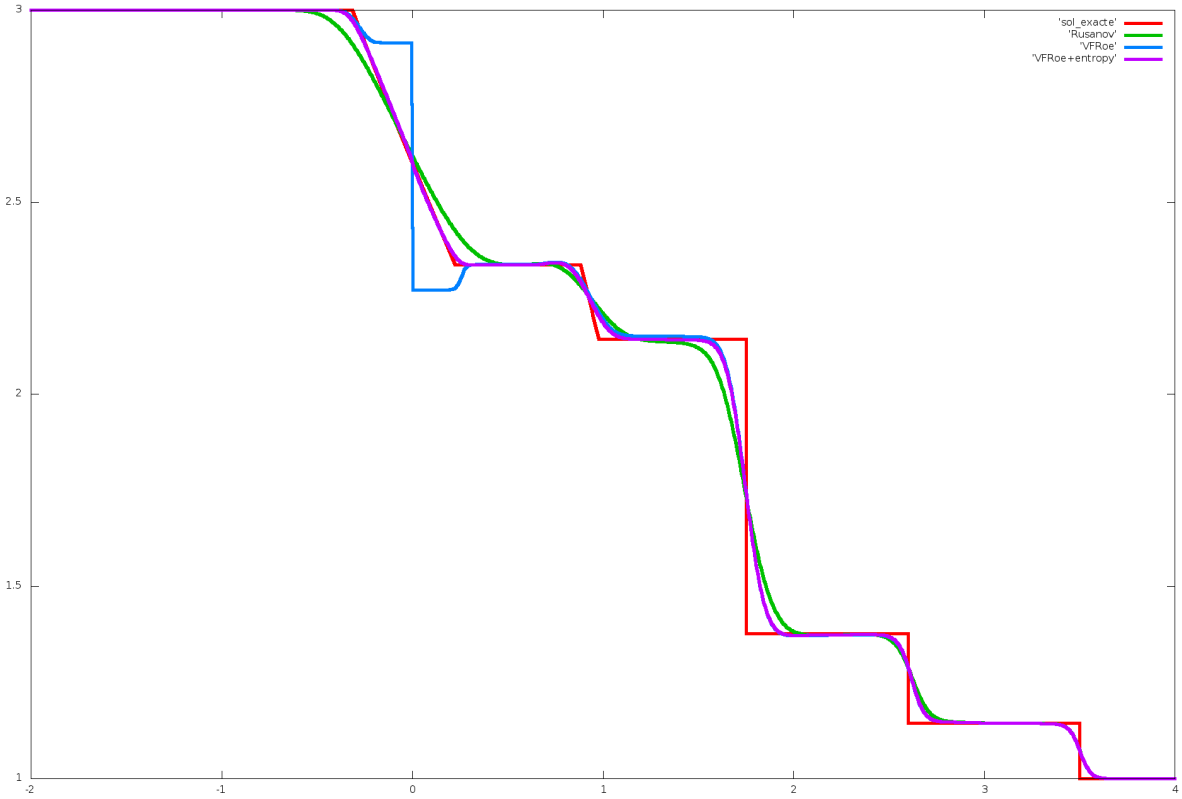

FIGURE 2. Comparison of the numerical fluxes with a one-dimensional Riemann problem. Red: exact, Green: Rusanov, Blue: VFRoe, Purple: VFRoe+entropy fix.

Variables	States
γ	$5/3$
ρ	γ^2
p	γ
u_x	$-\sin(y)$
u_y	$\sin(x)$
u_z	0
B_x	$-\sin(y)$
B_y	$\sin(2x)$
B_z	0

TABLE 2. Initial states for the strong shock test case

4.3. **Speedup**

OpenCL is a generic tool for programing GPUs or multicore CPUs. It is thus possible to compare the same algorithm running on these two architectures or on single core of one CPU. We observe in Table 3 the following facts:

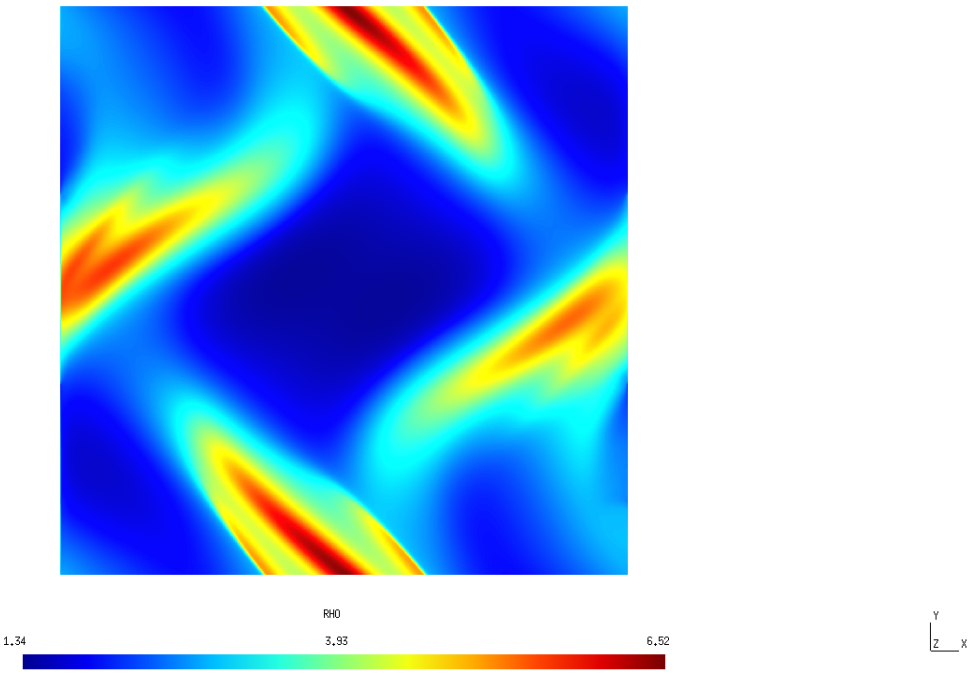

FIGURE 3. Density at time T=1, obtained with the Rusanov flux on a 4096x4096 grid for the Orzag-Tang's test case [7]

