

Rust, OpenCL, GPU

Philippe Helluy

IRMA Strasbourg, Inria Macaron

June 2024

What is Rust ?

Rust (“rouille” in French) is a programming language created in 2009, using only old (rusty), but robust ideas. Some features, compared to C, C++ and Python, are:

- ▶ no memory leak or segfault, generally guaranteed at compile time [JJKD17];
- ▶ no uninitialized memory, no hidden type conversion (such as int to float, float32 to float64);
- ▶ no race conditions, generally guaranteed at compile time;
- ▶ strict ownership system, fast executable;
- ▶ Cargo, which replaces cmake, doxygen, ctest, anaconda, etc. in a single utility.
- ▶ "most loved programming language" in the Stack Overflow Developer Survey every year since 2016.

1D solver

Wave equation with unknown $u(x, t)$, $x \in \mathbb{R}$, $t \in]0, T[$,

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0, \quad u(x, 0) = u^0(x), \quad \frac{\partial u}{\partial t}(x, 0) = 0.$$

Leapfrog (“saute-mouton”) scheme

$$u_i^n \simeq u(x_i, t_n), \quad x_i = i\Delta x, \quad t_n = n\Delta t,$$

$$u_i^{n+1} = -u_i^{n-1} + 2(1 - \beta^2)u_i^n + \beta^2 (u_{i-1}^n + u_{i+1}^n), \quad (1)$$

where $\beta = \Delta t / \Delta x$.

Rust code

- ▶ Variables are immutable by default.
- ▶ An object passed to a function cannot be used anymore: use reference instead.
- ▶ Only one mutable reference or several immutable references allowed at a time.
- ▶ The compiler messages are generally helpful. Cargo clippy gives hints about what can be improved.
- ▶ It is recommended to use iterators for efficient and robust programs. Use of three different arrays for storing u_i^{n-1} , u_i^n and u_i^{n+1} .
- ▶ Automatic parallelism with the rayon library, without race condition.
- ▶ Source code at <https://github.com/phelluy/tutorust>

2D solver

We now consider the 2D case on a square. In 2D the leapfrog scheme reads:

$$p_{i,j}^{n+1} = -p_{i,j}^{n-1} + 2(1 - 2\beta^2)p_{i,j}^n + \beta^2 (p_{i-1,j}^n + p_{i+1,j}^n p_{i,j-1}^n + p_{i,j+1}^n). \quad (2)$$

Boundary conditions:

$$p_{0,j}^n = p_{1,j}^n. \quad (3)$$

$$p_{N+1,j}^n = p_{N,j}^n. \quad (4)$$

$$p_{i,0}^n = p_{i,1}^n. \quad (5)$$

$$p_{i,N+1}^n = p_{i,N}^n. \quad (6)$$

Code description

- ▶ We define a 2D array struct.
- ▶ Overloading of the `[][]` operator for immutable or mutable access.
- ▶ `template <T>`.
- ▶ 2D plot with Python.

OpenCL wrapper in Rust

- ▶ C bindings.
- ▶ Unsafe code.
- ▶ Memory management.
- ▶ One example of macro.
- ▶ Doc and testing with Cargo.
- ▶ Application to the wave equation on GPU.
- ▶ Source code at <https://github.com/phelluy/minicl>

Application to MHD

MHD equations:

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

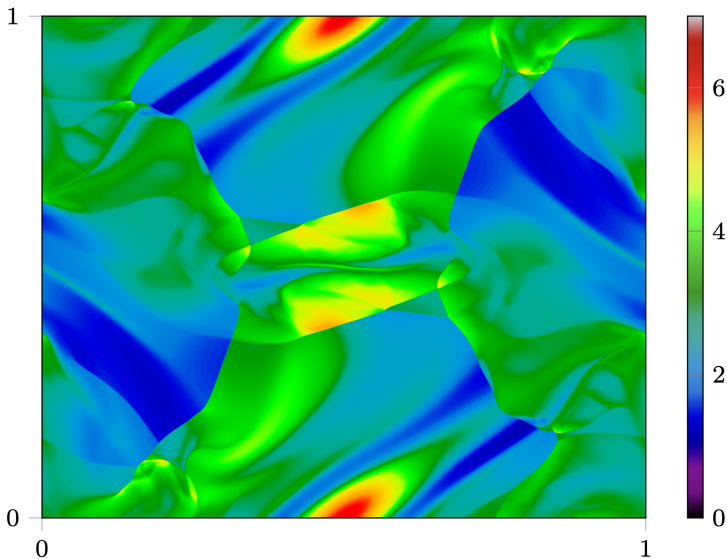
where the velocity and magnetic field are denoted

