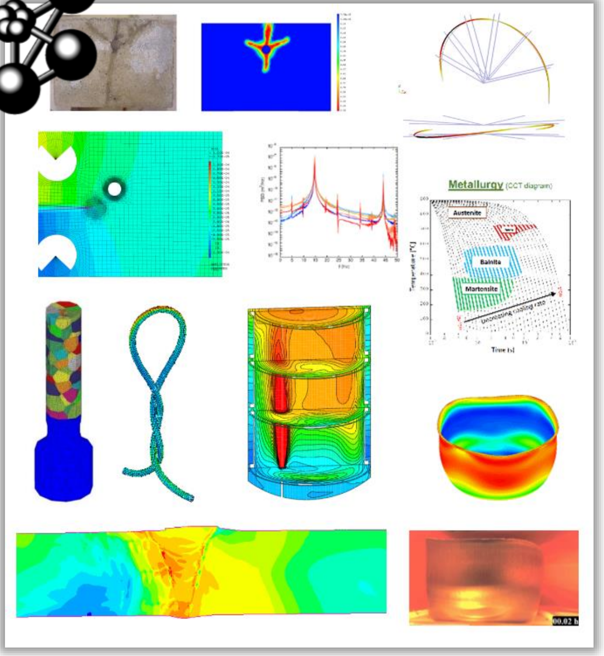
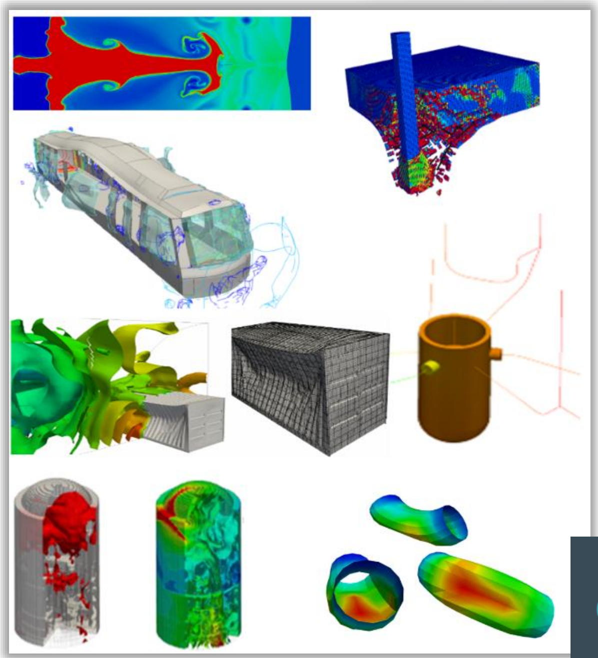
MANTA – the CEA’s future platform for simulations in structural mechanics and their interactions

1. Context & Objectives

Legacy softwares

- Lot of **functionalities**
- Address **today industrial problems**
- **Mature** and **robust**





- **Technical debt**
  - Difficult to evolve and maintain
- Computational **performances limited**
  - No more extensible

Numerical simulation of the **mechanics of structures** and their **interactions** for **civil nuclear applications**, under **nominal (Cast3m)** **and accidental (EPX) conditions**

Main objectives

2030: industrial operation

- **Explicit dynamics** for **structures** and **compressible fluids**
- **Fluid / structure interactions**
- **Industrial applications**
- **Finite-elements**, **finite-volumes**, sph, discrete element method
- ~40 years of development

