Solving PDEs with Feel++

Organization of the Course Solving PDEs with Feel++

Part I: Introduction to Feel++

Part II: Solving Algebraic Problems with Feel++

Part III: Solving PDES with Feel++ Toolboxes

Part IV: Programming with Feel++

Outline of Part I: Introduction to Feel++

Outline of Part II: Solving Algebraic Problems with Feel++

Solver and Preconditioner Framework

- Preconditioning
- PETSc
- Basic Preconditioners
- Multigrid preconditioners
- Domain decomposition preconditioners
- KKT or saddle point systems

Outline of Part III: Solving PDES with Feel++ Toolboxes

Toolbox Framework

CFD Toolbox

Computation Solid Mechanics toolbox

Fluid Structure Interaction toolbox

Heat Transfer toolbox

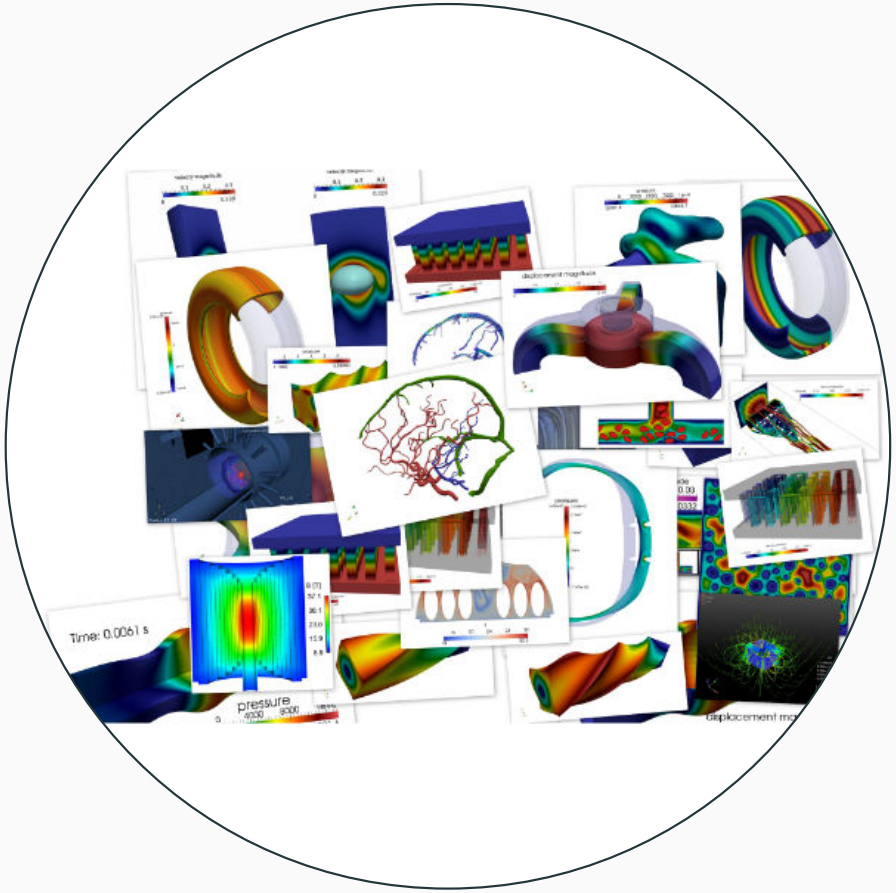
Thermo Electric toolbox

Outline of Part IV: Programming with Feel++

Part I

Introduction to Feel++

Versatile



A large range of numerical methods to solve partial differential equations: cG, dG, hdG, crb, ...in 1D, 2D and 3D

Powerful

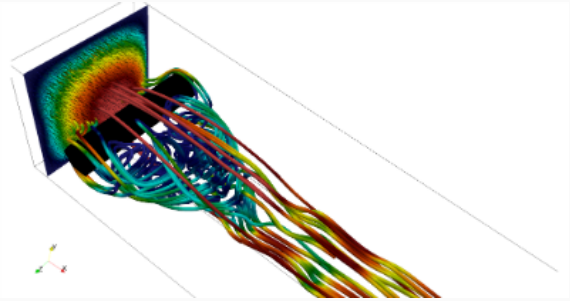
Support for high performance computing up to thousands of cores for linear and non-linear problems using PETSc/SLEPc and InHouse solution strategies

Expressive

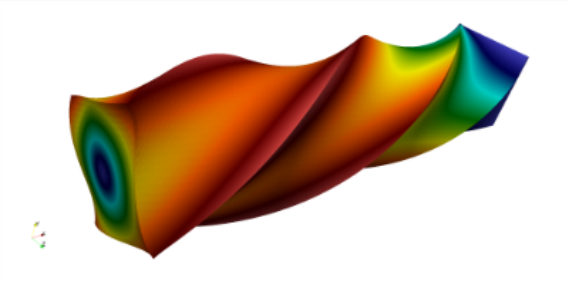
A language for Galerkin methods embedded into C++ for maximal mathematical expressivity.
Support for meshes, function spaces and elements, bilinear and linear forms, algebraic representation and post-processing.

