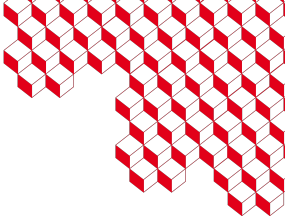
UFR de mathématique et d'informatique
Université de Strasbourg



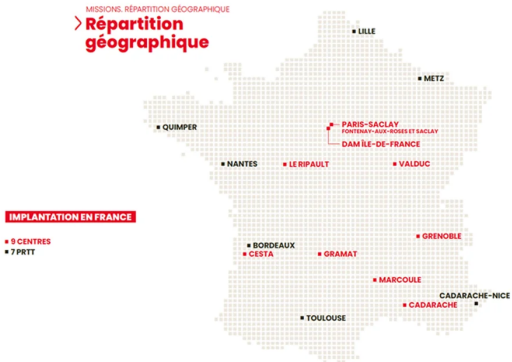
Development of algorithms to compute the distances to an interface in an HPC code

Intern: Demuth Axel
Tutors : Gilet Nicolas, Girardin Mathieu

27,28 August 2025
Internship presentation

Presentation of the CEA

- Defense and Security;
- Energy;
- Digital Transition;
- Health Technologies;
- Fundamental Research;

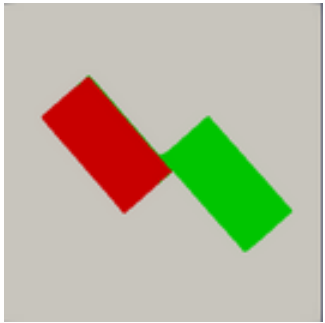


Context

The **hydrodynamic code ARMEN** is developed at CEA to perform simulations with multiple materials :

- A numerical model is used to better account for the slipping between materials. It requires to know the **distance to the interface**.
- The **euclidean norm** is used and is **extremely time consuming** especially in 3D as it's complexity is $\mathcal{O}(N_{mesh} \times N_{interface})$.

N_{mesh} : number of cells in the mesh.
$N_{interface}$: number of points in the interface that is reconstructed on the mesh.



To **reduce this computation time**, I used an other method to compute the distance to the interface based on solving the **eikonal equation** using numerical methods called **Fast Methods**.

Eikonal equation and *Fast Methods*

Implementation and numerical results

ARMEN extension

Conclusions et Perspectives

Eikonal equation

A **simplified form of the eikonal equation** is widely used for calculating the smallest distance in fields such as seismology and tomography. It reads

$$\begin{cases} |\nabla T(x)| = 1 & \text{for } x \in \Omega \subset \mathbb{R}^D \\ T(x) = 0 & \text{for } x \in X_s \subset \Omega \end{cases} \tag{1}$$

- $T(x)$: the **distance to the source points**.
- X_s : **the source points**.
- Ω : the domain.

Indeed, we consider a **constant local velocity equal to** 1 so that **the time to reach a point is equal to the distance to this point**.

A simple **upwind scheme** is used to **solve the eikonal equation**.

Upwind scheme (1/2)

The partial derivatives of T using the upwind scheme can be approximated as

$$\begin{aligned} \frac{\partial T}{\partial x} &\approx \max\left(D_{ij}^{-x}T, -D_{ij}^{+x}T, 0\right) \\ \frac{\partial T}{\partial y} &\approx \max\left(D_{ij}^{-y}T, -D_{ij}^{+y}T, 0\right) \end{aligned} \tag{2}$$

where

$$\begin{aligned} D_{ij}^{\pm x}T &= \frac{T_{i\pm1,j} - T_{i,j}}{\pm\Delta x} \\ D_{ij}^{\pm y}T &= \frac{T_{i,j\pm1} - T_{i,j}}{\pm\Delta y} \end{aligned} \tag{3}$$

$$|\nabla T(x)| = \sqrt{\left(\frac{\partial T}{\partial x}\right)^2 + \left(\frac{\partial T}{\partial y}\right)^2} = 1 \approx \sqrt{\max\left(D_{ij}^{-x}T, -D_{ij}^{+x}T, 0\right)^2 + \max\left(D_{ij}^{-y}T, -D_{ij}^{+y}T, 0\right)^2} = 1 \tag{4}$$