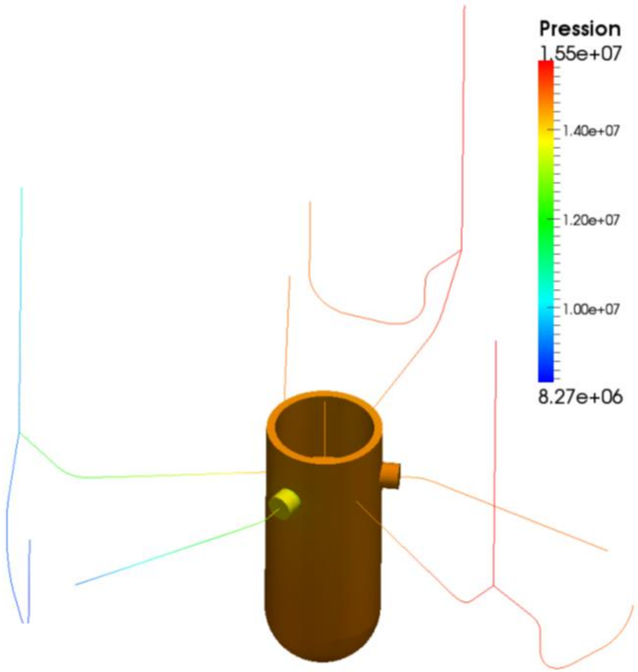
- **Generic tool** for "**implicit problems**"
- **Mainly geared for (non-linear) mechanics**
- **… but also applied to** incompressible fluids, electromagnetism, metallurgy, …
- **Industrial applications**
- **Finite-elements**
- ~40 years of development

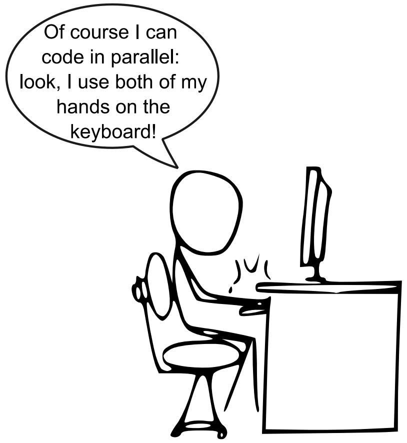
MANTA

- Next gen., **HPC** oriented
- Structure / compressible fluids / … , interactions
- Industrial applications
- Every mesh-based method (FE, FV, HDG, …)
- C++
- "automatic parallelism"
- Easy to maintain and evolve on the long term
- Open-source

Software engineering objectives

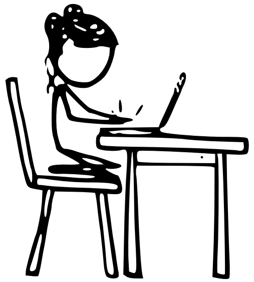
- Target industrial applications
  - Multi PDE
  - Lagrangian, Eulerian, ALE approaches
  - Multi “areas” (more general than “multi-material”: may overlap, not cover the whole mesh, …)
  - Multi topological dimension (Volume, shell, beam elements in a single calculation)
  - Various geometrical supports (tetrahedral, hexahedra, prims, pyramids, quadrangles, triangles, segments)
  - Very high “flexibility”, which may affect performances
  - Implicit & explicit problems
- HPC
  - Native distributed parallelism
  - Total distribution of the data, workload
    - No specificity of the process 0
    - No array of size O(global numerical model size)
  - Performance portability: ability to adapt to various hardware architectures (GPU, ARM, …)
- “Automatic parallelism”
  - Feedback from EPX: strong requirement. the code features must be able to be extended and maintained by developers knowing almost nothing about parallelism
  - Code new functionality “as in a sequential code”, and works in //