Toolboxes :: Monophysics

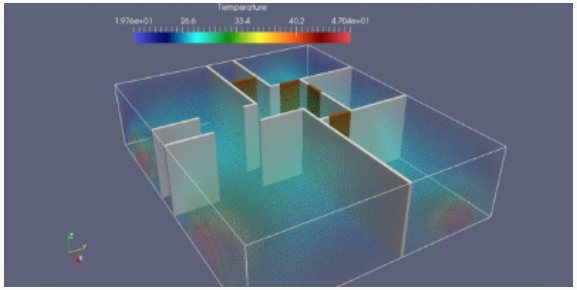
CFD



CSM

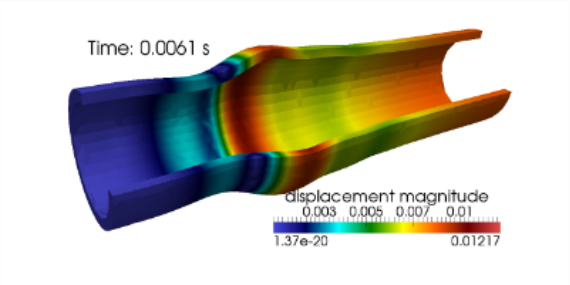


Heat Transfer

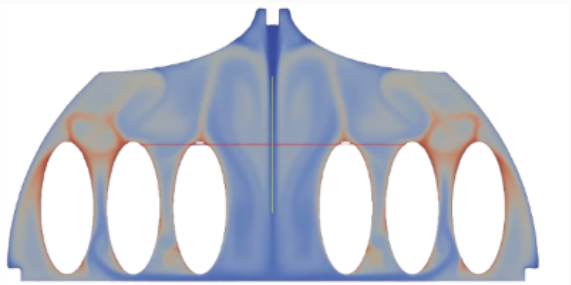


Toolboxes :: Multiphysics

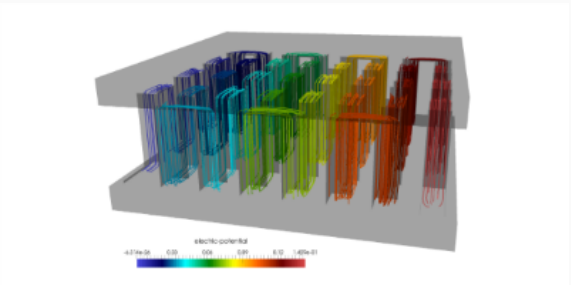
FSI



Heat & Fluid

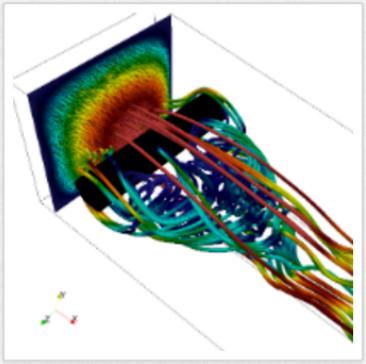


Thermoelectric

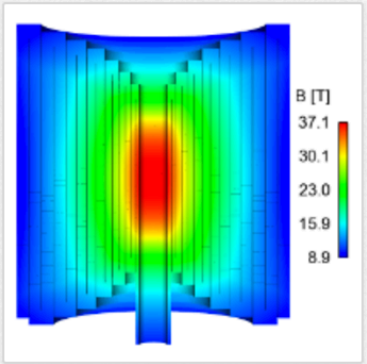


A wide range of numerical methods

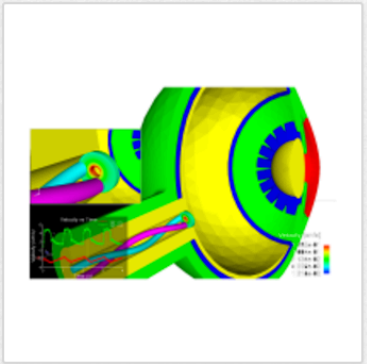
Continuous Galerkin



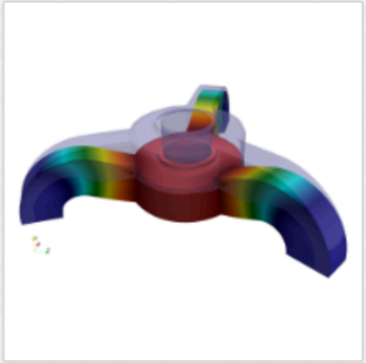
Hybrid Methods



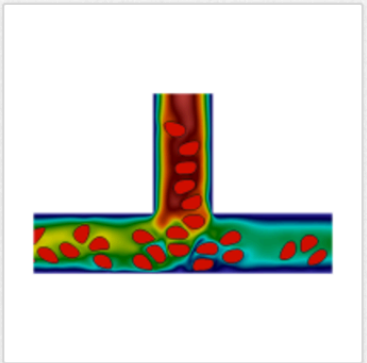
Hybrid Discontinuous Galerkin



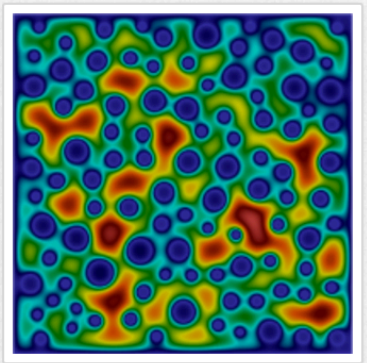
Reduced Basis



Levelset Methods

