Advances in Feel++ : A Domain Specific Embedded Language in C++ for Partial Differential Equations

A. Samaké*
V. Chabannes*
C. Daversin†
V. Doyeux‡
M. Ismail‡
G. Pena§
C. Prud'homme¶
C. Trophime†
S. Veys*

ECCOMAS 2012

Vienna, September 12, 2012

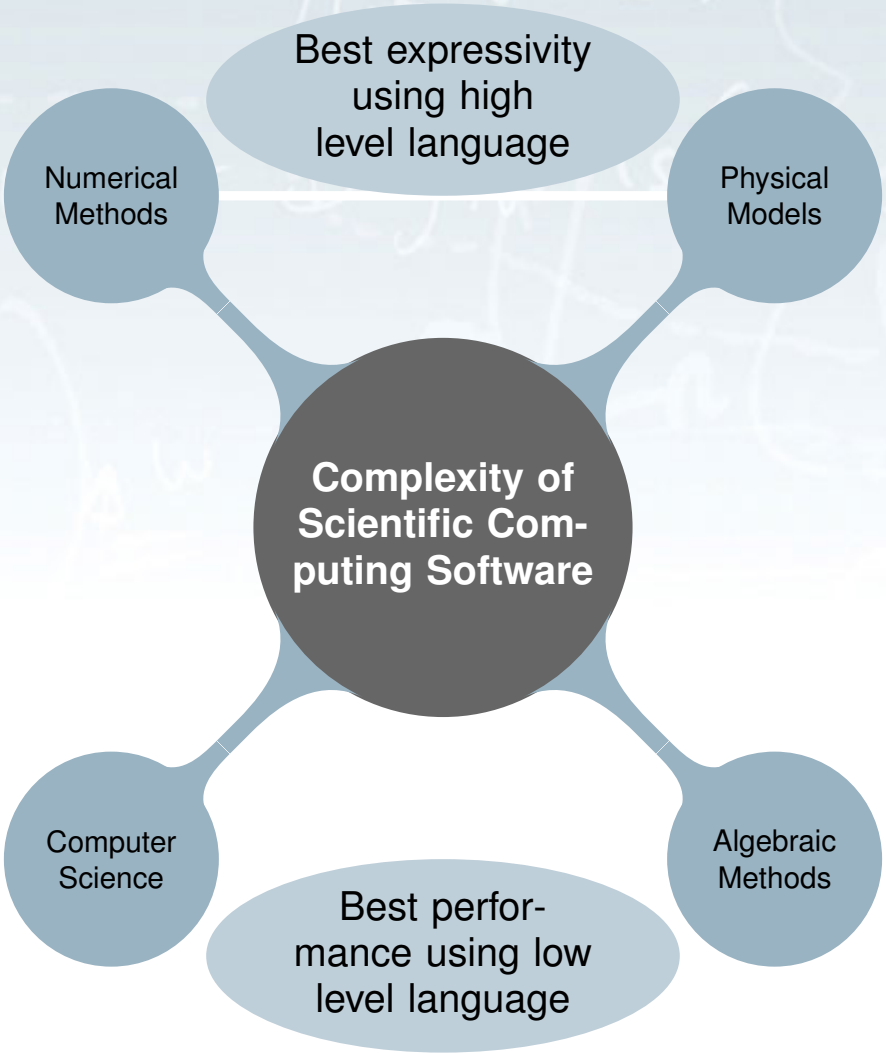
*Université Grenoble 1 / CNRS, LJK
†LNCMI-G,CNRS-UJF-UPS-INSA
‡Université

FEEL++ Collaborators

S. Bertoluzza (IMATI/CNR/Pavia)
V. Chabannes (UJF/LJK)
R. Chakir (UPMC/LJLL)
C. Daversin (CNRS/LNCMI)
V. Doyeux (UJF/LIPHY)
J.M. Gratien (IFPEN)
M. Ismail (UJF/LIPHY)
P. Jolivet (UPMC/LJLL)
F. Nataf (UPMC/LJLL)
G. Pena (UC/CMUC)
D. Di Pietro (IFPEN)
A. Samake (UJF/LJK)
M. Szopos (UDS/IRMA)
R. Tarabay (UDS/IRMA)
C. Trophime (CNRS/LNCMI)
S. Veys (UJF/LJK)
+ Master students

Outline

Generative Programming and DS(E)L

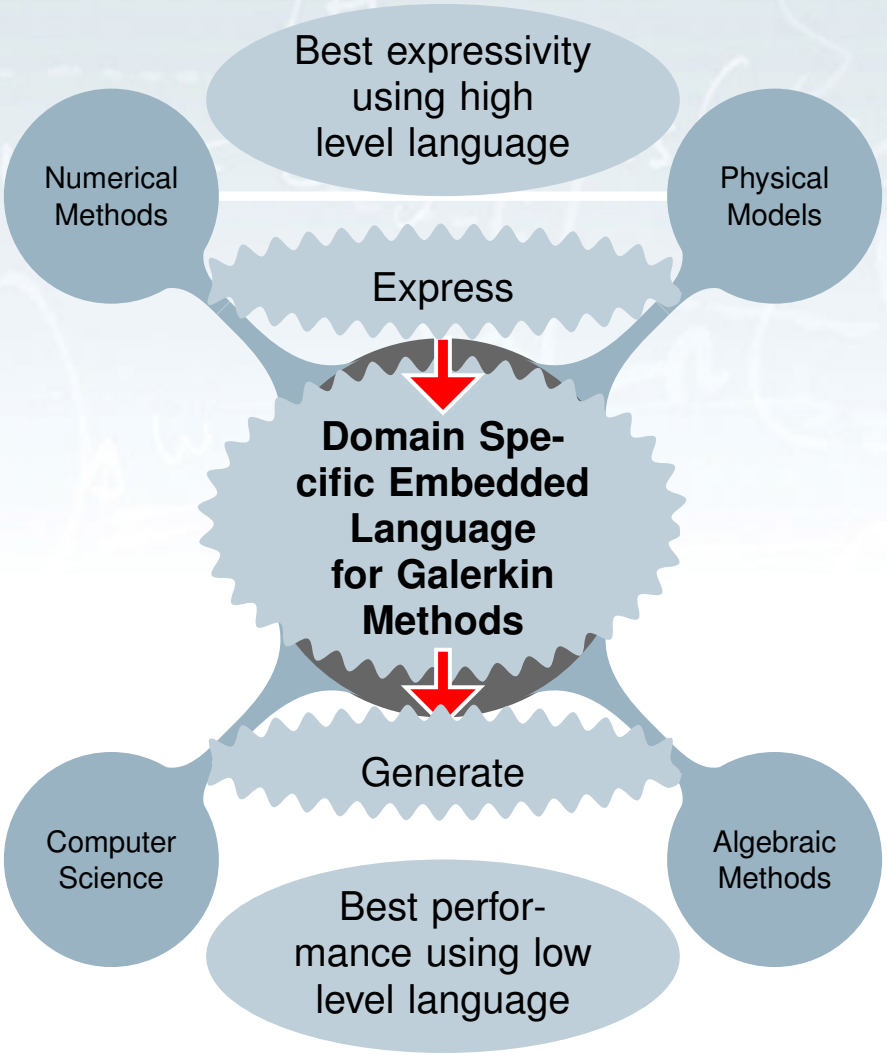


Complexity Types

- Algebraic
- Numerical
- Models
- Computer science

- Numerical and model complexity are better treated by a high level language
- Algebraic and computer science complexity perform often better with low level languages

Generative Programming and DS(E)L



Generative paradigm

- distribute/partition complexity
- developer: The computer science and algebraic complexity
- user(s): The numerical and model complexity

