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

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

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

- **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

- 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

2030: industrial operation
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

C++ “expert” API

C++ “user” API

Numerical method Layer

Core Layer

Python user API

JSON data file

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

Plugins C++

• High level of control and flexibility

- High level of control and flexibiliy
Design constraints for HPC
**Layers**

**End user Layer**
- MFront

**Numerical methods Layer**
- MFront

**Core Layer**
- Castem / gmsh / VTK / MED / PDI

**Services**
- MPI
- Kokkos/SYCL / openMP
- Distributed mesh backend: Moab/pumi/LibMesh
- Distributed linear solver: PETSc/Trilinos/Alien

**Formulations**
- Implicit features
- Explicit features
- Deformable structures
- Compressible fluids
- Thermal conduction

**Fields**
- Spatial integration
- Generic pipeline

**Mesh**
- Distributed memory //
- Shared memory //

**IOs**
- Linear Systems

**Linear Systems**
- Distributed linear solver: PETSc/Trilinos/Alien
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

- **Assembling**: spatial integration over (possibly non-conforming) unstructured meshes
  - Split global integral over mesh entities: \[ M = \sum_i A_i \int_{E_i} m(x) \, dx \]
  - Use finite-element mapping with reference element to integrate using standard quadrature formulae:
    \[ M = \sum_i A_i \sum_j w_j m(\xi_j^i) |\text{det}(\phi_i^j(\xi_j^i))|, \text{ where } (x \in E_i) = \phi_i(\xi) \]
  - Programming of actual problems through entry points:
    - Integrand::addOn \[ \rightarrow \quad w_j m(\xi_j) |\text{det}(\phi_i^j(\xi_j))| \]
    - Assembler::assemble \[ \rightarrow \quad A_i \]

- **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

\[
\begin{bmatrix}
A & C0^t & C1^t & \cdots \\
C0 & 0 & 0 & \cdots \\
C1 & 0 & 0 & \cdots \\
\vdots & \vdots & \vdots & \ddots
\end{bmatrix} \begin{bmatrix}
X \\
\lambda_0 \\
\lambda_1 \\
\vdots
\end{bmatrix} = \begin{bmatrix}
B \\
D0 \\
D1 \\
\vdots
\end{bmatrix}
\]
“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
A few illustrations
Some tools

Languages & compilers

HPC benchmark

Collaborative workflow
3. Roadmap & some directions for HPC
Geometrical intersections detection with distributed parallelism

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
- \( \text{size}(\lambda) \ll \text{size}(X) \)
- 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 $\leftrightarrow$ 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?