Which can be rewritten

$$\max\left(\frac{T_{i,j} - \min(T_{i+1,j}, T_{i-1,j})}{\Delta x}, 0\right)^2 + \max\left(\frac{T_{i,j} - \min(T_{i,j+1}, T_{i,j-1})}{\Delta y}, 0\right)^2 = 1 \tag{5}$$

Upwind scheme (2/2)

To simplify the notation, let us denote :

- $T = T_{ij}$
- $T_x = \min(T_{i+1,j}, T_{i-1,j})$
- $T_y = \min(T_{i,j+1}, T_{i,j-1})$

We suppose that T is always greater than T_x and T_y, which will be discussed later on, we end up with

$$\left(\frac{T - T_x}{\Delta x}\right)^2 + \left(\frac{T - T_y}{\Delta y}\right)^2 = 1 \tag{6}$$

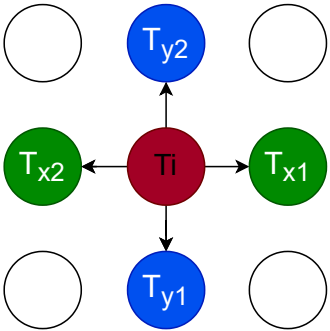
which is nothing but

$$\frac{T^2 - 2T * T_x + T_x^2}{\Delta x^2} + \frac{T^2 - 2T * T_y + T_y^2}{\Delta y^2} = 1 \tag{7}$$

We recognize a quadratic equation of the form $aT^2 + bT + c = 0$ with:

- $a = \Delta x^2 + \Delta y^2$
- $b = -2(\Delta y^2 T_x + \Delta x^2 T_y)$
- $c = (\Delta x T_y)^2 + (\Delta y T_x)^2 - \Delta x^2 \Delta y^2$

Eikonal equation numerical scheme

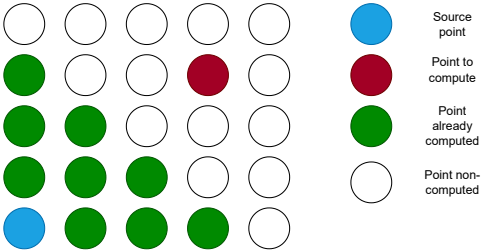


The eikonal equation numerical scheme to compute T_i is composed of two steps

- We get $\min(T_{x1}, T_{x2})$ and $\min(T_{y1}, T_{y2})$ to compute a,b,c
- We solve the quadratic equation $aT_i^2 + bT_i + c = 0$

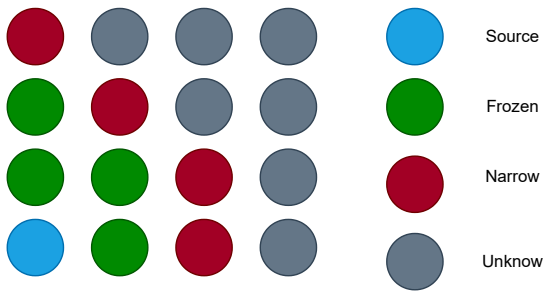
We need only to know $\min(T_{x1}, T_{x2})$ and $\min(T_{y1}, T_{y2})$ to use this scheme, this will be done using a careful choice of **grid exploration from the source points**.

Grid exploration



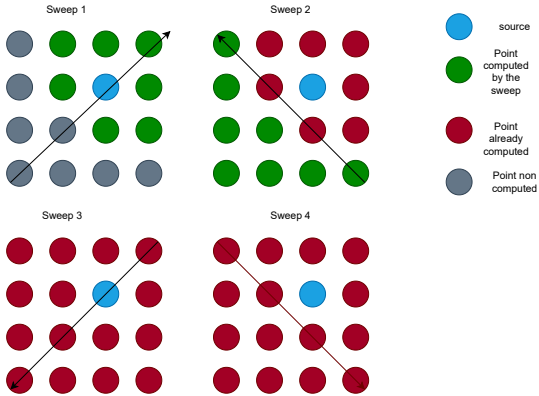
- Idea : **propagate the information from all source points simultaneously at the same speed**.
- We will use methods call **Fast Methods**.