FEEL++: http://www.feelpp.org

Finite Element Embedded Library in C++ : A DS(E)L in C++ for PDEs

Features

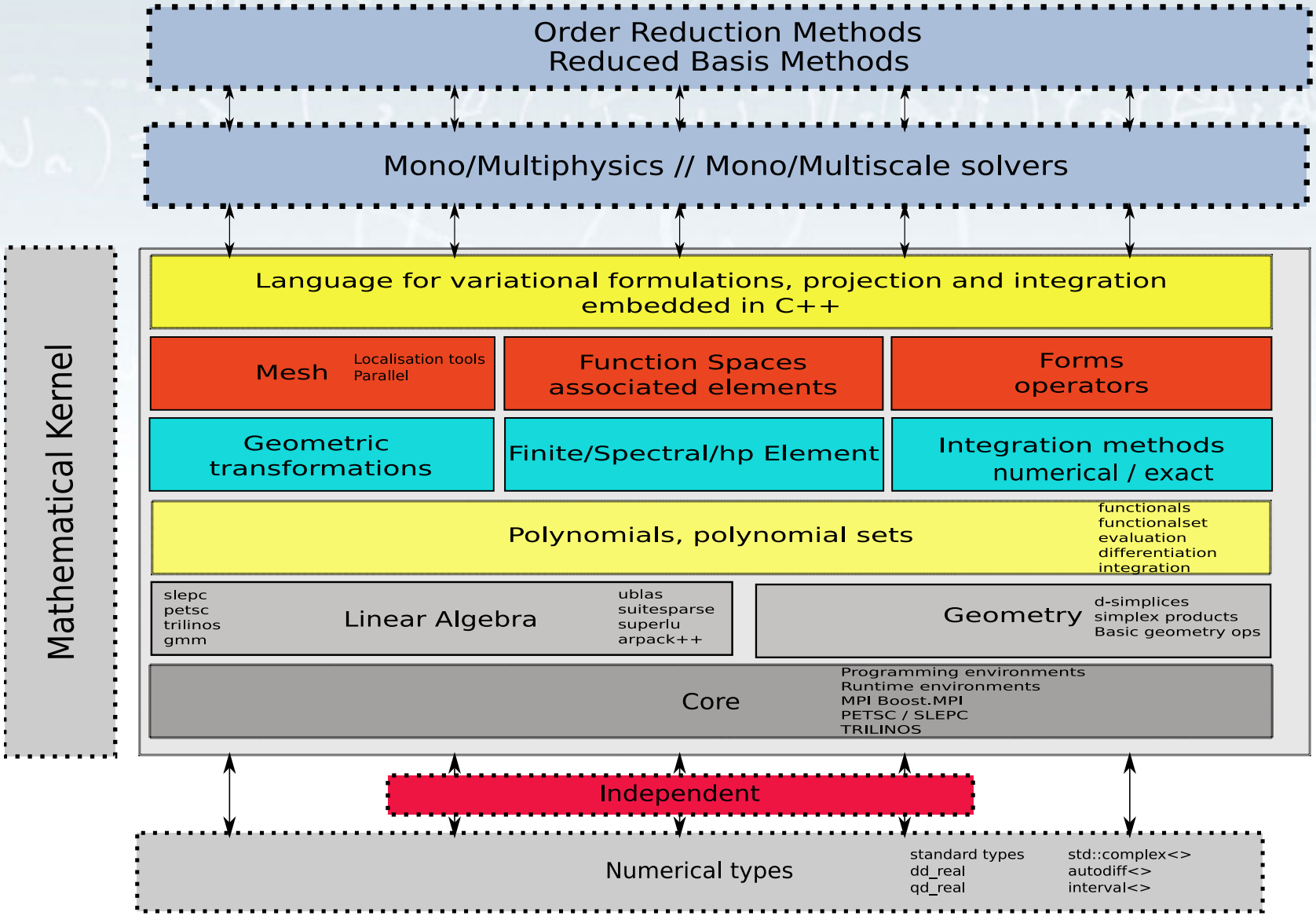
- Generalized Galerkin (cG, dG) methods in 1D, 2D and 3D
- Support for simplices, tensor products and high order ALE
- Support for various polynomial sets (modal, nodal) of arbitrary order (>=0)
- Support for parallel computating
- (Non-)Linear algebra using PETSc/SLEPc and Trilinos
- Domain specific language embedded in C++ for variational formulations
- Operators, function spaces, elements of function spaces (also parallel) ...
- A computational framework that maps closely the mathematical one
- A modern C++ library: use Boost library and C++11 as much as possible

Available in Debian/Ubuntu

FEEL++: http://www.feelpp.org

- Université de Grenoble: LIPHY and LJK
- Dept. of Mathematics, U. Coimbra
- CNRS: LNCMI
- IFPEN
- CNR: IMATI

Architecture



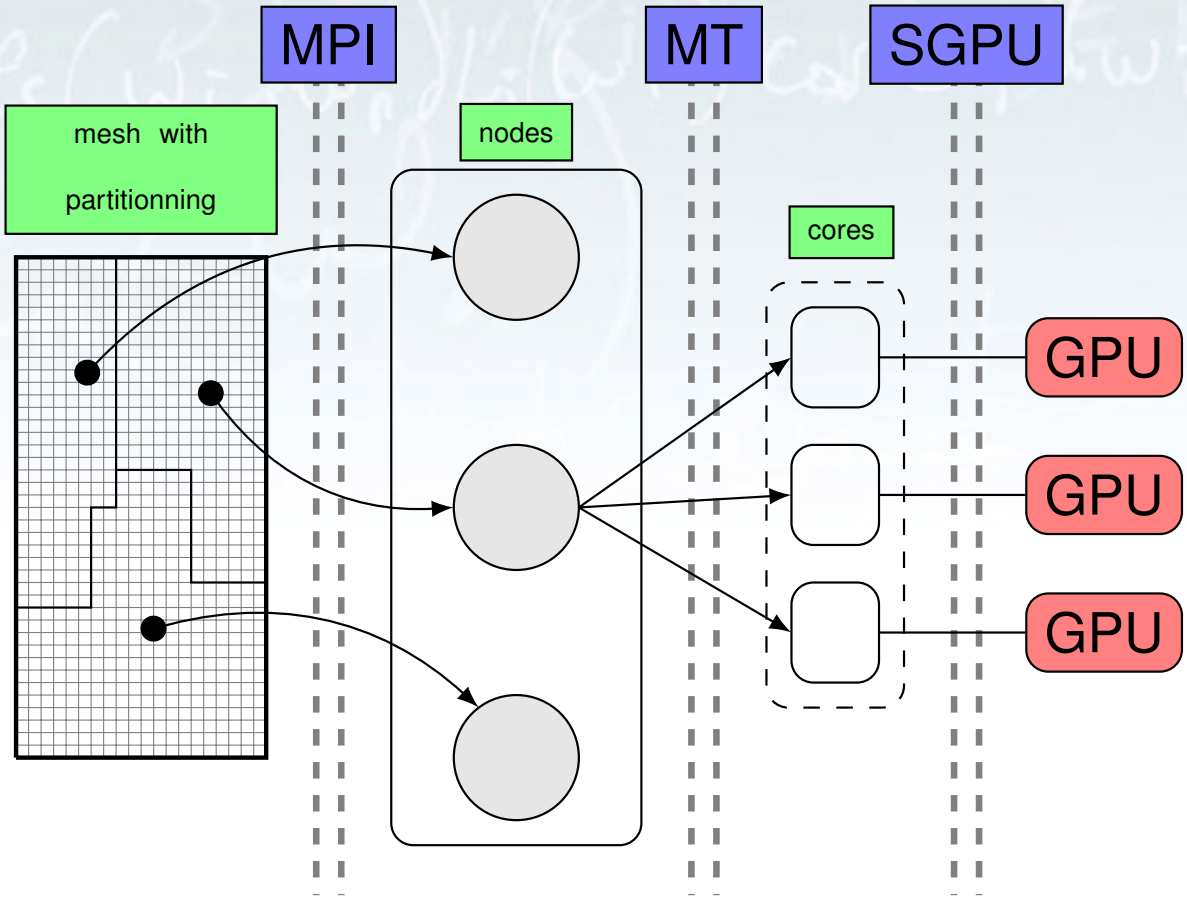
Seamless parallelisation

Hybrid architectures

- many nodes, many cores, hybrid nodes
- MPI, multi-threads, GPU

MPI implementation :