$\mathbf{u} = (u_1, u_2, u_3)^T$, $\mathbf{B} = (B_1, B_2, B_3)^T$, the pressure is given by a perfect-gas law with a constant polytropic exponent $\gamma > 1$

$$\rho = (\gamma - 1) \left(Q - \rho \frac{\mathbf{u} \cdot \mathbf{u}}{2} - \frac{\mathbf{B} \cdot \mathbf{B}}{2} \right).$$

Orszag-Tang test case

Grid size: 4096x4096 [BDH⁺23]



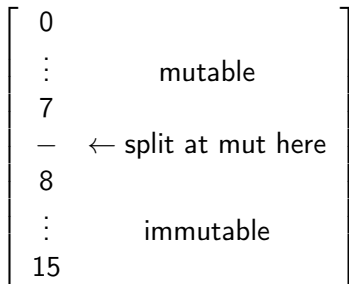
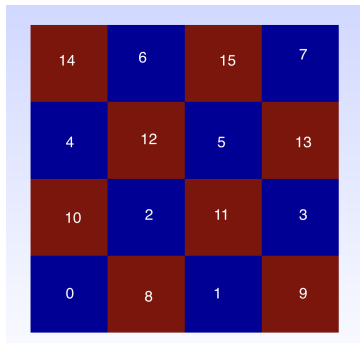
LBM with OpenCL

	prec.	b (GB/s, shift-only)	b (GB/s, shift-relax)	max. b (GB/s)
Intel	float32	17.58	13.38	60
Intel	float64	19.12	17.48	60
Iris 640	float32	26.20	24.98	34
Iris 640	float64	20.08	3.78	34
GTX	float32	147.54	146.94	192
GTX	float64	148.76	49.72	192
Quadro	float32	336.45	329.06	432
Quadro	float64	344.50	127.21	432
V100	float32	692.31	676.44	900
V100	float64	705.88	610.17	900

Table 2: Bandwidth efficiency of the LBM algorithm. Comparison of the data transfer rates of the shift-only algorithm and of the shift-and-relaxation algorithm. The resulting bandwidth is compared with the maximal memory bandwidth advertised by the vendors of the hardware devices.

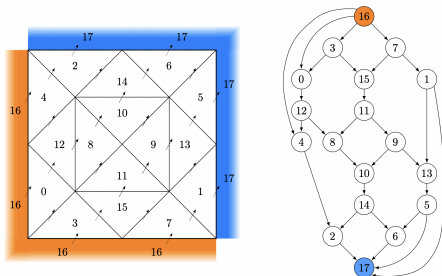
Rust automatic parallelism

- ▶ Resolution of a PDE on a regular mesh split into “boxes”. The computations on red boxes depend only on blue boxes and vice versa.
- ▶ Sort the array of boxes by color. Then “split at mut” the array: separate access to the blue and red boxes.
- ▶ Automatic parallelization, without race condition.



Upwind scheme in 2D or 3D

Dependency graph of the computations



- ▶ The solution can be explicitly computed by following a topological ordering of a Direct Acyclic Graph (DAG), e.g. 3, 7, 0, 15, 1, etc.
- ▶ In addition there is parallelism: (3,7) can be computed in parallel, then (0,15,1) can be computed in parallel, etc.
- ▶ Low storage: the solution can be replaced in memory during the computations.

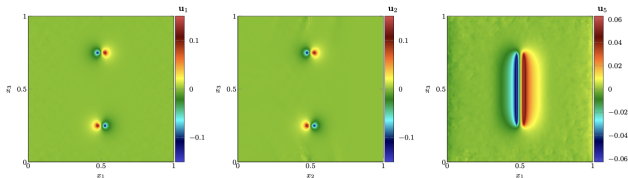
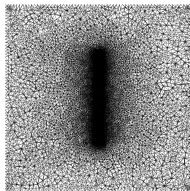
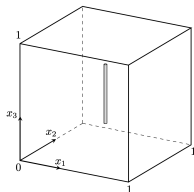
Rust implementation

We have implemented the upwind algorithm in Rust with the “split at mut” trick. More details in [GHMD21].