Fast Marching Method (FMM) and Fast Iterative Method (FIM)



- 3 categories of points : Frozen, Unknow, Narrow
- We solve the eikonal equation for points in Narrow
- Neighbouring points of Narrow from Unknow are added to Narrow
- Once a point of Narrow converges, he is passed to Frozen
- The difference between both methods resides in the process of handling the Narrow band,

Fast Sweeping Method (FSM)



- No categorization of points
- We sweep the mesh in pre-define directions
- We solve the eikonal equation on each point of the mesh during each sweep

Eikonal equation and *Fast Methods*

Implementation and numerical results

ARMEN extension

Conclusions et Perspectives

Implementation and first numerical results

Implementation :

- Setting up of a **C++ code implementing the FMM, FIM and FSM in 2D and 3D** with a python visualisation
- Extension of this C++ code to **multi-threading parallelization (OpenMP)**.

All three methods give good results on a simple test case with five source points generated randomly :

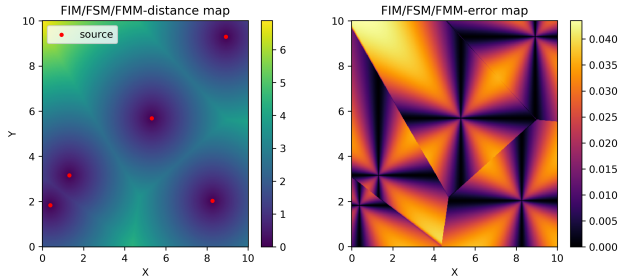


Figure 1: Distance map (left) and error (right) for the FMM/FIM/FSM with five sources points in red, for mesh size $N = 400 * 400$.

Test case of a simple interface

The second test case is made of a simple interface, once again all three methods give good results :

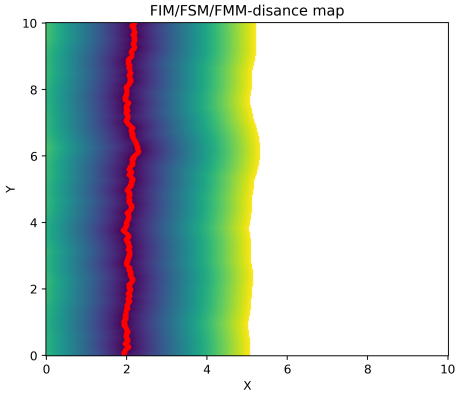


Figure 2: Distance map for the FMM/FIM/FSM for mesh size $N = 400 * 400$.

- We introduced a **threshold on the distance to limit our grid exploration**.
- This is a nice feature of the **Fast Methods** when compared to the **euclidean norm**.

CPU time with 50 sources generated randomly

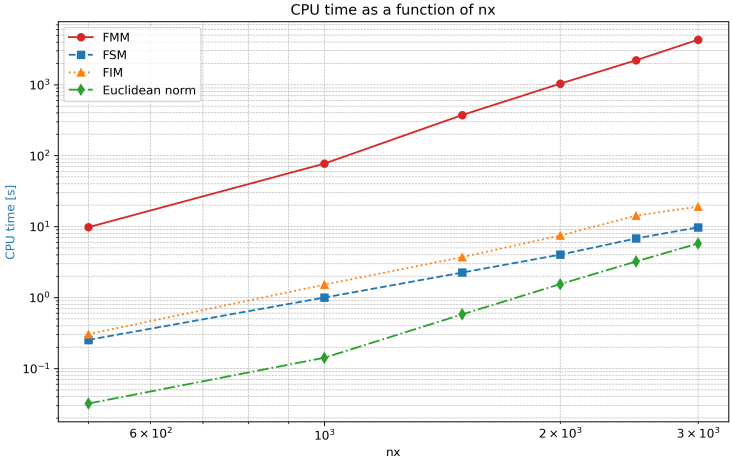
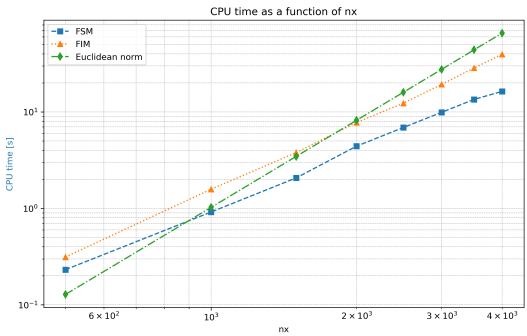