Users and APIs

- Different kind of users
- An API suited to each



- High stability of the API
- Less risk of doing mistakes

Python user API

JSON data file

C++ “expert” API

C++ “user” API

Numerical method Layer

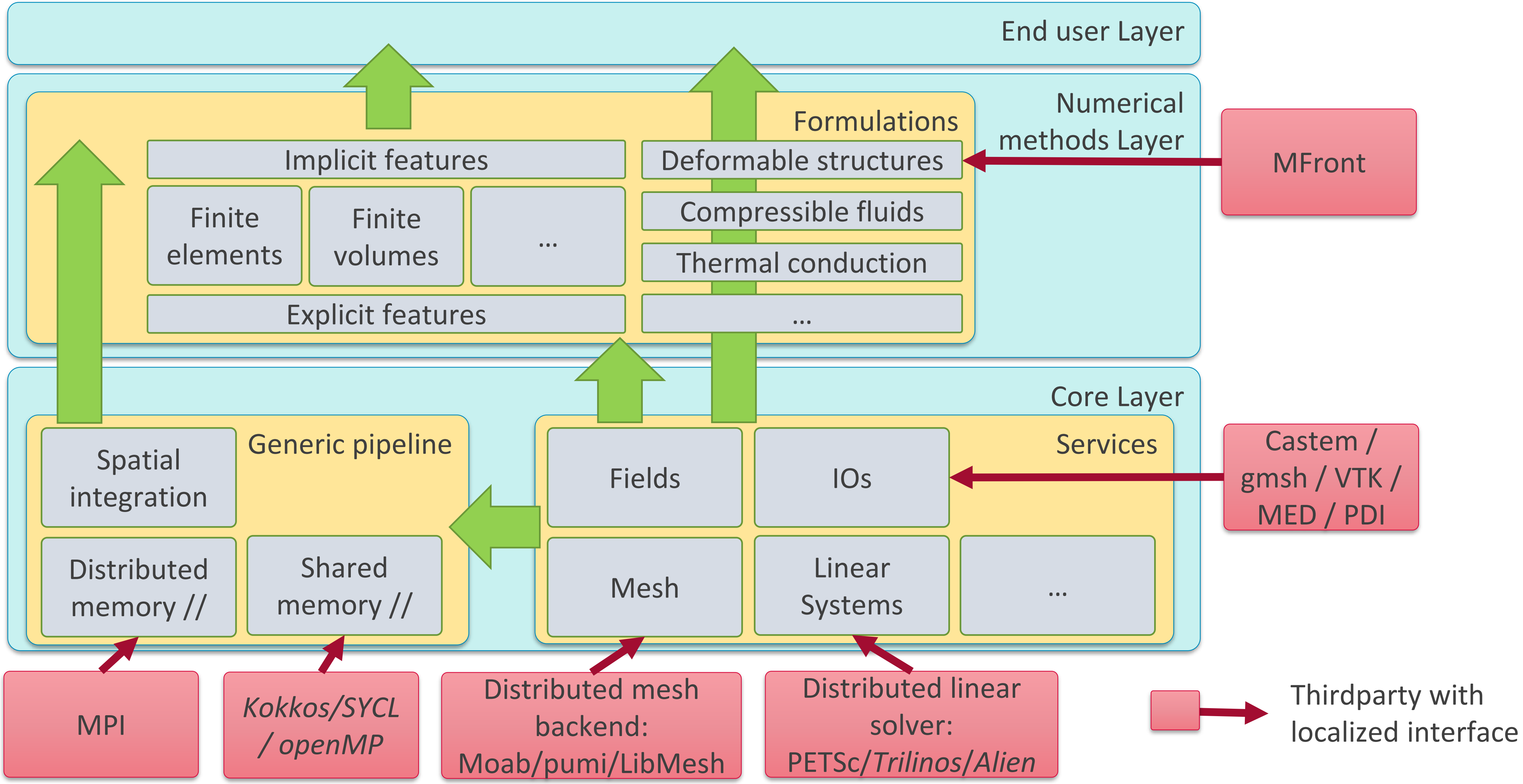
Plugins C++

Core Layer

- High level of control and flexibiliy

2. Design constraints for HPC

Layers


End user Layer
Numerical methods Layer
Formulations
Implicit features
Finite elements
Finite volumes
…
Explicit features
Deformable structures
Compressible fluids
Thermal conduction
…
MFront
Core Layer
Generic pipeline
Spatial integration
Distributed memory //
Shared memory //
Services
Fields
IOs
Mesh
Linear Systems
…
Castem / gmsh / VTK / MED / PDI
MPI
Kokkos/SYCL / openMP
Distributed mesh backend: Moab/pumi/LibMesh
Distributed linear solver: PETSc/Trilinos/Alien
Thirdparty with localized interface

Genericity: the "pipeline"

- Purpose
  - **Assemble distributed linear systems** resulting from **spatial integration on unstructured meshes**
  - Attach "constraints" to linear systems
  - **Solve the (saddle point problems) linear systems**
  - Support all the parallelism

$$\begin{bmatrix} A & C0^t & C1^t & \cdots \\ C0 & 0 & 0 & \cdots \\ C1 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} X \\ \lambda 0 \\ \lambda 1 \\ \vdots \end{bmatrix} = \begin{bmatrix} B \\ D0 \\ D1 \\ \vdots \end{bmatrix}$$