- mesh partitioning
- dof table partitioning
- PETSc interface



- The parallelism is completely transparent (implicit use)
- It can also make explicit (control communications)

Mesh(Parallel)

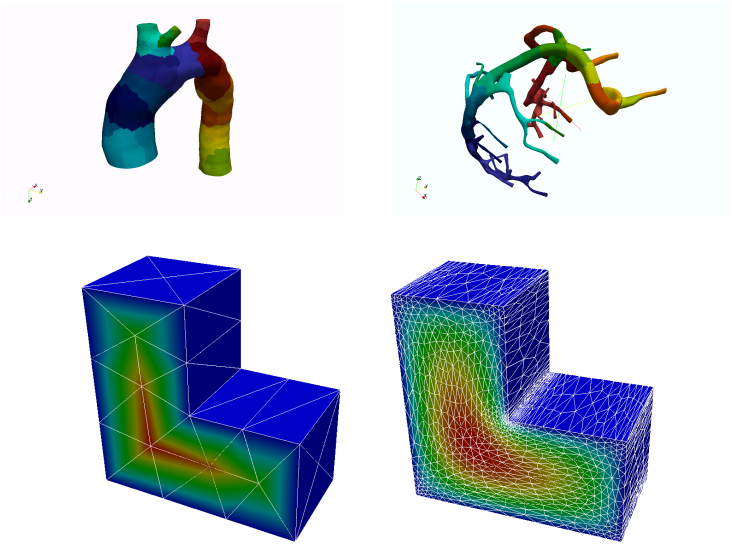


Figure 2.1: Mesh adaptation on 3D L-shape

- Convexes and associated geometric transformation ($\mathbb{P}_N, N = 1, 2, 3, 4, 5...$)
- Support for high order ALE maps
- Geometric entitites are stored using Boost.MultiIndex
- Element-wise partitioning using Scotch/Metis, sorting over process id key
- Mesh adaptation (isotropic versus anisotropic)

Example

```
elements(mesh [, processid]);
markedfaces(mesh, marker [, processid]);
markededges(mesh, marker [, processid]);
```

Function Spaces(Parallel)

- Product of N-function spaces (a mix of scalar,vectorial, matricial,different basis types, different mesh types, conforming and non-conforming)
- Use C++ Variadic templates
- Use Boost.MPL and Boost.Fusion heavily
- Get each function space and associated “component” spaces
- Associated elements/functions of N products and associated components, can use backend (petsc/slepc)
- Support periodic and non-periodic spaces

Example

```
typedef FunctionSpace<Mesh,bases<Lagrange<2,Vectorial>,
                                 Lagrange<1,Scalar> > > space_t;
space_t Xh( mesh );
auto Uh = Xh.functionSpace<0>();
auto x = Xh.element();
auto p = x.element<1>(); // view
```

The dof table partitioning

Goal : create the local and global dof tables

local representation :

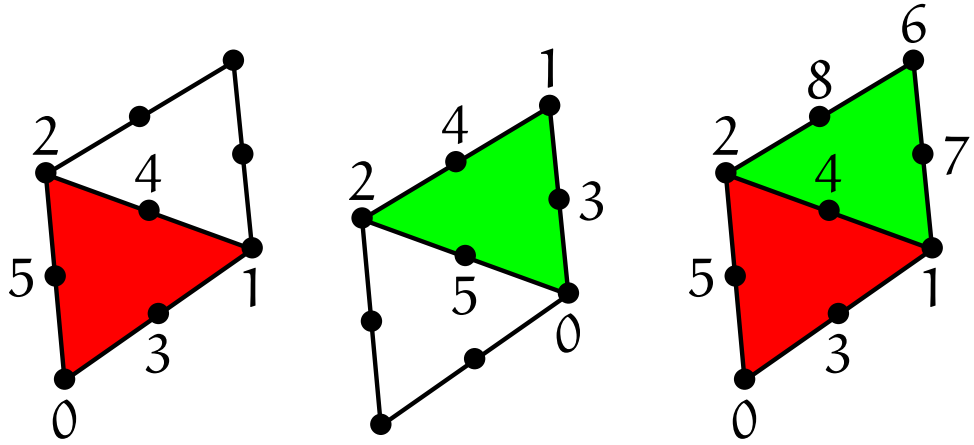
proc0	0	1	2	3	4	5
proc1	0	1	2	3	4	5

global representation :

proc0	0	1	2	3	4	5
proc1		6		7	8	

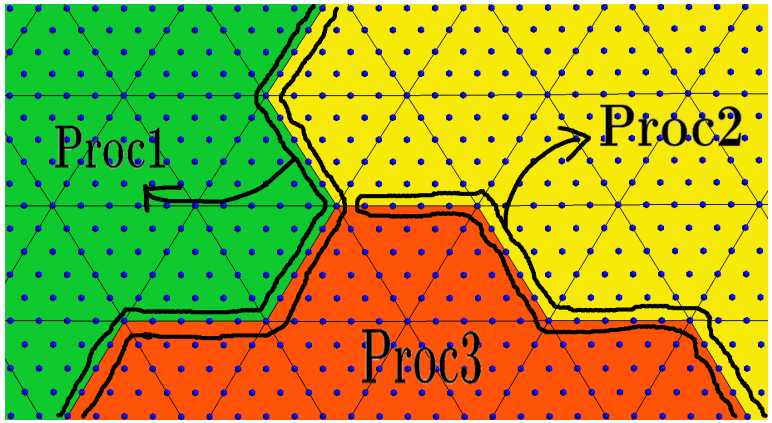
localToGlobal mapping :

proc0	0	1	2	3	4	5
proc1	1	6	2	7	8	4



(a) proc0 (b) proc1 (c) global

- Step 1 : build the local dof table with ghost dofs
- Step 2 : build the global dof table without ghost dofs
  - the single global dof belongs to the process of smallest rank
  - communication : update the id of interprocess dofs on the global table



Operators and Forms (Parallel)

- Linear Operators/ Bilinear Forms represented by full, blockwise matrices/vectors
  - Full matrix $\begin{pmatrix} A & B^T \\ B & C \end{pmatrix}$, Matrix Blocks A, B^T, B, C
- The link between the variational expression and the algebraic representation

Example

```
X1 Xh; X2 Vh;
auto u = Xh.element(); auto v = Vh.element();
// operator T : X1 → X2
auto T = LO( Xh, Vh [, backend] );
T = integrate(elements(mesh), id(u)*idt(v) );
// linear functional f : X2 → ℝ
auto f = LF( Vh [, backend] );
T.apply( u, f ); f.apply( v );
```

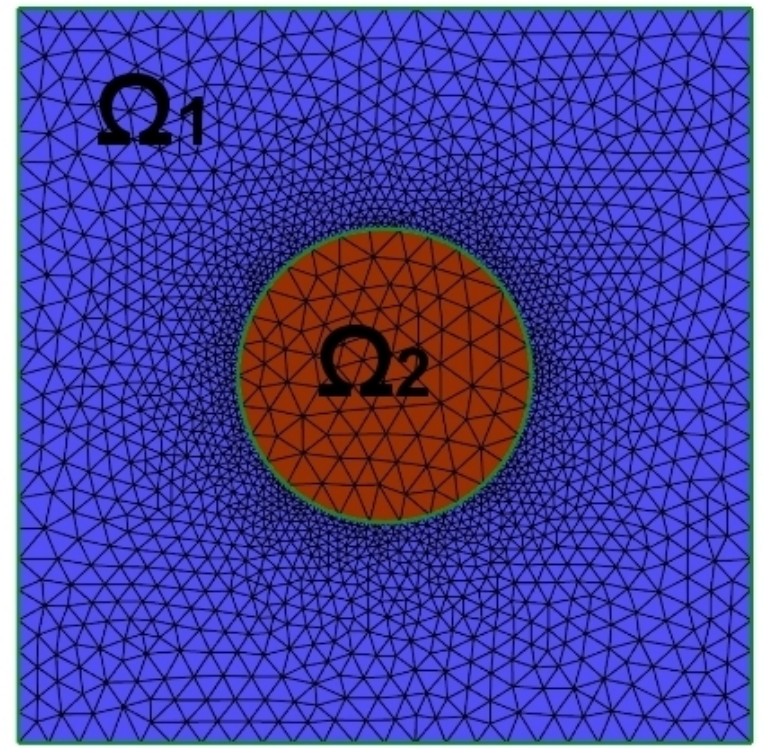
Seamless Interpolation Tool(Parallel)