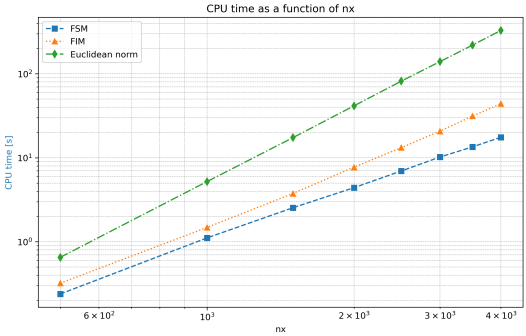
Figure 3: CPU time of FMM (red curve), FSM (blue curve), FIM (orange curve) and euclidean norm (green curve) for different mesh sizes from 500x500 to 4000x4000, with $n_s = 50$.

CPU time with 200 and 1000 sources generated randomly

.



(a) $n_s = 200$



(b) $n_s = 1000$

Figure 4: CPU time of FSM (blue curve), FIM (orange curve) and euclidean norm (green curve) for different mesh sizes from 500x500 to 4000x4000, with $n_s = 200$ (a), $n_s = 1000$ (b)

Influence of the number of sources on the CPU time

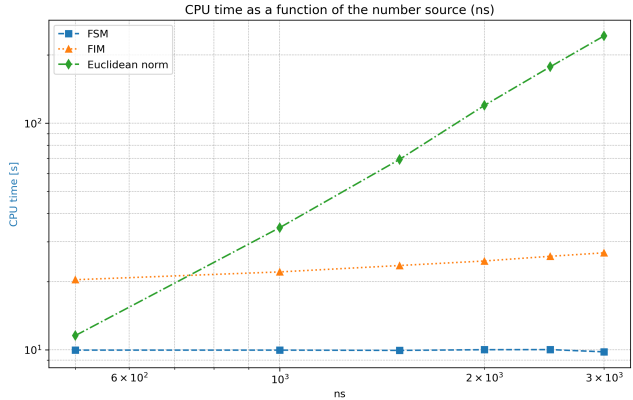


Figure 5: CPU time of FSM (blue curve), FIM (orange curve) and Euclidean method (green curve) for different number of sources X_s from 500 to 3000.

Influence of the threshold on the CPU time

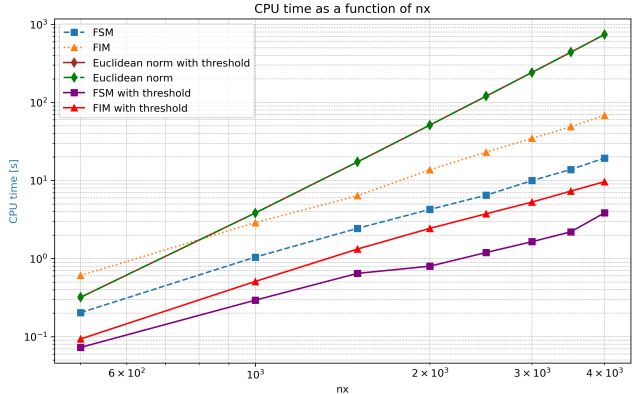


Figure 6: Comparaison of CPU times for FIM/FSM/euclidean method on an interface with (respectively orange, blue, green curve) and without a threshold (respectively red, purple, brown curve)

OpenMP Parallelization

OpenMP strategies for grid exploration :

- FSM : each thread perform a sweep on a copy of the mesh, them we take the minimum value on each point (reduction operation)
- FIM : each thread take a equal part of sources and perform FIM on it