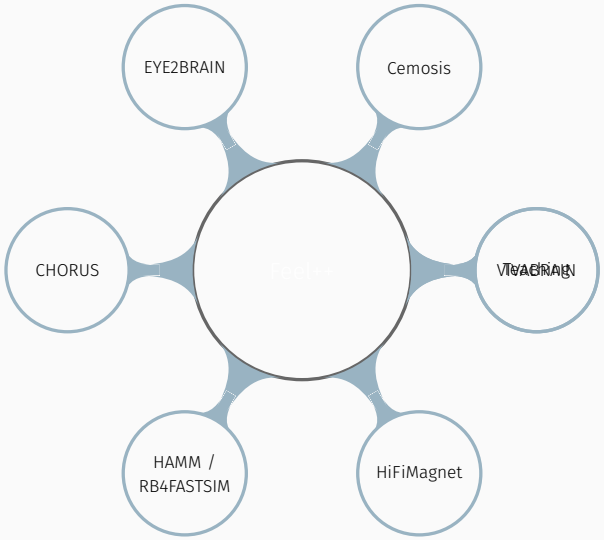


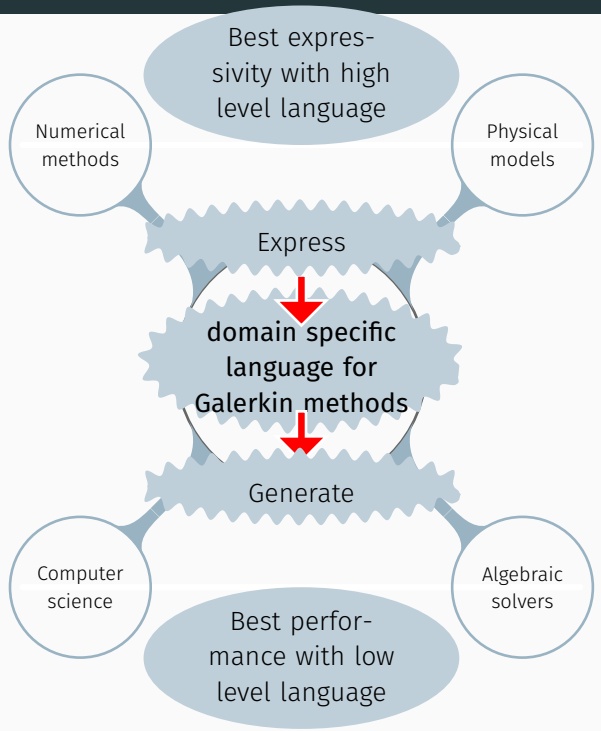
Fictitious Domain Methods



A large range of numerical methods to support of large range of applications in fluid mechanics, solid mechanics, heat transfert, blood rheology, eye modeling and simulation and electro-magnetism.

Feel++





Mathematical language for scientific computing

- to communicate between disciplines (Math, CS, Physics, Engineering...)
- to break complexity

Usage Scenarii: Research, R&D (“Bureau d’étude”), Teaching

Partners/Collaborators

France

- U. Strasbourg
- U. Grenoble Alpes
- UPMC
- LNCMI/CNRS
- U. Toulouse

Abroad

- IUPUI (USA)
- IMATI (Italy)
- U. Coimbra (Portugal)