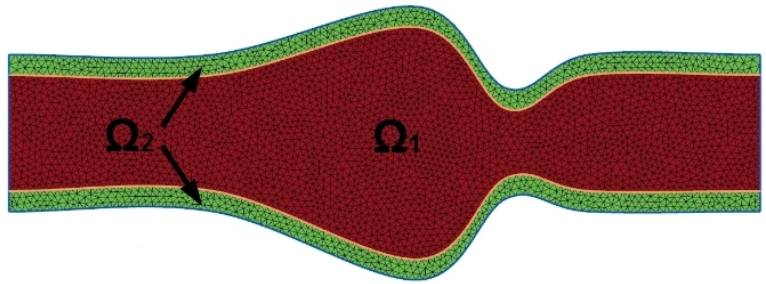
Motivations

- Interpolation between different meshes (h) or function spaces (N)
- $\forall \mathrm{d} = 1, 2, 3, \forall \mathrm{N}, \forall \mathrm{N}_{\mathrm{geo}}$ at dof or quadrature nodes
- Computation of different operations (id, ∇, $\nabla\cdot$, $\nabla\times$,...)
- $\mathrm{I}_{\mathrm{h}}^{\mathrm{LAG}}, \mathrm{I}_{\mathrm{h}}^{\mathrm{CR}}, \mathrm{I}_{\mathrm{h}}^{\mathrm{RT}}, \mathrm{I}_{\mathrm{h}}^{\mathrm{Her}}, \ldots$

Some Applications

- Multiphysics coupling (e.g. FSI)
- Fictitious domain meth. (e.g. FBM)
- Domain decomposition meth. (e.g. Schwartz)





FEEL++: First Strategy for Domain Decomposition

Implicit

- Use PETSc parallel (implicit communications)
- Automatic mesh partitioning using gmsh (Scotch/Metis)
- FEEL++ data structures are parallel (e.g. Function Spaces...)
- Use parallel PETSc solvers and (sub-)preconditioners
  - Krylov SubSpace methods(KSP) and Direct solvers(MUMPS)
  - Preconditioners : Block-Jacobi, ASM, GASM ...

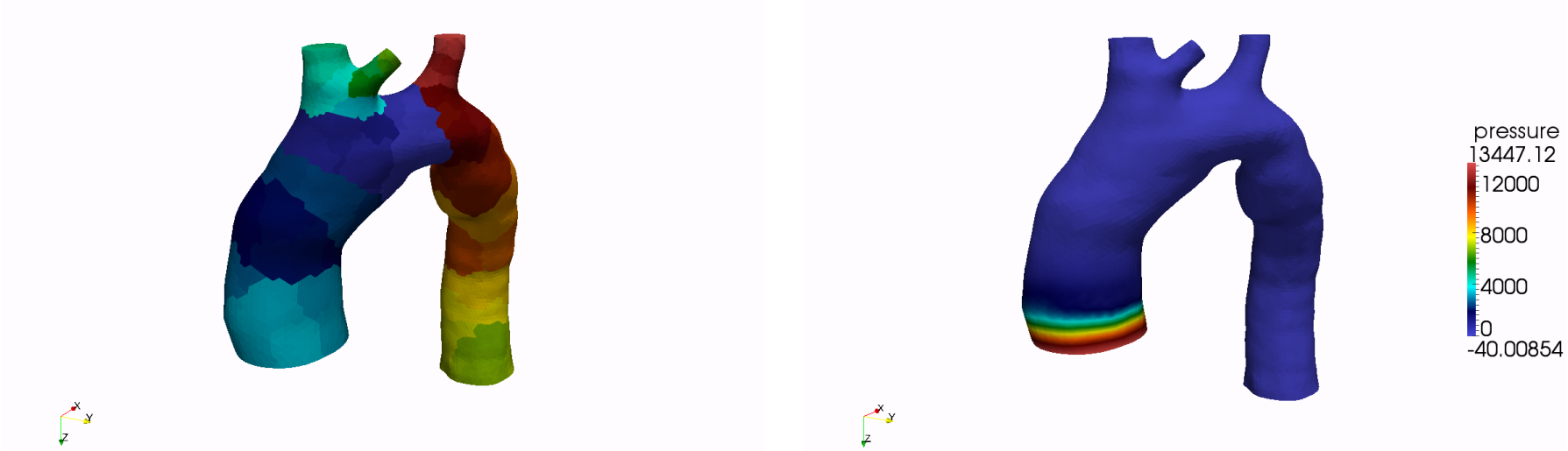
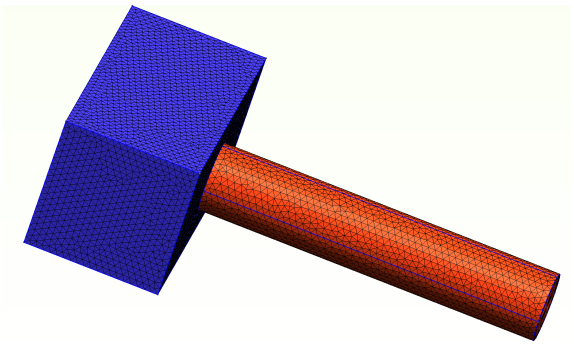


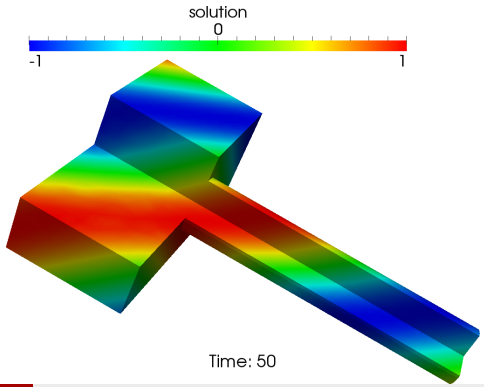
Figure 3.1: Mesh partitioning using gmsh

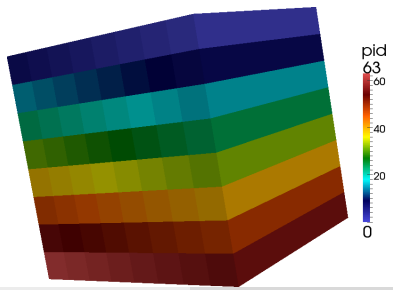
FEEL++: Second Strategy for Domain Decomposition

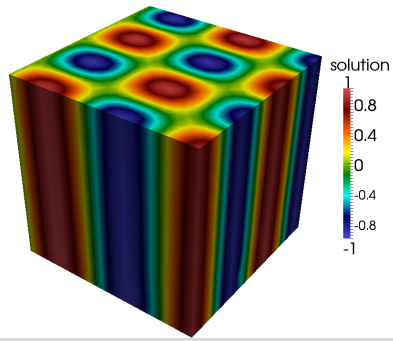
Explicit

- Make PETSc sequential even though the code is parallel (mpi communicators)
- Send and receive complex data structure using Boost.MPI and Boost.Serialization
  - mesh data structures
  - elements of functions space (traces)
- Define Two different communicators
  - a global one for communication between subdomains
  - a local one (sub-communicator) that activates only the
  - FEEL++ and PETSc see only the local one and thinks the computations are sequential
- Operator interpolation (already available in sequential)