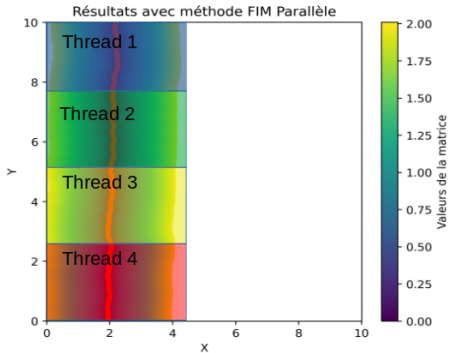


Figure 7: FIM thread partition representation

OpenMP results on the simple configuration with 5 sources generated randomly

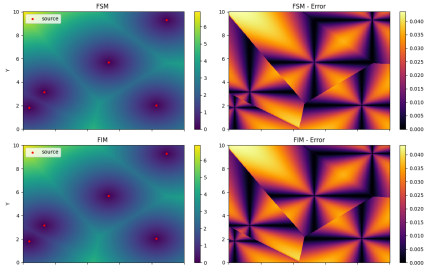


Figure 8: Result with N = 400x400, 5 interfaces, 8 threads for FIM and Euclidean, 4 threads FSM

Number of OMP Threads	1	2	4	8	12
CPU TIME FIM [s]	4.65	2.3	0.965	0.345	0.315
CPU TIME FSM [s]	0.46	0.36	0.225	0.258	0.297

Table 1: Table of CPU times for FIM and FSM based on the number of OMP threads.

OpenMP results on the simple interface with a threshold test case

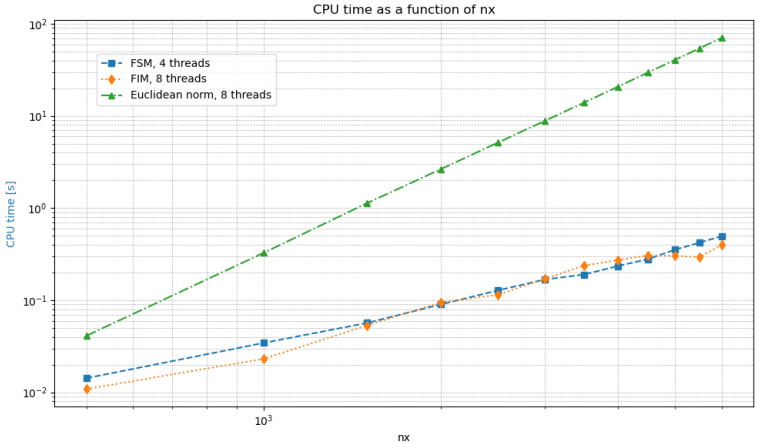


Figure 9: CPU time for FIM with 8 threads (orange), FSM with 4 threads (blue) and euclidean norm with 8 threads (green) for the simple interface test case with a threshold of 100 steps in each direction around the interfaces.

Eikonal equation and *Fast Methods*

Implementation and numerical results

ARMEN extension

Conclusions et Perspectives

Presentation of ARMEN

The hydrodynamic code ARMEN features several numerical solvers to solve the gas dynamics equations that reads

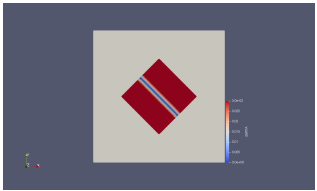
$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \\ \dfrac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) + \nabla p = \mathbf{0} \\ \dfrac{\partial E}{\partial t} + \nabla \cdot (\mathbf{u}(E + p)) = 0 \end{cases}$$

where:

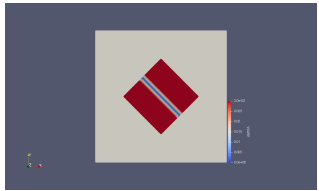
- ρ is the density,
- $\mathbf{u}$ is the velocity vector,
- p is the pressure,
- E is the total energy per unit volume, $E = \rho e + \dfrac{1}{2}\rho \|\mathbf{u}\|^2$, with e being the internal energy per unit mass.

There is also the possibility to perform simulation for solids to some extent using the Wilkins hypoelastic model.