Companies


AIRBUS
GROUP


eDF


AxesSim


Kitware


CHARIER


PLASTIC OMNIUM


Atos

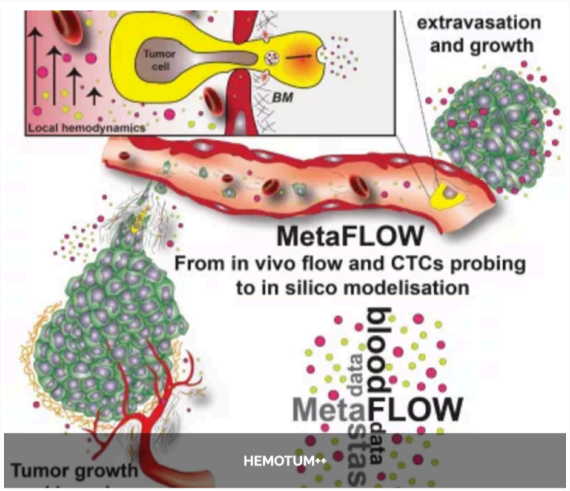

GAZOMAT

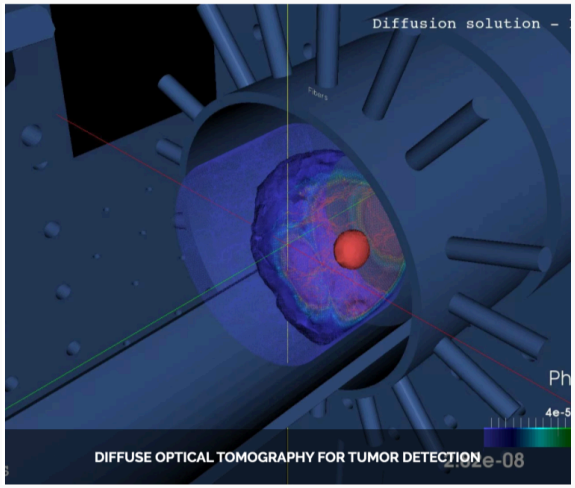

Holo3

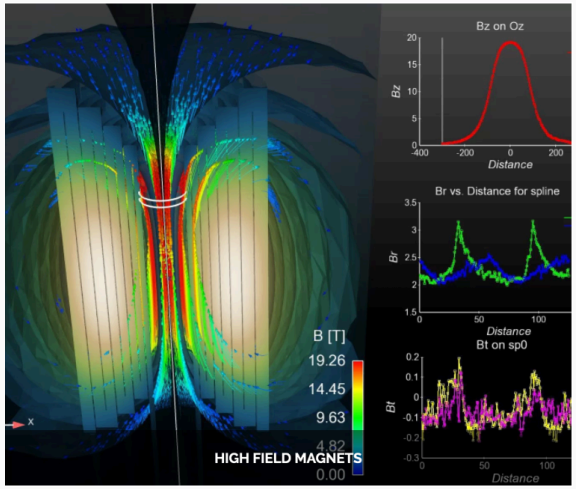

SIGMAPHI
MAGNETS AND BEAM TRANSPORT


see-∂
Sciences et Entreprises - ∂écisions

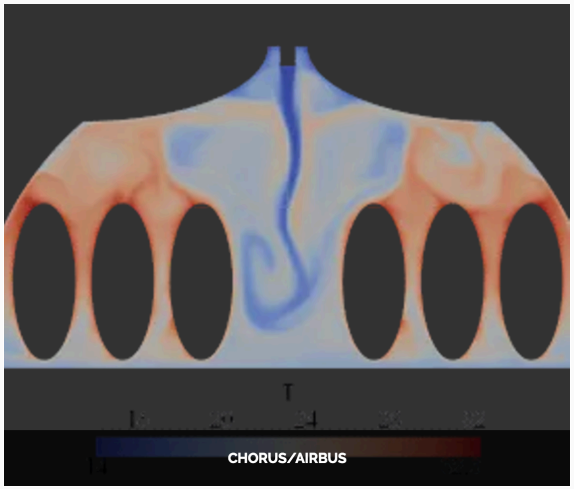
Projects with Feel++





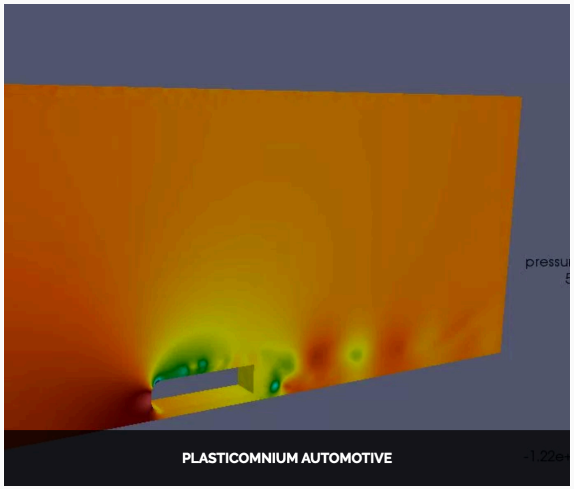


Projects with Feel++


CHORUS/AIRBUS


HOLO3


MSO4SC


PLASTICOMNIUM AUTOMOTIVE

SIVIBIR++

GAZOMAT

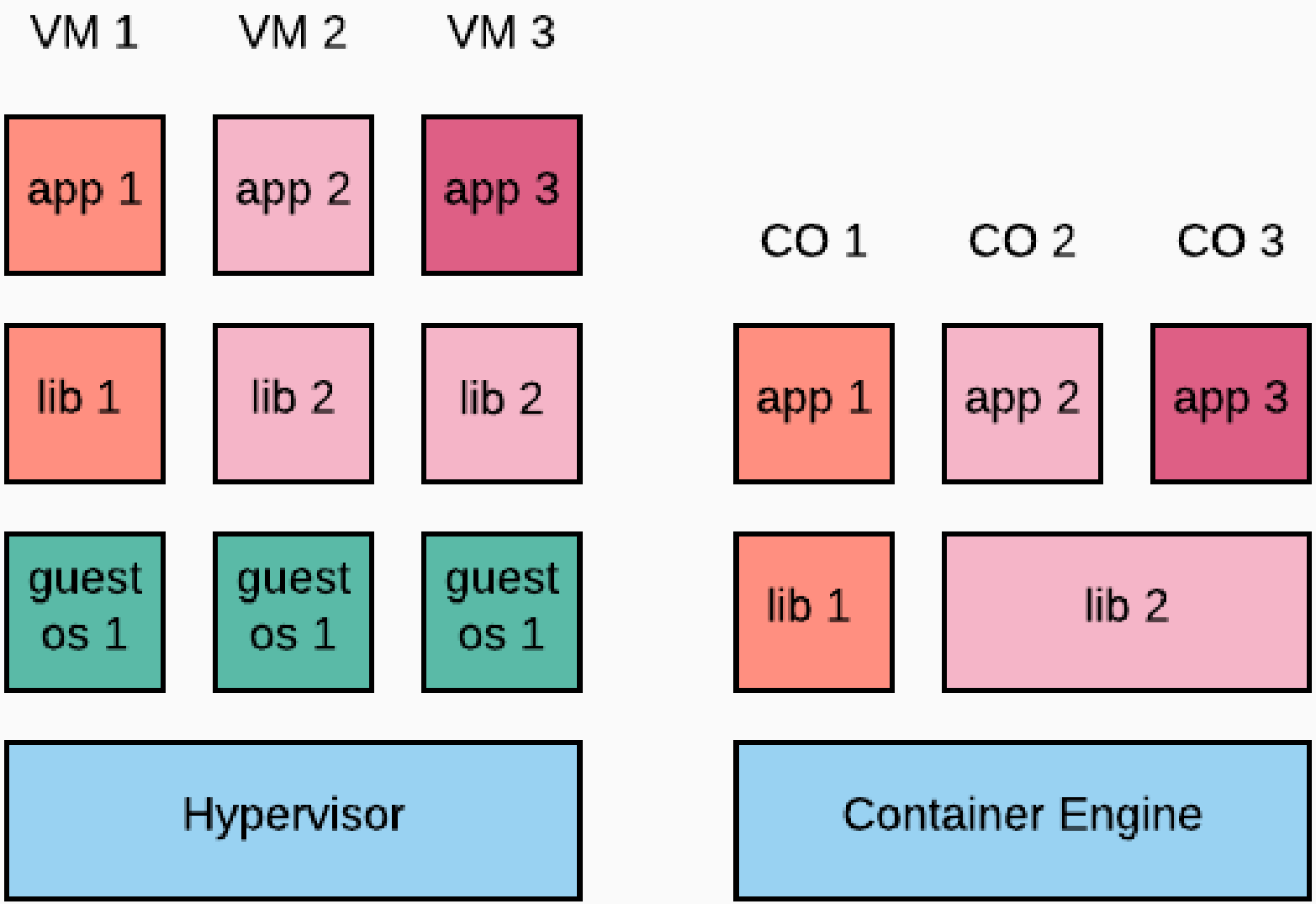
Feel++ web ecosystem

- : `http://www.feelpp.org`
- :: `https://github.com/feelpp/feelpp`
- : `http://plus.feelpp.org`
- : `http://forum.feelpp.org` (Gitter)