- **Assembling**: spatial integration over (possibly non-conforming) **unstructured meshes**
  - Split global integral over mesh entities: $\boldsymbol{M} = \sum_i \mathcal{A}_i \int_{E_i} \boldsymbol{m}(\underline{x})\, dx$
  - Use finite-element mapping with reference element to integrate using standard quadrature formulae:

$$\boldsymbol{M} = \sum_i \mathcal{A}_i \sum_j w_j \boldsymbol{m}(\underline{\xi}_j) |det(\underline{\phi}_i(\underline{\xi}_j))| \text{ , where } (\underline{x} \in E_i) = \underline{\phi}_i(\xi)$$

  - Programming of actual problems through **entry points**:
    - Integrand::addOn → $w_j \boldsymbol{m}(\underline{\xi}_j) |det(\underline{\phi}_i(\underline{\xi}_j))|$
    - Assembler::assemble → $\mathcal{A}_i$

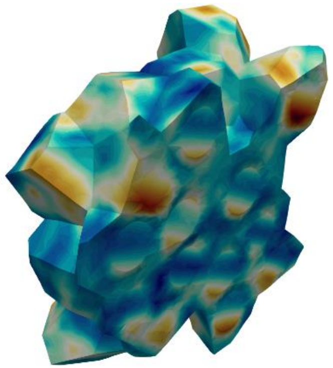
- Adverse impact on sequential and // performances
  - **No predetermined algorithmic motif**, very few assumptions in the generic pipeline about what the terminal code will do.
  - Multi-zone, multi-PDEs**: lots of indirections**, **complex memory layout**
  - **Unstructured meshes**

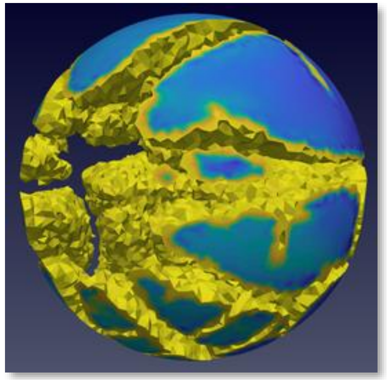
“Automatic parallelism”

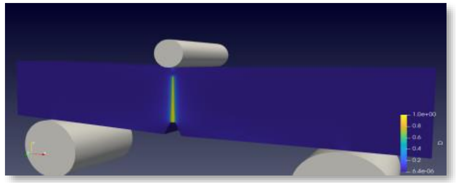
- “Automatic” parallelism: code terminal problems as in sequential
  - Generic pipeline: implement everything through the entry points & core tools
  - Ghosting
    - Each process can replicate any mesh cell owned by another process → ghost cell
    - When imported, a ghost entity carries all the data it is related to (e.g. MeshSet belongings), and recursively for its lower dimensional entities (may induce an excess of communication volume)
    - A ghost entity (as a local one) should be the same as in sequential
    - Functions to synchronize field values on ghost entities
- Adverse impact on sequential and // performances
  - No specific tailored optimization for each problem
  - Over-abundance of data transferred when importing cells as ghosts