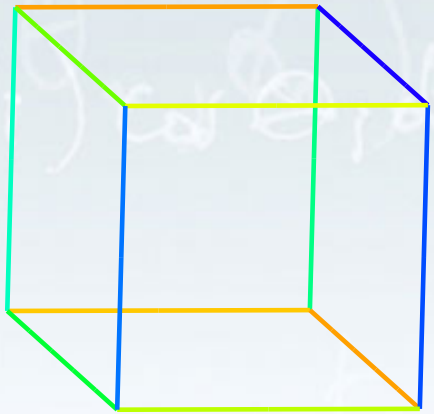




Tools for Substructuring Preconditioners (e.g mortar)

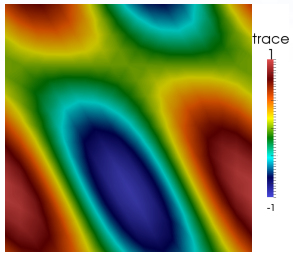
Listing 1: Mass matrix on wirebasket

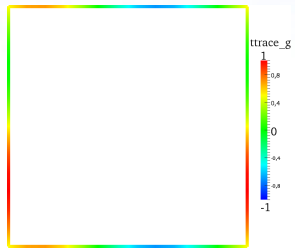
```
auto Xh = space_type::New( _mesh=mesh );
auto wirebasket = createSubmesh( mesh, markededges(mesh,"WireBasket") );
auto Wh = trace_trace_space_type::New( _mesh=wirebasket );
auto w = Wh->element();
auto z = Wh->element();
auto M = M_backend->newMatrix( _test=Wh, _trial=Wh );
form2( _trial=Wh, _test=Wh, _matrix=M ) =
    integrate(_range=elements(wirebasket), _expr=idt(w)*id(z) );
```

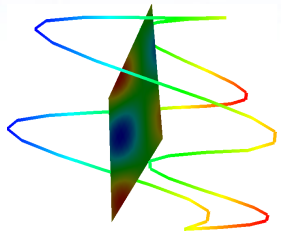


Listing 2: Jump matrix on interfaces

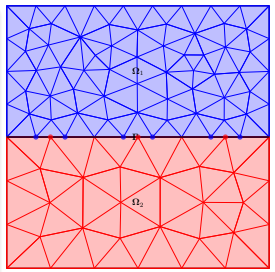
```
auto Xh = space_type::New( _mesh=mesh );
lag_mesh = mesh->trace( markedfaces( mesh, "marker" ) );
auto Lh = trace_space_type::New( lag_mesh );
auto u = Xh->element();
auto mu = Lh->element();
auto B = M_backend->newMatrix( _trial=Xh, _test=Lh );
form2( _trial=Xh, _test=Lh, _matrix=B ) =
    integrate( elements(lag_mesh), idt(u)*id(mu) );
```



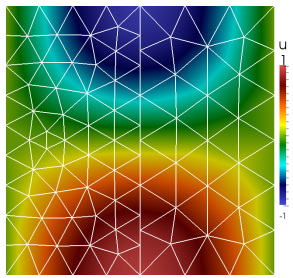




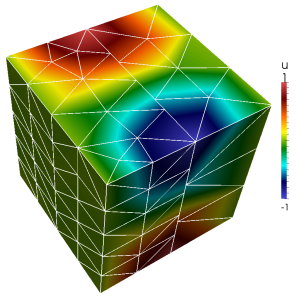
- Extract trace mesh (`TDim=RDim-1`)
- Extract trace of trace mesh (`TDim=RDim-2`)
- Assembly mass matrix on wirebasket
- Assembly stiffness matrices on all faces
- Assembly jump matrices on all non-mortar sides
- Operators trace and lift (hamonic/by constant)



(a) mesh 2D



(b) mortar in 2D



(c) mortar in 3D

FEEL++ FSI framework

- Strategy :
  - Partionned method
  - Implicit and Semi-implicit schemes
  - Fixed point with Aitken relaxation
- Models developped :
  - Fluid model : incompressible Navier-Stokes with ALE framework

$$\rho_f \left.\frac{\partial \boldsymbol{u}_f}{\partial t}\right|_{x^*} + \rho_f \left(\boldsymbol{u}_f - \boldsymbol{w}_f \cdot \nabla\right) \boldsymbol{u}_f - \nabla \cdot \boldsymbol{\sigma}_f = \boldsymbol{f}_f$$

$$\nabla \cdot \boldsymbol{u}_f = 0$$

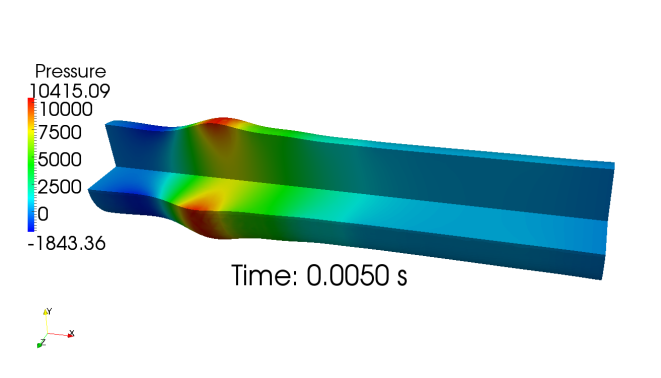
    with $\boldsymbol{w}_f$ the mesh velocity, $\mathcal{A}_t$ ALE map and $x = \mathcal{A}_t(x^*)$
  - Structure model : hyper-elastic and compressible with Lagragian framework

$$\rho_s \frac{\partial^2 \boldsymbol{\eta}_s}{\partial t^2} - \nabla \cdot (\boldsymbol{F}_s \boldsymbol{\Sigma}_s) = \boldsymbol{f}_s, \quad \boldsymbol{\Sigma}_s = \lambda_s \left(\mathrm{tr}\boldsymbol{E}_s\right) \boldsymbol{I} + 2\mu_s \boldsymbol{E}_s.$$

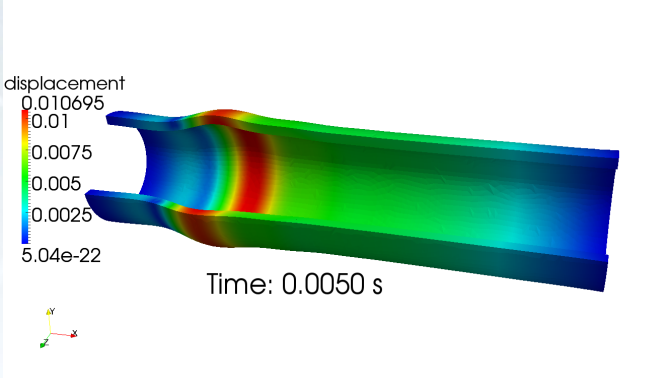
  - FSI coupling conditions : $\boldsymbol{u}_f = \boldsymbol{\eta}_s$ and $\boldsymbol{\sigma}_f \vec{n}_f + J_{\mathcal{A}} \boldsymbol{F}_s \boldsymbol{\Sigma}_s \boldsymbol{F}_{\mathcal{A}} \vec{n}_s = 0$

FSI applications : Pressure pulse propagating in blood flow

- Geometry order 2 :



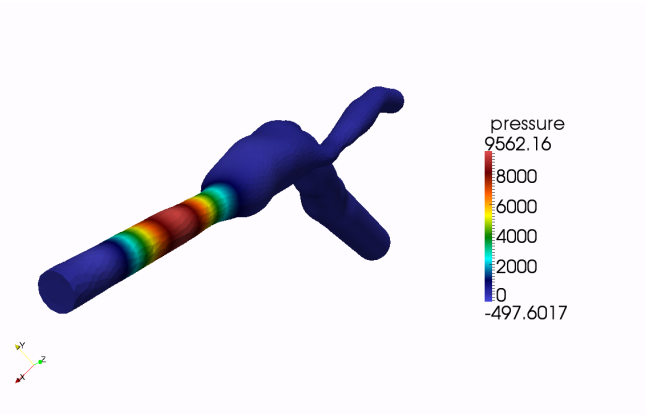
(d) Fluid pressure(disp magnified 10 times)