- : `http://publications.feelpp.org` (Hal)
- : `http://youtube.feelpp.org`
- : `http://twitter.feelpp.org`

Why Docker?

- Supports for all recent plateforms including Windows and Mac
- Ease of construction for various Linux flavors (see `http://travis-ci.org/feelpp/feelpp`)
- Ease of deployment thanks to `http://hub.docker.com/u/feelpp/dashboard/`

Some remarks about Docker



Docker (recommended) I

feelpp/feelpp-libs

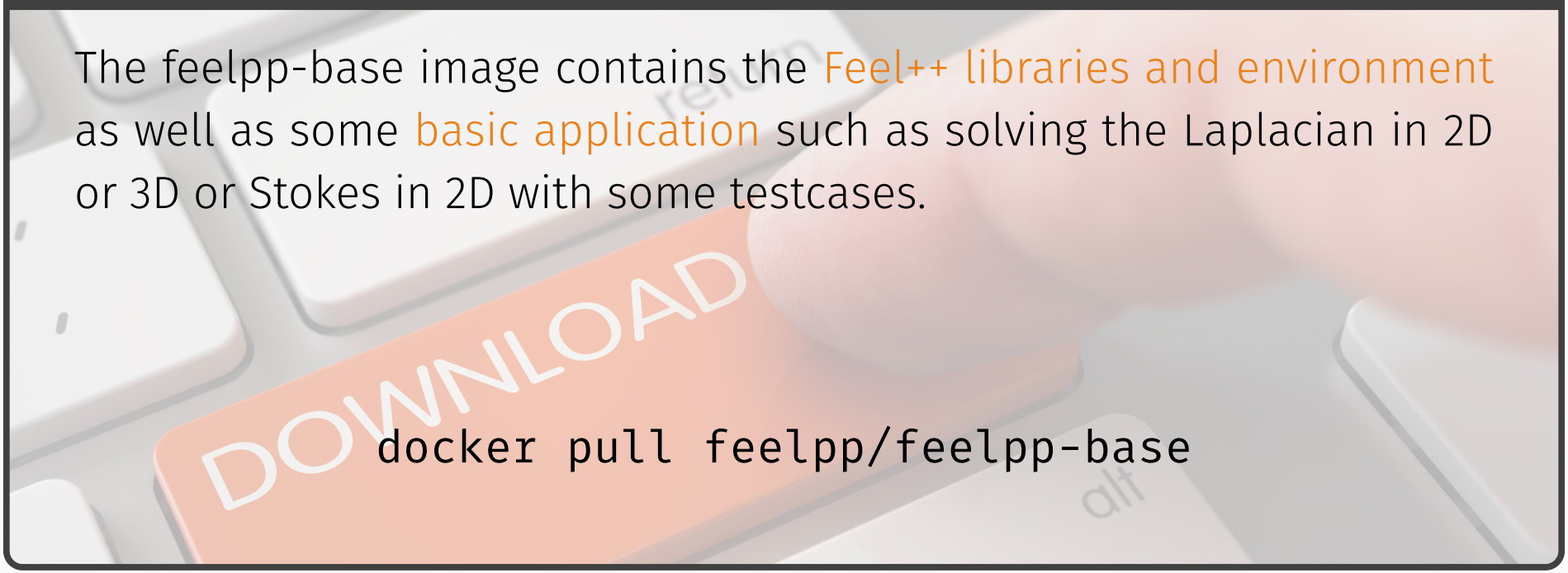
Feel++ Libraries and development environment is available from Docker Hub. It includes a mesh partitioner that generates partitioned mesh that can be read e.g. on a supercomputer in an efficient way.

```
docker pull feelpp/feelpp-libs
```

