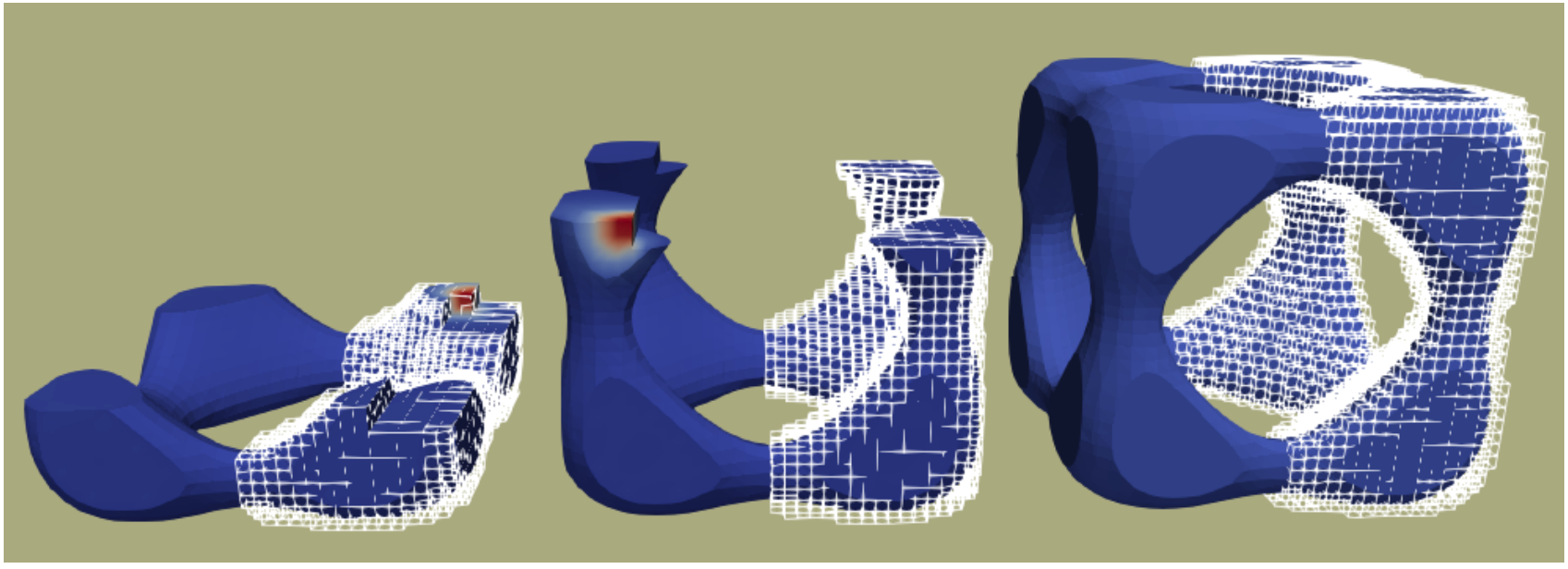
performance in Julia

Santiago Badia, F. Verdugo



MWNDEA

Monash University, February 14th 2020

MONASH University

Numerical PDE solvers

PDEs: multiphysics, multiscale, constitutive models

Discretisation: (grad, curl, div)-conforming, dG, unfitted FEM, hybrid and virtual elements, h/p-adaptivity