(e) Structure displacement(disp magnified 10 times)

- Realistic meshes :

(f) Fluid pressure in aorta (216proc)



(g) Fluid pressure in artery(108proc)

FSI applications : FBM

Goal : take into account elastic particles in a fluid flow

- Fluid solver use fictitious method domain: FBM
- FBM principle : Transform the original problem into several sub problem :
  - one on the global mesh
  - several on the local domain(around the perforations)

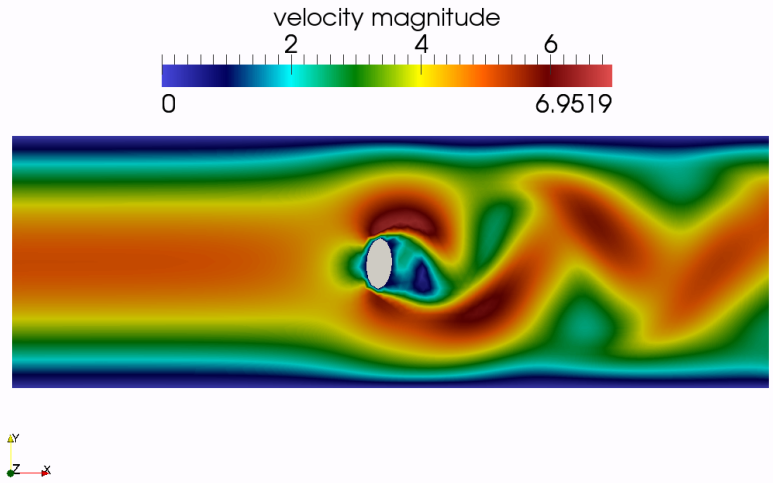


Figure 4.1: Particle displacement in a flow (parabolic inlet)

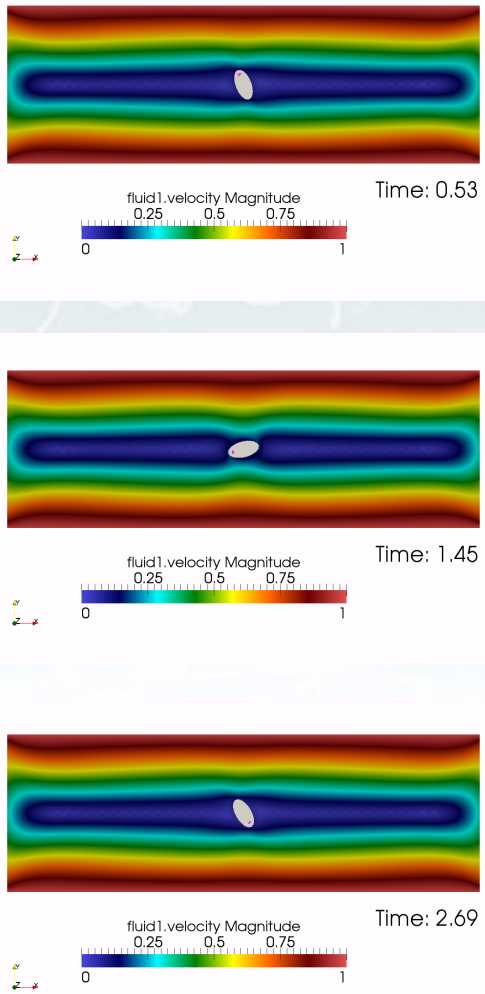
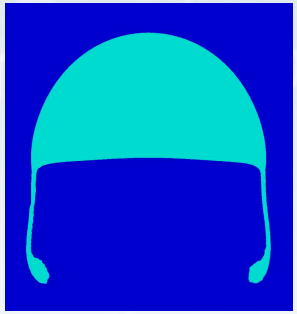


Figure 4.2: Particle displacement in a shear flow

Another strategy : FSI with level set methods

- Goal : simulate the behavior of inextensible membranes in a fluid flow
- Strategy :
  - interface between fluids captured by a level set function ϕ
  - interfacial forces are projected on the region where $|\phi| < \varepsilon$
  - Lagrange multiplier used to impose inextensibility



- Model :
  - Fluid equations : Navier Stokes with $\rho(\phi), \mu(\phi), \boldsymbol{f}(\phi)$
  - Level Set advected by fluid velocity

$$\frac{\partial \phi}{\partial t} + \boldsymbol{u} \cdot \nabla \phi = 0$$

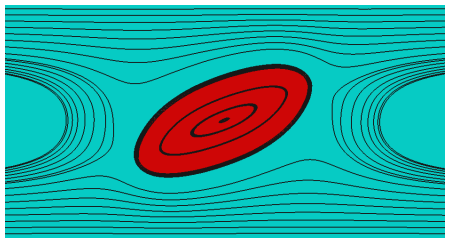
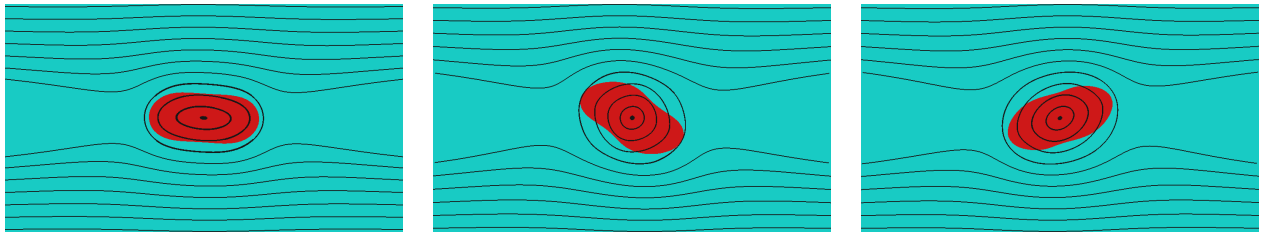


Figure 4.3: Tank treading $\mu_{in} = \mu_{out}$



(a) t=100 (b) t=110 (c) t=600

Figure 4.4: Tumbling motion $\mu_{in} >> \mu_{out}$

Usage context for reduced basis methods

Parametrized PDE

- Input-parameter examples : geometric configuration, physical properties, boundary conditions, sources.
- Output examples : mean temperature over a subdomain, flux on a boundary, etc.

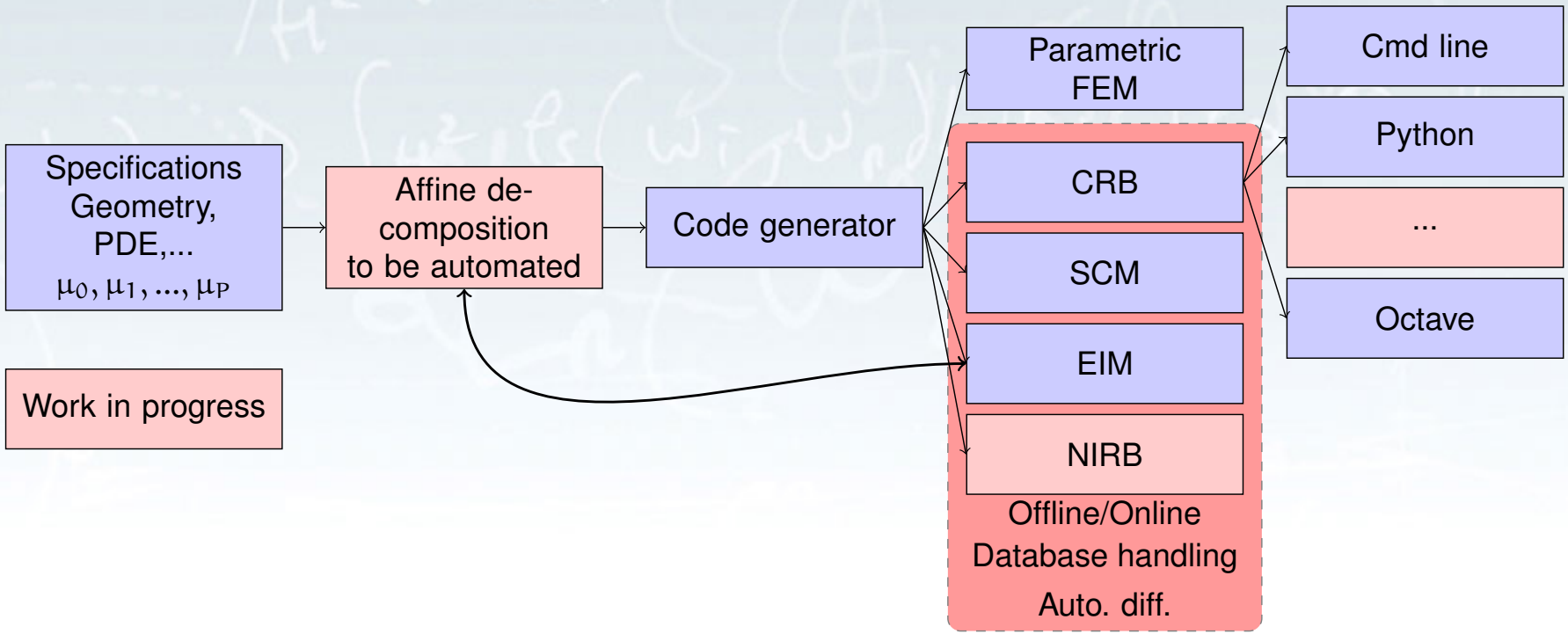
Motivation : rapid and reliable evaluation of input-output relationships