Docker (recommended) II

feelpp/feelpp-base

The feelpp-base image contains the Feel++ libraries and environment as well as some basic application such as solving the Laplacian in 2D or 3D or Stokes in 2D with some testcases.




```
docker pull feelpp/feelpp-base
```

Docker (recommended) III

feelpp/feelpp-toolboxes

The feelpp-toolboxes image contains the Feel++ libraries and environement, the basic applications and the toolboxes in Computational Fluid Dynamics/CFD, Computational Solid Mechanics/CSM, Fluid Structure Interaction/FSI, Heat Transfer/HT and Thermo-Electric/TE.

```
docker pull feelpp/feelpp-toolboxes
```

Ubuntu 16.10

```
sudo apt-get install automake autoconf libtool libboost-all-dev
bash-completion emacs24 gmsh libgmsh-dev libopenturns-dev
libbz2-dev libhdf5-openmpi-dev libeigen3-dev libcgal-dev
libopenblas-dev libcln-dev libcppunit-dev libopenmpi-dev
libann-dev libglpk-dev libpetsc3.7-dev libslepc3.7-dev
liblapack-dev libmpfr-dev paraview python-dev libhwloc-dev
libvtk6-dev libpcre3-dev python-h5py python-urllib3 xterm tmux
screen python-numpy python-vtk6 python-six python-ply wget bison
sudo xauth cmake flex gcc-6 g++-6 clang-3.9 clang++-3.9 git
ipython openmpi-bin pkg-config
```

Debian 9 and Sid

```
apt-get -y install
autoconf automake bash-completion bison cmake emacs24
flex git gmsh ipython libann-dev libboost-all-dev
libbz2-dev libcgal-dev libcln-dev libcppunit-dev
libeigen3-dev libglpk-dev libgmsh-dev
libhdf5-openmpi-dev libhwloc-dev liblapack-dev
libmpfr-dev libopenblas-dev libopenmpi-dev
libopenturns-dev libpcre3-dev libtool libvtk6-dev
openmpi-bin paraview petsc-dev pkg-config python-dev
python-h5py python-numpy python-ply python-six
python-urllib3 python-vtk6 screen slepc-dev sudo
tmux wget xauth xterm zsh
```

Using Homebrew on MacOS X

```
brew tap homebrew/homebrew-science
brew tap feelpp/homebrew-feelpp
# install feel++
brew install feelpp
# or  its dependencies
brew install --only-dependencies feelpp
```

and Windows?

There is a good chance that with Bash for Windows 10, Feel++ can be compiled right away on Windows (Ubuntu).

Laplacian

Look for u such that

$$\begin{cases} -\Delta u = f \text{ in } \Omega \\ u = g \text{ on } \partial\Omega_D \\ \dfrac{\partial u}{\partial n} = h \text{ on } \partial\Omega_N \\ \dfrac{\partial u}{\partial n} + u = l \text{ on } \partial\Omega_R \end{cases}$$

Laplacian

We now turn to the finite element approximation using Lagrange finite element. We assume Ω to be a segment in 1D, a polygon in 2D or a polyhedron in 3D. We denote $V_\delta \subset H^1(\Omega)$ an approximation space such that $V_{g,\delta} \equiv P^k_{c,\delta} \cap H^1_{g,\Gamma_D}(\Omega)$.

Look for $u_\delta \in V_\delta$ such that

$$\int_{\Omega_\delta} \nabla u_\delta \cdot \nabla v_\delta + \int_{\Gamma_{R,\delta}} u_\delta \, v_\delta = \int_{\Omega_\delta} f \, v_\delta + \int_{\Gamma_{N,\delta}} g \, v_\delta + \int_{\Gamma_{R,\delta}} l \, v_\delta, \quad \forall v_\delta \in V_{0,\delta}$$

Laplacian I

```
//# marker1 #
using namespace Feel;
using Feel::cout;
    po::options_description laplacianoptions( "Laplacian options" );
    laplacianoptions.add_options()
    ( "mu", po::value<double>()->default_value( 1.0 ), "coeff" )
    ( "no-solve", po::value<bool>()->default_value( false ), "No solve
            ;

    Environment env( _argc=argc, _argv=argv,
                _desc=laplacianoptions,
                _about=about(_name="qs_laplacian",
                             _author="Feel++ Consortium",
                             _email="feelpp-devel@feelpp.org"));
//# endmarker1 #
```

Laplacian II

```
tic();
//! [mesh]
auto mesh = loadMesh(_mesh=new Mesh<Simplex<FEELPP_DIM,1>>);
//! [mesh]
toc("loadMesh");

tic();
//! [discr]
auto Vh = Pch<2>( mesh );
auto u = Vh->element("u");
auto mu = doption(_name="mu");
auto f = expr( soption(_name="functions.f"), "f" );
// Robin left hand side expression
auto r_1 = expr( soption(_name="functions.a"), "a" );
// Robin right hand side expression
auto r_2 = expr( soption(_name="functions.b"), "b" );
// Neumann expression
```