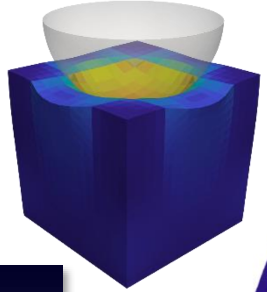
cea

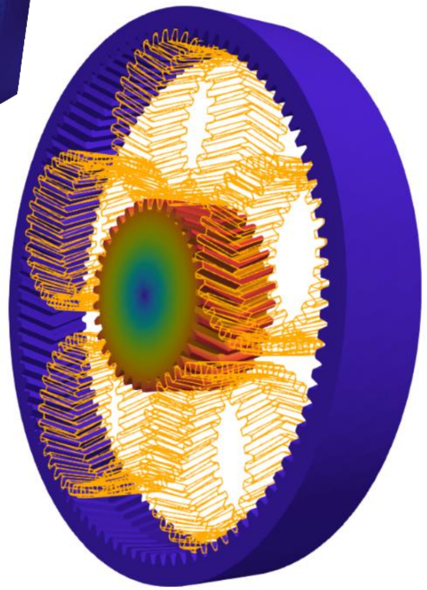
A few illustrations

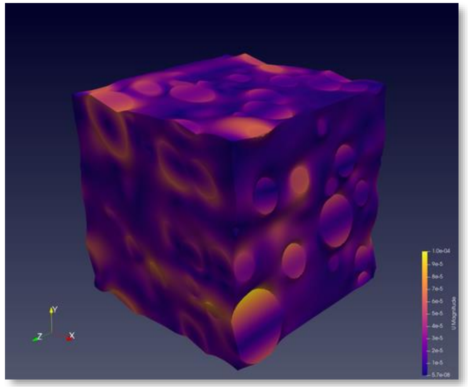




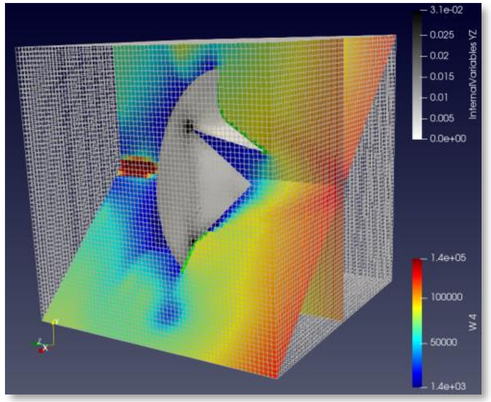


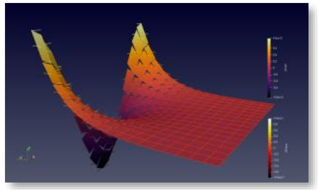


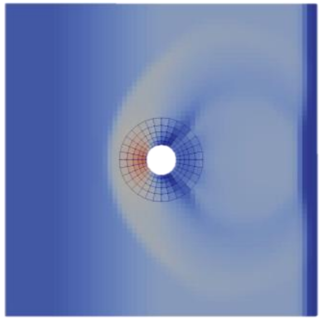


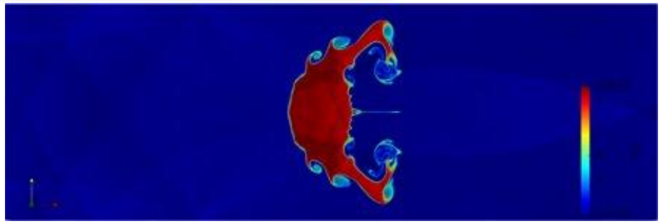


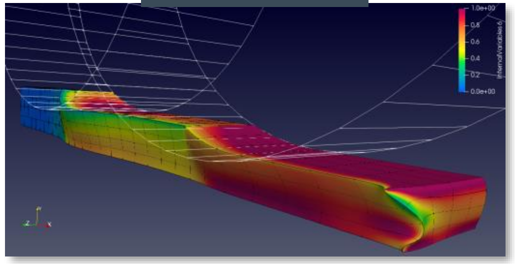
complex behaviors
steel-concrete bond
structural mechanics
chimera method
implicit/explicit time integration
eulerian/lagrangian methods
complex boundary conditions
modal solver
phasefield damage
multicomponent flows
finite volume
linear/quadratic spatial approximation
solid and structural elements (beam, shells)
ersed fluid structure interaction
ite strains
elements
compressible fluid dynamics
contact mechanics
heat transfer
1D/2D/3D problems
hybrid high order method
hdg
stokes problem