- Real-time : parameter-estimation, control...
- Many-query : sensitivity analysis, optimization....

Objectives

- Develop a generic framework
- Application to industrial problems (when possible, e.g. (non-)linear multiphysics problems)

Reduced basis framework



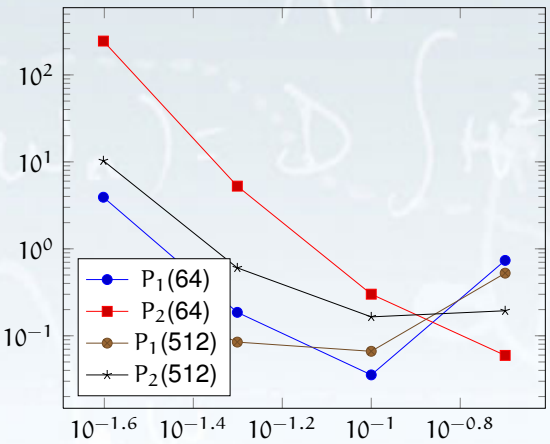
Sensitivity analysis

OpenTURNS[‖] can be used and is interfaced with python scripts.

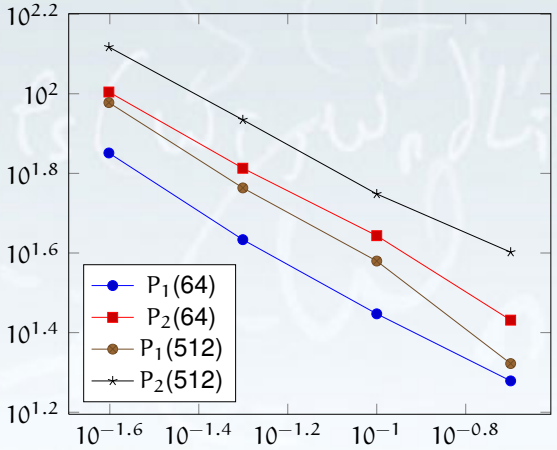
[‖] http://www.openturns.org

PETSC/gasm Solver: Time and Iterations

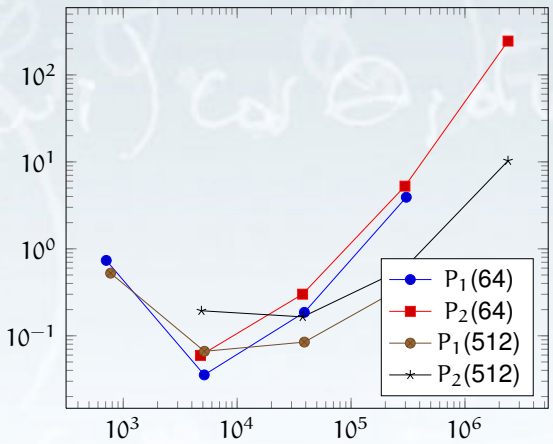
Skip



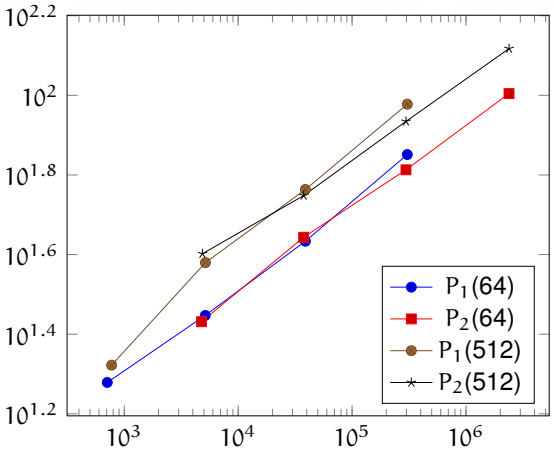
(a) solver time vs h



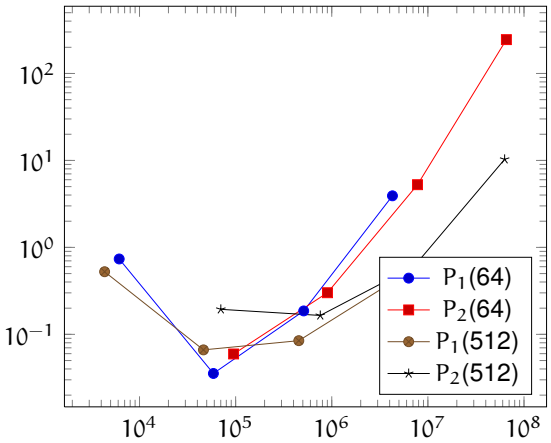
(b) solver iterations vs h



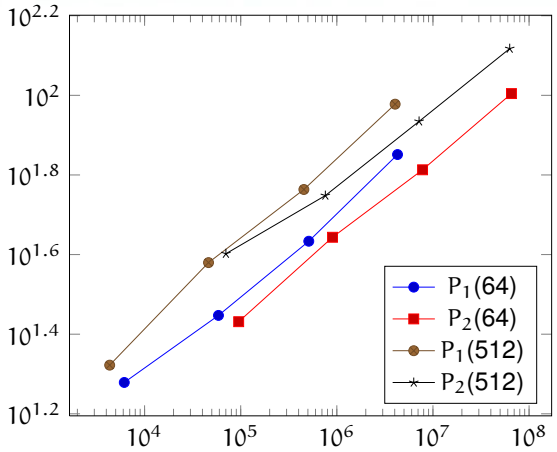
(c) solver time vs ndof



(d) solver iterations vs ndof



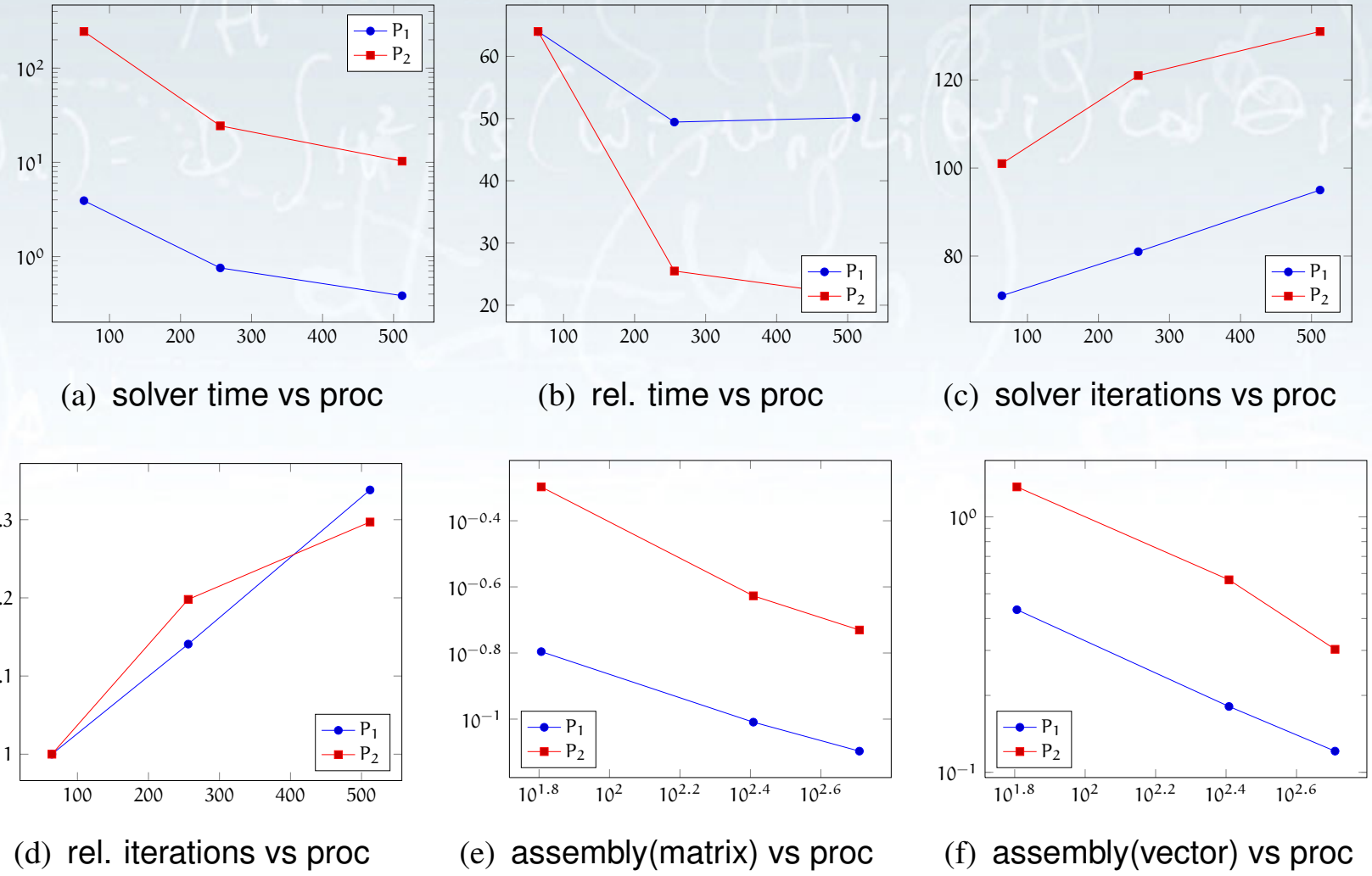
(e) solver time vs nnz



(f) solver iterations vs nnz

Figure 6.1: Laplacian in 3D P_1 and P_2 using the PETSC/gasm solver with lu in the

Strong Scalability

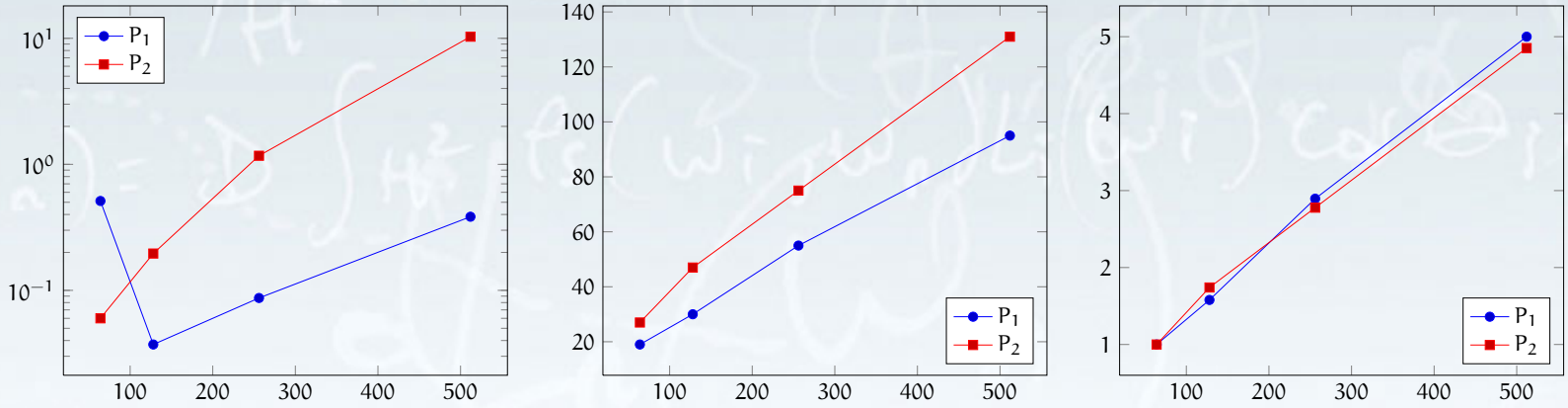


(a) solver time vs proc (b) rel. time vs proc (c) solver iterations vs proc

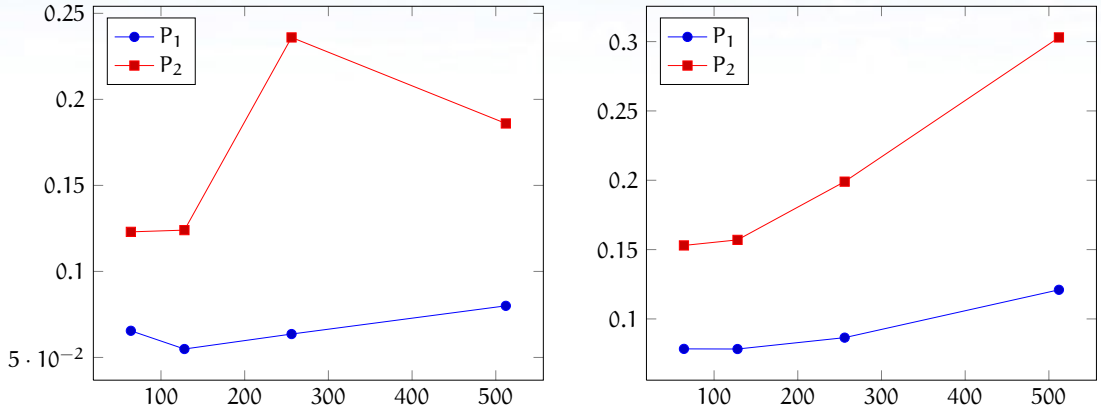
(d) rel. iterations vs proc (e) assembly(matrix) vs proc (f) assembly(vector) vs proc

Figure 6.2: Laplacian in 3D P_1 and P_2 using the PETSC/gasm solver with lu in the subdomains from 64 to 512 processors

Weak Scalability



(a) solver time vs proc (b) solver iterations vs proc (c) rel. iterations vs proc



(d) assembly(matrix) vs proc (e) assembly(vector) vs proc

Figure 6.3: Laplacian in 3D P_1 and P_2 using the PETSC/gasm solver with lu in the subdomains from 64 to 512 processors

Conclusions and Perspectives

Conclusions

- Parallelisation of Feel++ (almost) done
- First scalability results (almost) ok
- Domain decomposition framework (Schwarz, mortar, three fields)
- Generative programming for PDE works thanks to C++ (GCC and C++11) and compilation time improves (not there yet but better)
- Feedback: fast prototyping(at least for methodology), domain specific language, devil lurks in the details (interpolation, ...), used by physicist in micro-fluidic
- Wide range of applications

Perspectives

- New preconditioners (e.g. substructuring one for mortar in 2D and 3D)
- Exploit hybrid architectures (CPU/GPGPU)
- Application to multi-physics/multiscale problems

References I

Thank you !