Laplacian III

```
auto n = expr( soption(_name="functions.c"), "c" );
auto g = expr( soption(_name="functions.g"), "g" );
//! [v]
auto v = Vh->element( g, "g" );
//! [v]
//! [discr]
toc("Vh");



tic();
//! [vf]
auto l = form1( _test=Vh );
l = integrate(_range=elements(mesh),
              _expr=f*id(v));
l+=integrate(_range=markedfaces(mesh,"Robin"), _expr=r_2*id(v));
l+=integrate(_range=markedfaces(mesh,"Neumann"), _expr=n*id(v));
//! [vf]
```

Laplacian IV

```
toc("l");

tic();
//! [vf]
auto a = form2( _trial=Vh, _test=Vh);
a = integrate(_range=elements(mesh),
              _expr=mu*gradt(u)*trans(grad(v)) );
a+=integrate(_range=markedfaces(mesh,"Robin"),
             _expr=r_1*idt(u)*id(v));
a+=on(_range=markedfaces(mesh,"Dirichlet"), _rhs=l,
      _element=u, _expr=g );
//! if no markers Robin Neumann or Dirichlet are present in the mesh t
//! impose Dirichlet boundary conditions over the entire boundary
if ( !mesh->hasAnyMarker({"Robin", "Neumann","Dirichlet"}) )
    a+=on(_range=boundaryfaces(mesh), _rhs=l, _element=u, _expr=g );
//! [vf]
toc("a");
```

Laplacian V

```
tic();
//! [solve]
//! solve the linear system, find u s.t. a(u,v)=l(v) for all v
if ( !boption( "no-solve" ) )
    a.solve(_rhs=l,_solution=u);
//! [solve]
toc("a.solve");

//! [ug]
cout << "||u_h-g||_L2=" << normL2(_range=elements(mesh),
                                  _expr=idv(u)-g) << std::endl;
//! [ug]

tic();
//! [export]
auto e = exporter( _mesh=mesh );
e->addRegions();
e->add( "u", u );
```

Laplacian VI

```
    e->add( "g", v );
    e->save();
    //! [export]

    toc("Exporter");
    return 0;

}
```

Running Laplacian in Docker

```
# 2D
cd Testcases/quickstart/laplacian/feelpp2d/
mpirun -np 4 feelpp_qs_laplacian_2d --config-file feelpp2d.cfg

# 3D
cd Testcases/quickstart/laplacian/feelpp3d/
mpirun -np 4 feelpp_qs_laplacian_3d --config-file feelpp3d.cfg
```

Stokes

Find (u, p) such that

$$\text{Stokes:} \begin{cases} -\mu\Delta\mathbf{u} + \nabla p = \mathbf{f} \\ \nabla \cdot \mathbf{u} = 0 \\ \text{BC} \end{cases} \tag{1}$$

Stokes I

```
using namespace Feel;
    po::options_description laplacianoptions( "Stokes options" );
    laplacianoptions.add_options()
    ( "mu", po::value<double>()->default_value( 1.0 ), "viscosity" )
    ( "no-solve", po::value<bool>()->default_value( false ), "No solve
            ;

    Environment env( _argc=argc, _argv=argv,
                _desc=laplacianoptions,
                _about=about(_name="qs_stokes",
                             _author="Feel++ Consortium",
                             _email="feelpp-devel@feelpp.org"));

tic();
auto mesh = loadMesh(_mesh=new Mesh<Simplex<FEELPP_DIM,1>>);
toc("loadMesh");
```

Stokes II

```
tic();
auto Vh = THch<1>( mesh );
auto U = Vh->element("U");
auto u = U.element<0>();
auto p = U.element<1>();
auto v = U.element<0>();
auto q = U.element<1>();
auto mu = doption(_name="mu");
auto f = expr<FEELPP_DIM,1>( soption(_name="functions.f"), "f" );
auto g = expr<FEELPP_DIM,1>( soption(_name="functions.g"), "g" );
toc("Vh");

tic();
auto l = form1( _test=Vh );
l = integrate(_range=elements(mesh),
              _expr=inner(f,id(v)));
toc("l");
```

Stokes III

```
tic();
auto a = form2( _trial=Vh, _test=Vh);
auto Id = eye<FEELPP_DIM,FEELPP_DIM>();
auto deft = sym(gradt(u));
auto sigmat = -idt(p)*Id + 2*mu*deft;
a = integrate(_range=elements(mesh),
              _expr=inner( sigmat, grad(v) ) );
a += integrate(_range=elements(mesh),
               _expr=id(q)*divt(u) );
a+=on(_range=markedfaces(mesh,"inlet"), _rhs=l, _element=u, _expr=g );
a+=on(_range=markedfaces(mesh,{"wall","letters"}), _rhs=l, _element=u,
      _expr=zero<FEELPP_DIM,1>() );
toc("a");

tic();
//! solve the linear system, find u s.t. a(u,v)=l(v) for all v
if ( !boption( "no-solve" ) )
    a.solve(_rhs=l,_solution=U);
```

Stokes IV

```
toc("a.solve");

tic();
auto e = exporter( _mesh=mesh );
e->addRegions();
e->add( "u", u );
e->add( "p", p );
e->save();
toc("Exporter");
return 0;
```