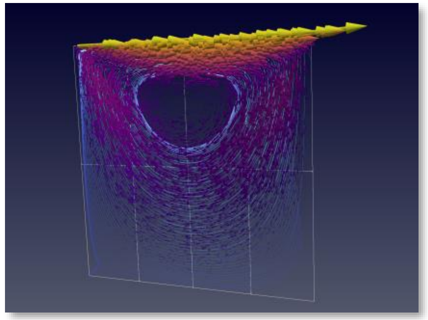


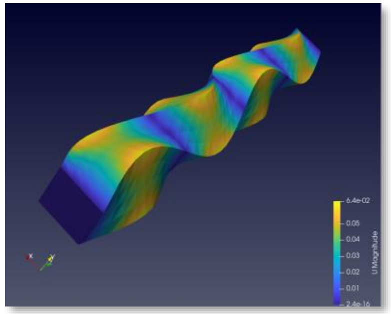






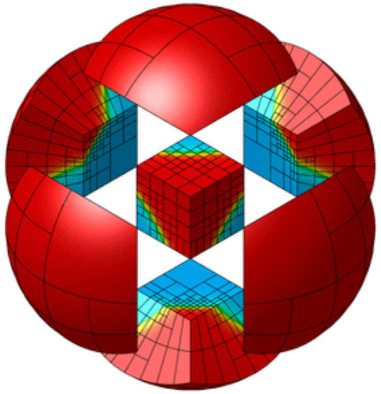






Some tools

HPC benchmark

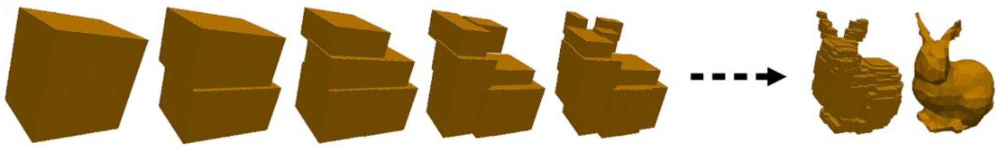


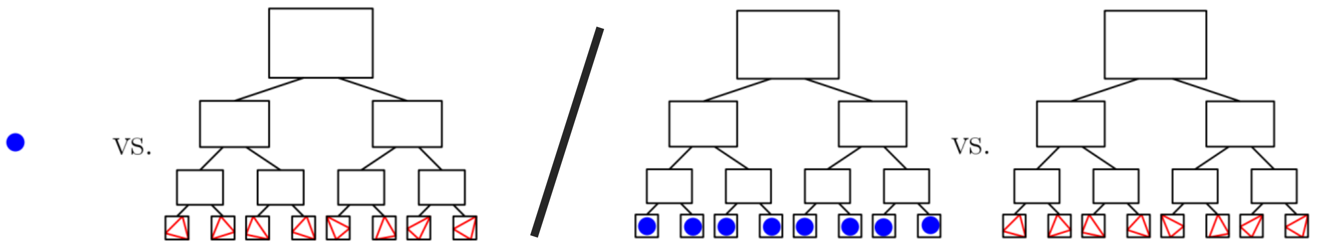
Languages & compilers

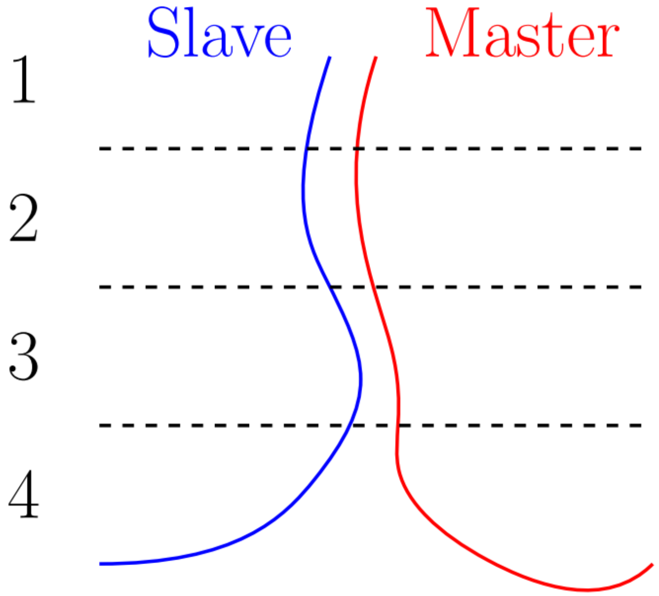
Collaborative workflow

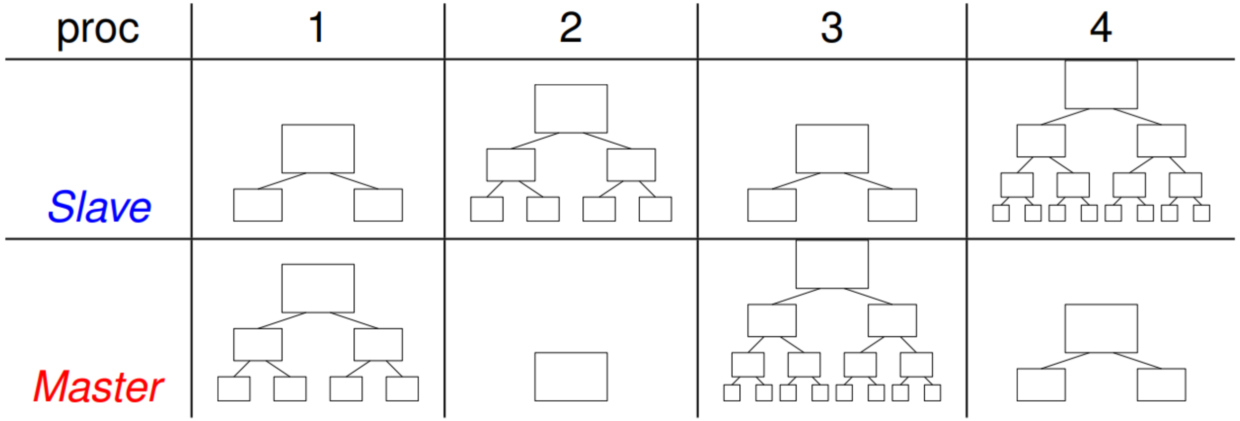
3. Roadmap & some directions for HPC

Geometrical intersections detection with distributed parallelism









On doit réaliser $S_i \cap M_j \quad \forall i, j \in [\![1, p]\!]$

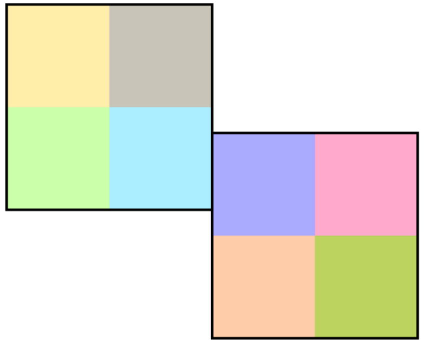
$S_i \cap M_i$ est séquentiel, $S_i \cap M_j$ est parallèle $\forall i \neq j$

- Antoine Motte’s PhD Thesis