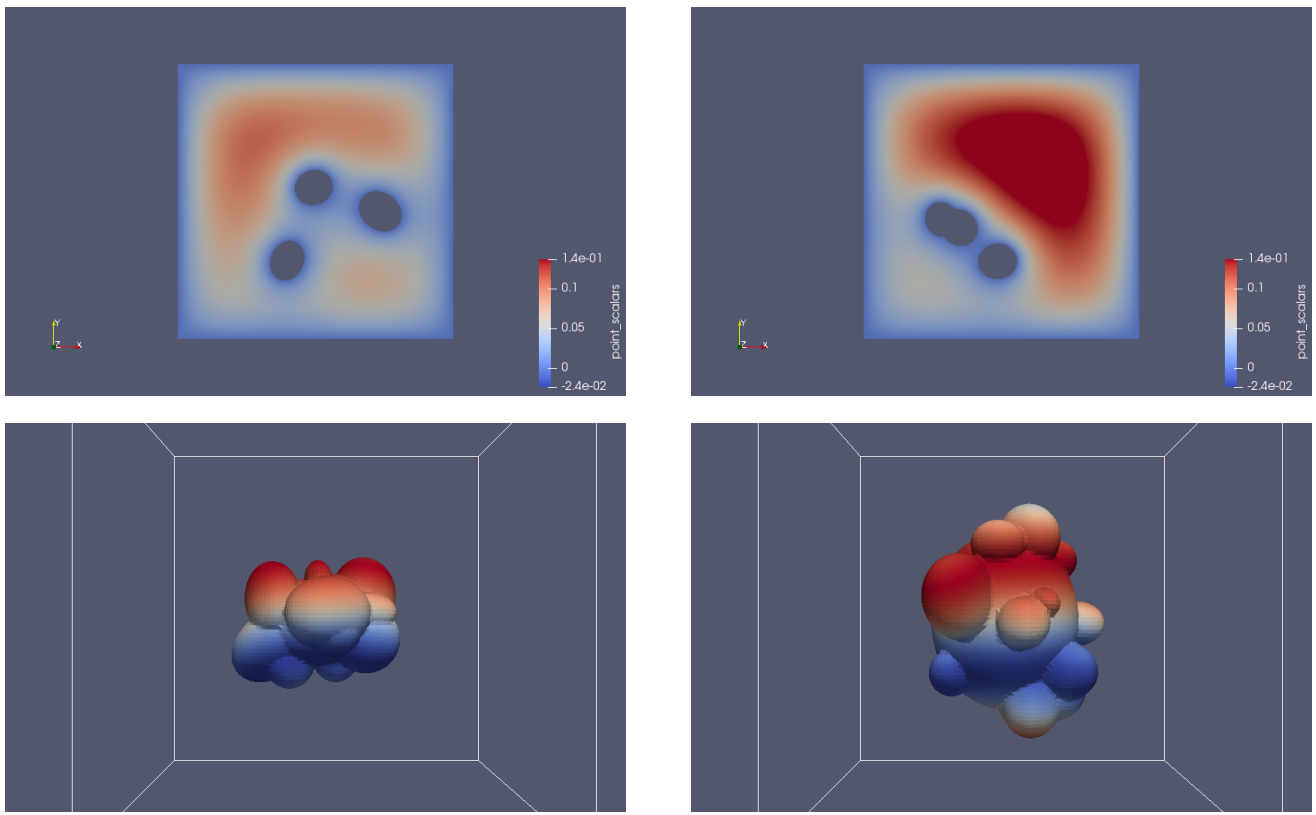
(Non)linear solvers: multiscale/multilevel solvers strongly coupled to PDE structure (not black-box)

Large scale computations: distributed-memory implementations, accelerators, ...

Other ingredients: forward/inverse UQ, data-driven parameter identification, ...

Numerical PDE solvers

E.g., forward UQ on random domains (unfitted, MLMC, multilevel solvers, (inter, intra)-sample parallelism)



SB, J. Hampton, J. Principe. *Embedded multilevel Monte Carlo for uncertainty quantification in random domains*, arXiv:1911.11965 (2020)

Computational math labs

Scientific software + computers are **our labs**

Advanced algorithms and complex problems require **advanced software engineering** (extensibility, readability)

Complex problems require **high-performance** codes (parallel, optimised)

Open source scientific software for reproducible science

Scientific computing teams

Excellent pool of high-performance libraries:

deal.ii, Fenics, FEMPAR, MOOSE, libmesh, Firedrake, DUNE, etc.

- C++ or OO FORTRAN08 (static/compiled languages), some w/ Python interfaces (dynamic language)
- Excellent if they provide all you need (*user*)
- Far more involved if not (*library developer*)

Scientific computing teams

PhD students (3-4y), postdocs (1-3y)

No computer scientists

Software dev policies

Reuse: New algorithms may involve extensions of the PDE library core (*developers*), getting started *very time-consuming* (productivity loss)

Start from scratch: Academic MATLAB, Python codes (waste previous work, hard to reach state of the art)

Productivity vs performance

Dilemma: productivity vs performance

Productivity

Related to **dynamic languages** (Python, MATLAB...): More expressive, no compilation step, interactive development (debugging on-the-fly), better for math-related bugs (no benefit from static compilation), no set-up of environment (compilers, system libraries, etc)

Performance

Related to **static languages** (C/C++,FORTRAN,...): Compilers generate highly optimised code

Compromise solutions

Dynamic-static combinations: vectorised PDE solvers in Python + external pre-compiled libraries (NumPy in C); high-level Python interface of a static PDE library (Fenics, C++), etc.