Method	CFL β	Δt	Error e_r		CPU (s)	
			$\nu = 2$	$\nu = 5$	1 thread	24 threads
RK3DG	0.37	0.00009	0.00070	0.01238	4,607.95	785.28
D3Q4P	0.37	0.00009	0.00103	0.01467	1,524.45	234.48
RK3DG	0.93	0.00023	0.00070	0.01238	2,189.76	384.79
D3Q4P	0.93	0.00023	0.00103	0.01467	613.44	90.84
RK3DG	1.85	0.00046	0.00070	0.01238	1,121.96	212.60
D3Q4P	1.85	0.00046	0.00103	0.01467	304.41	45.14
D3Q4P	3.70	0.00091	0.00103	0.01468	153.09	22.40
D3Q4P	9.25	0.00228	0.00104	0.01479	61.60	8.96
D3Q4P	18.50	0.00456	0.00115	0.01619	30.76	4.53
D3Q4P	37.00	0.00912	0.00210	0.02992	15.34	2.46
D3Q4P	92.50	0.02281	0.01107	0.16589	6.17	0.92
D3Q4P	185.00	0.04562	0.04509	0.40344	3.10	0.48

Application to an electromagnetic solver

- ▶ The transport solver is the building block of our CFL-less scheme for conservation laws.
- ▶ Unstructured mesh of the unit cube made of large and small cells. A small electric wire at the middle of the mesh.
- ▶ Resolution of the Maxwell equations.



(a) Solution at time $t = 0.75$; left panel: $E_1|_{x_2=0.5}$; middle panel: $E_2|_{x_1=0.5}$; right panel: $H_2|_{x_2=0.5}$.

“Kyoto” test-case

Maxwell, 6 millions tetrahedrons [GHMDW24].

solver	hardware platform	computation time
CLAC	6 × Nvidia GeForce GTX 1080 Ti single precision arithmetic	31 h
KOUGLOFV	AMD EPYC 7713x2, 128 cores, 2 GHz double precision arithmetic	20 h

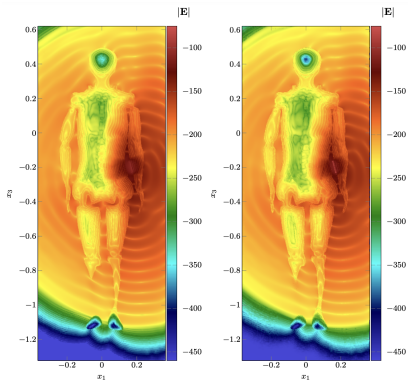


FIGURE 6.9. Numerical approximation of $|E|$, sliced in the (x_1, x_2) plane, at $t = 0.9$ ns. Left panel: results from CLAC; right panel: results from KOUGLOFV.

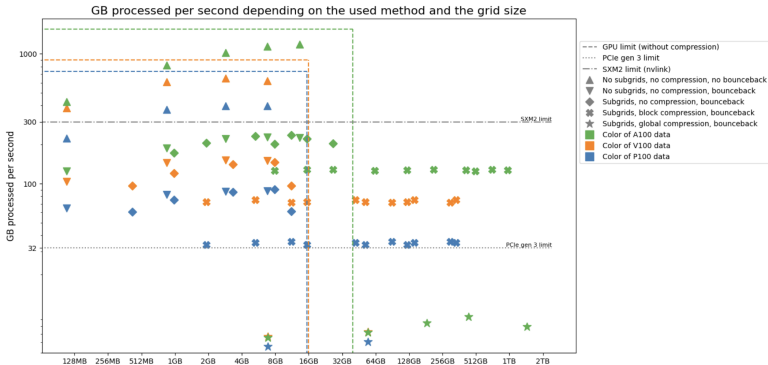
Conclusion

- ▶ Practical use of Rust in a scientific computing context.
- ▶ GPU computing support through OpenCL: works on Linux, Windows, Mac.
- ▶ Less bugs, which was the objective.
- ▶ Automatic, fast and robust parallelism.
- ▶ Friendly environment.
- ▶ Many other features, which we have not yet explored.
- ▶ But if I had to develop a big project, Kokkos is probably a much safer choice !

Post scriptum: on-the-fly memory compression

PhD thesis of Clement Flint: CFD on a structured grid. The grid does not enter the GPU memory.

- ▶ Wavelet compression for virtually increasing the memory.
- ▶ By turns, each block is uncompressed, processed and recompressed (in cache memory)
- ▶ C++ and CUDA (for the moment)



Bibliography I



Hubert Baty, Florence Druil, Philippe Helluy, Emmanuel Franck, Christian Klingenberg, and Lukas Thanhäuser.

A robust and efficient solver based on kinetic schemes for magnetohydrodynamics (mhd) equations.

Applied Mathematics and Computation, 440:127667, 2023.



Pierre Gerhard, Philippe Helluy, and Victor Michel-Dansac.

Cfl-less discontinuous galerkin solver.
2021.



Pierre Gerhard, Philippe Helluy, Victor Michel-Dansac, and Bruno Weber.

Parallel kinetic schemes for conservation laws, with large time steps.

Journal of Scientific Computing, 99(1):5, 2024.



Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer.

Rustbelt: Securing the foundations of the rust programming language.

Proceedings of the ACM on Programming Languages, 2(POPL):1–34, 2017.