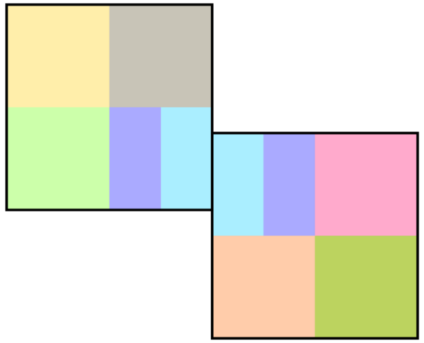
Dynamic load balancing: application to contact mechanics

- Several “stages” in the computation of a time step
  - Assembling of the “mass”/“stiffness”/”forces”
  - Detection of the contacts
  - Assembling of the contact constraints
  - Solving of the saddle point problem
- “Best partition” different for each stage
  - “Compromise” to find
  - Optimization throughout all the stages
- Contact zones may evolve a lot during computation

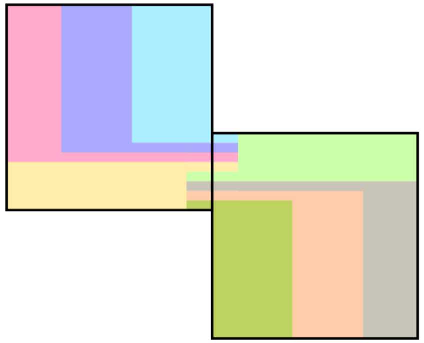
  → “dynamic” load balancing
  - Optimal frequency?
  - Compromise between the cost of the rebalancing, and the cost of unbalanced calculations
- Interaction with other approaches causing dynamic load balancing issues: AMR ?