	Time	Speedup
Intel Xeon 2,3GHz (1 thread)	9.6 days	1
Intel Xeon 2,3GHz (24 threads)	18.8 hours	12
AMD Radeon HD 7970	2.4 hours	93

TABLE 3. Computation times and speedups obtained on different architectures

5. Conclusion

We have proposed an efficient algorithm, based on directional splitting, for solving the two-dimensional MHD equations. The resulting algorithm is very well adapted to GPU architecture, because it allows optimal bandwidth memory access. We have implemented the method in OpenCL. On the Orszag-Tang vortex case, with a rather fine mesh, we have obtained very interesting speedups compared to the sequential algorithm.

References

[1] François Bouchut, Christian Klingenberg, Knut Waagen, *A multiwave approximate Riemann solver for ideal MHD based on relaxation II*, Numer. Math., 115(4):647–679, 2010.

[2] P. Helluy, J.M. Hérard, H Mathis, S. Müller, *A simple parameter-free entropy correction for Approximate Riemann solver*, Comptes rendues mécanique ISSN 1631-0721, vol 338, no. 9, 493-498, 2010.

[3] C. Altmann, T. Belat, M. Gutnic, P. Helluy, H. Mathis, E. Sonnendrücker, W. Angulo, J-M. Hérrard, *A local time-stepping discontinuous Galerkin Algorithm for the MHD system*, ESAIM: Proc.28 33-54, 2009.

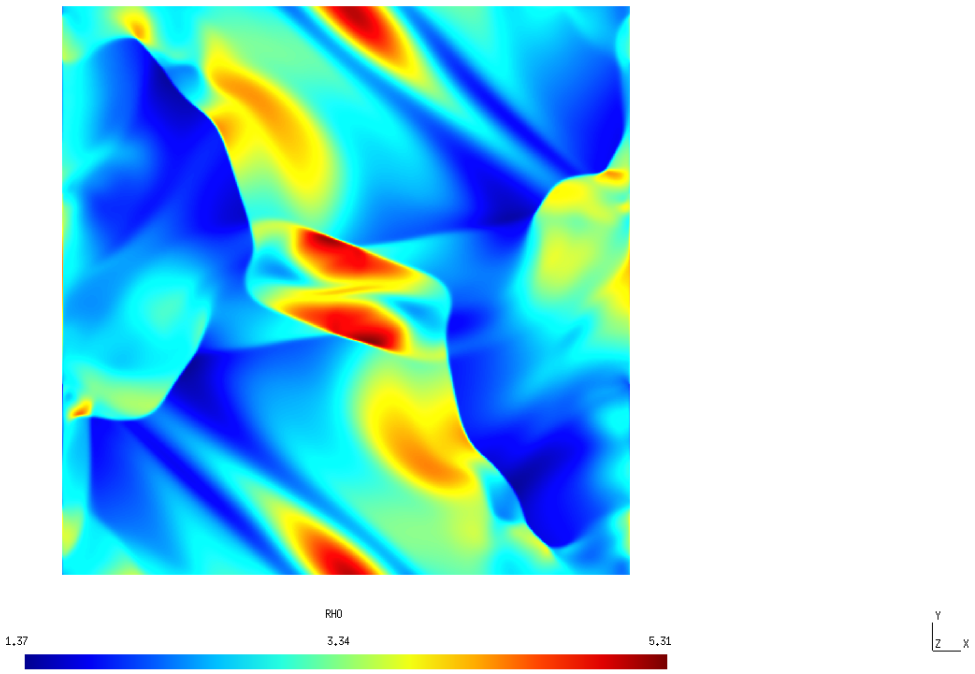

FIGURE 4. Density at time T=3, obtained with the Rusanov flux on a 4096x4096 grid for the Orzag-Tang's test case [7]

[4] P. HELLUY, JONATHAN JUNG, *OpenCL numerical simulations of two-fluid compressible flows with a 2D random choice method*, IJFV International Journal on Finite Volumes, 10:1-38, 2013.
[5] MASELLA, J.-M.; FAILLE, I.; GALLOUËT, T., *On an approximate Godunov scheme.*, Int. J. Comput. Fluid Dyn. 12, no. 2, 133–149, 1999
[6] DEDNER A., KEMM F., KROENER D., MUNZ C.-D., SCHNITZER T., AND WESENBERG M.*Hyperbolic divergence cleaning for the MHD equations. J. Comput. Phys., 175(2):645–673, 2002*
[7] ORZAG A., TANG C.-M., *Small-scale structure of two-dimensional magnetohydrodynamic turbulence*, J. Fluid Mech, vol 90, 129-143, 1979.
[8] G. RUETSCH, P. MICIKEVICIUS, *Optimizing matrix transpose in Cuda*, NVidia GPU Computing SDK, 1-24, 2009.