- Constraints over the dynamic code (e.g. vectorisation)
- Two-language barrier: When changes require to get into static library

All-in-one

Productive: Dynamic language (as Python, MATLAB...)

Performant: Advanced type-inference system + just-in-time (JIT) compilation

21st century FORTRAN, designed for numerical computation (MIT, 2011-)

Solve previous issues: for-loops not a problem, *everything* can be written in Julia (see e.g. Flux for ML, 100% Julia-code)

Julia features

- **Not OO:** No inheritance of concrete types (only abstract types), *use composition, not inheritance*, *classify by their actions, not their attributes*...
- **Multiple dispatching paradigm:** functions not bound to types, dispatching wrt all arguments (solving multiple inheritance)
- **Performant Julia code is not obvious:** help JIT compiler to infer types, *type-stability* to allow JIT compiler create performant code

Julia universe

- Package manager is awesome
- Every project comes with its list of dependencies (automatic process)
- Seamless integration with ‘Github‘ (register packages, automatically-generated code documentation)
- Unit testing and performance tools...

Gridap

Gridap seed started in Christmas 2018... trying to increase productivity in my team

Not good PDE discretisation libraries in Julia

Some key decisions based on previous experience:

- Functional-like style i.e. **immutable objects**, no *state diagram* (just cache arrays for performance)
- **Lazy evaluation** of expressions (implement unary/binary expression trees for types)

CellFields

CellField is a key Gridap type

CellField

Given a cell in a partition $\mathcal{T}$ of a manifold $\mathcal{M}$ (e.g. cells, faces, edges in a mesh), it provides a Field. A Field assings a physical quantity (n-tensor) per space(-time) point in the manifold.

Given an array of points per cell in $\mathcal{T}$, we can evaluate a CellField, returning an array of scalars/vectors/tensors per cell per point

We also implement CellField operations:

- Unary operations:
- Binary operations: inner$(,)$, $\times$, etc.

With these types, we represent FE functions, FE bases, constitutive models, etc.

Applying these CellFields to integration points in a numerical quadrature plus operations we can integrate forms and assemble matrices

Let us look at Gridap Tutorial 1

Some comments about the code

- Nesting objects into other objects via composition (mesh in FE space in FE function + bilinear form (duck typing) + triangulation + quadrature in FE operator...). All objects are immutable
- No numerical computations at FEOperator declaration yet, just creating the expression tree ($\nabla()$ and ‘inner‘)
- Numerically intensive computations deployed in solve

Quite complicated PDE

Just some lines of code (laws, residuals, and Jacobians)

Gridap status

Gridap is pretty comprehensive (big thanks to F Verdugo's amazing work at UPC):

- Lagrangian, Raviart-Thomas, Nedelec, dG
- Multifield or multiphysics methods
- Interaction with GMesh, Pardiso, PETSc...
- dimension-agnostic (5-dim Laplacian), order-agnostic

Quite rich documentation, tutorials, automatic testing, etc.

After 1 year and two developers (not full time!)... highly productive environment

Gridap for teaching

Objective: same software for research and teaching

- FE tutorials in *MTH5321 - Methods of computational mathematics*
- One undergrad AMSI project on *Gridap*: from no idea about FEs/coding to MRI data of velocity of patient-specific aorta to pressure fields in 2 months

Gridap future

This is just the beginning:

- Distributed-memory integration/assembly (w/ A Martín)
- hp-adaptivity (w/ A Martín)
- Historic variables for nonlinear constitutive models
- Virtual element methods
- Space-time discretisations
- Interaction with optimisation, ML, UQ, ODE, automatic diff packages

Gridap future

Performance/productivity analysis:

- Running FE problems with $O(10^6)$ cells in my laptop in $O(10)$ sec (similar to FEMPAR), *performance analysis* on the way (x2-3 drop in performance OK if x2-3 productivity)