- Best if no contact
  - Minimal and balanced communications
  - Balanced workloads



- Minimizes communications due to contact
- But unbalanced workload

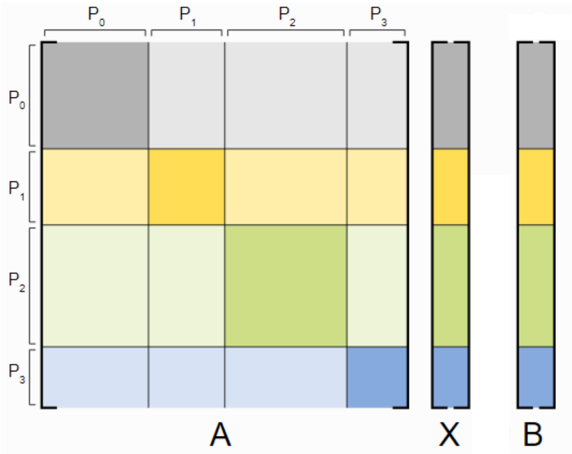


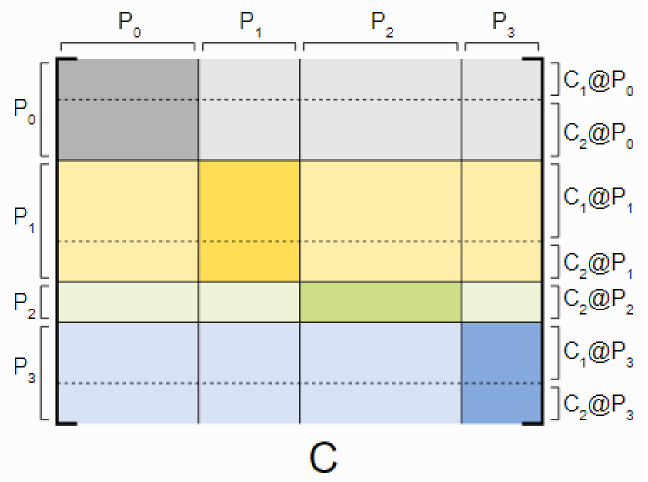
- Balanced workload and communications
- But excess of communications with respect to optimal case

Saddle point problem resolution with iterative solvers for distributed implicit problems

- Open research subject
- A and C very sparse
- C/D enforce complex boundary conditions (such as contact between structures)
  - Different context than the “classical” Stokes-problem
- $size(\lambda) \ll size(X)$
- Matrix free?
- PhD thesis project in collaboration with Sorbonne University starting in 2024

$$\begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} X \\ \lambda \end{bmatrix} = \begin{bmatrix} B \\ D \end{bmatrix}$$





Non conforming Adaptive Mesh refinement

- *A priori* cell-based
  - Forest of structured trees: possibility of specific optimization for structured meshes while keeping the entry points implementations
- Strong impact on load balancing (dynamic)
- Lot of questions:
  - Optimal frequency of the refinement/coarsening ⇔ optimal frequency of the load balancing ?
  - Which numerical methods (conforming, non-conforming) ?
  - Which preconditioners ?
  - …

Performance portability

- At this time
  - MPI only: decomposition of the global mesh into subdomains: each MPI process works out and stores only its subdomain (1 subdomain per MPI process)
    - “Almost (ghosts) Total” distribution of data
  - Vectorization: delegated to Eigen
- Directions for performance portability
  - Hybrid MPI+CPU-threads is not a goal in itself
  - No architecture specific developments
  - Delegation of the performance portability to a programing-model/library/middleware/…
  - First prototype with Kokkos in construction

Code generation

- Compromise
  - Performance
  - Code readability and accessibility
  - Factorization of the code
- Use automatic code generation to win on all fronts
  - Non-c++/parallel-ninja implement “master code” through a DSL
  - Code generator outputs non-factorized and unintelligible but efficient “slave code” implementing MANTA’s pipeline entry points
  - Maintenance occurs only on “master code” (and code generator)
- Automatic differentiation to generate code for Jacobian matrices
- Thesis starting in 2024 to work on that

Thanks for your attention
Some questions?