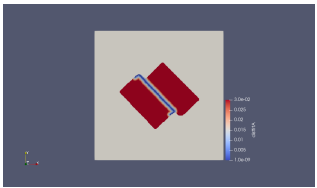
A slip between two materials



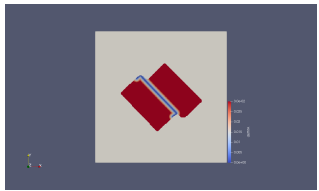
(a) Time step 0 FIM



(b) Time step 0 euclidean



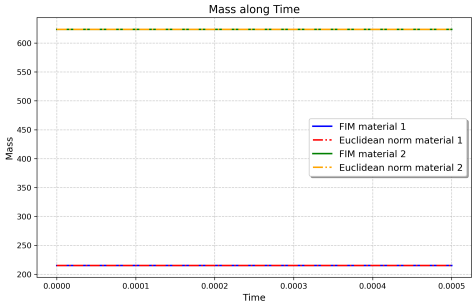
(c) Time step 10 FIM



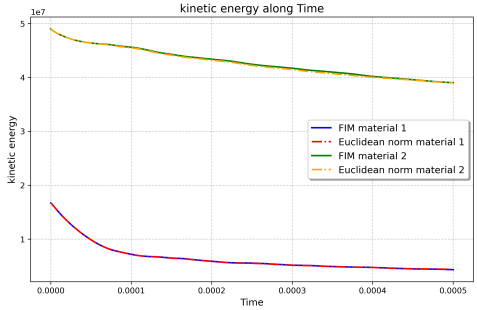
(d) Time step 10 euclidean

Figure 10: Distance map of FIM/euclidean on two different time steps(0,10) for the slip between two materials with a colorbar of the distance.

Metric comparaison



(a) Mass along time of each material with distance at interface computed by FIM and euclidean norm

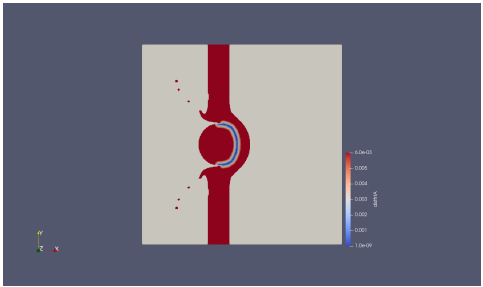


(b) Kinetic energy along time of each material with distance at interface computed by FIM and euclidean norm

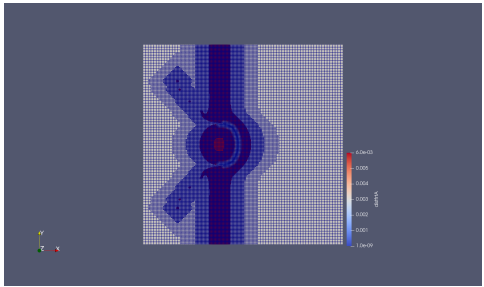
grid step	0.1	0.05	0.01	0.005	0.0025	0.00125
n_s at the start	8	15	60	172	253	485
CPU TIME FIM [s]	1.39e-01	4.24e-01	9.8e+00	4.26e+01	1.839e+02	7.32e+02
CPU TIME Euclidean norm [s]	1.36e-01	4.55e-01	9.69e+00	4.70e+01	1.95e+02	9.47e+02

Table 2: Table of CPU times for FIM and euclidean norm on the grid step.

Sphere impacting a plate with Adaptive Mesh Refinement



(a) FIM - Distance to the interface



(b) AMR mesh

Figure 12: Illustration of the sphere impacting a plate configuration at a time step when the sphere has impacted the plate.

Metric comparaison

Initial speed	1000	800	600	400	200
Final speed - FIM	842.5	655.3	455.1	197.6	94.9
Final speed - euclidean norm	841.1	652.9	452.1	198.5	94.8

Table 3: Final speed of the sphere obtained with FIM and euclidean norm.

Eikonal equation and *Fast Methods*

Implementation and numerical results

ARMEN extension

Conclusions et Perspectives

Conclusions and Perspectives

Conclusion:

- Implemented **3 fast methods** in 2D and 3D, with **FIM** and **FSM** achieving significantly faster CPU times for large meshes and great interfaces compared to the Euclidean norm.
- Study of parallelization in a sub-domain, with the **FSM** being the most efficient and the **FIM** the most generalizable.
- Implementation of FIM openmp algorithm in ARMEN, and validated it by comparison to the Euclidian norm on two test cases.

Future works:

- Implementation of FIM in 3D in ARMEN
- Implementation of the MPI parallelization.

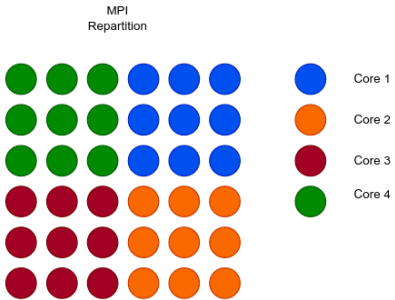
Thanks for your Attention !

Any Questions

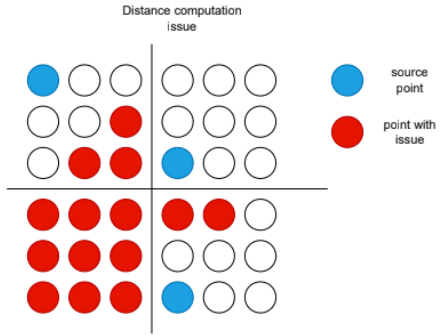
Intern : Demuth Axel, University of Strasbourg

Supervisors CEA : Gilet Nicolas, Girardin Mathieu

Appendix - MPI



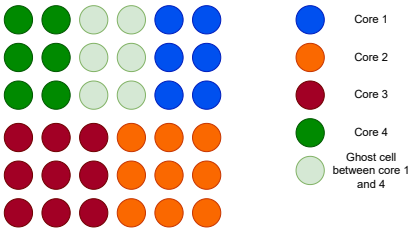
(a) Repartition of the mesh along the MPI core.



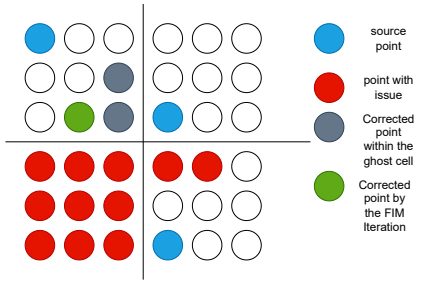
(b) Representation of all point with an issue with the MPI repartition

Figure 13: illustration of the MPI repartition and the difficulties

Appendix - MPI



(a) Representation of ghost cells between core 1 and core 4



(b) Representation of the correction from the ghost cells between core 1 and core 4.

OpenMP speedup on the simple interface with a threshold test case

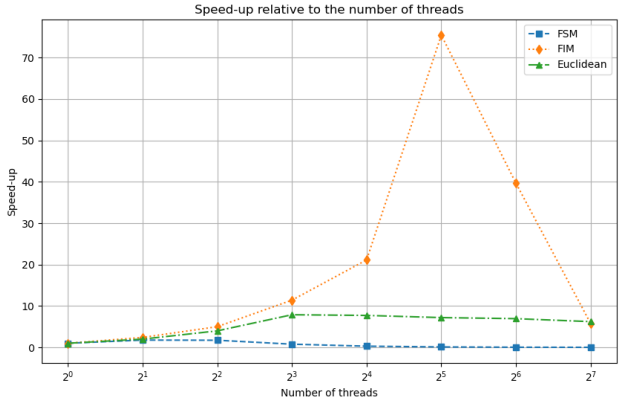


Figure 15: Speed-up of FSM(blue curve), FIM (orange curve), euclidean norm (green curve) on with a threshold, with $N = 4000 \times 4000$ and $n_s = 4000$

3D results

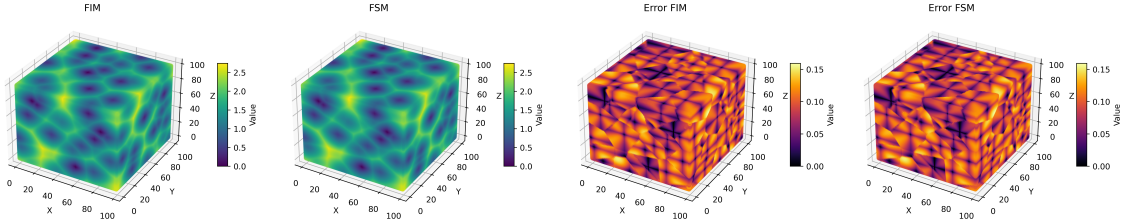


Figure 16: Distance map and error for the FIM and FSM with 200 sources points, for a mesh of $N = 100 * 100 * 100$.