Running Stokes in Docker

```
# 2D
cd Testcases/quickstart/laplacian/feelpp2d/
mpirun -np 4 feelpp_qs_laplacian_2d --config-file feelpp2d.cfg

# 3D
cd Testcases/quickstart/laplacian/feelpp3d/
mpirun -np 4 feelpp_qs_laplacian_3d --config-file feelpp3d.cfg
```

Part II

Solver and Preconditioner framework

Solver and Preconditioner Framework

- Preconditioning
- PETSc
- Basic Preconditioners
- Multigrid preconditioners
- Domain decomposition preconditioners
- KKT or saddle point systems

Preconditioning

A preconditioned iterative solver solves the system

$$M^{-1}Ax = M^{-1}b. \tag{2}$$

The matrix M is called the preconditioner. The preconditioner should satisfy certain requirements:

- Convergence should be much faster for the preconditioned system than for the original system. Normally this means that M is constructed as an easily invertible approximation to A. Note that if $M = A$ any iterative method converges in one iteration.
- Operations with M^{-1} should be cheap to perform, at least cheaper than A^{-1}

Preconditioning

Of course the matrix $M^{-1}A$ is not explicitly formed. The multiplication $u = M^{-1}Av$ can simply be carried out by the operations

$$t = Av; u = M^{-1}t$$

Preconditioning

Preconditioners can be applied in different ways: From the left

$$M^{-1}Ax = M^{-1}b,$$

centrally

$$M = LU; L^{-1}AU^{-1}y = L^{-1}b; x = U^{-1}y,$$

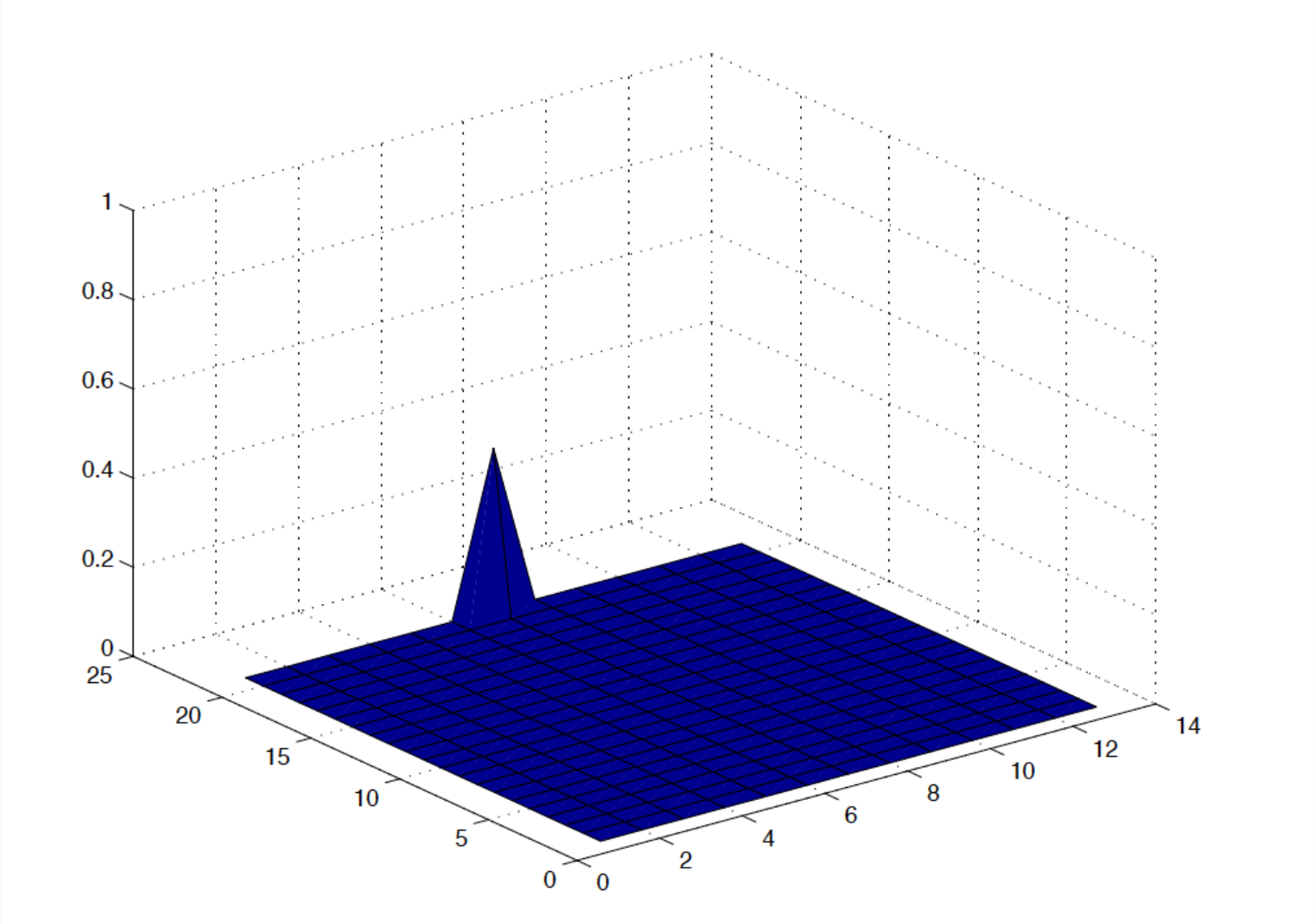
or from the right

$$AM^{-1}y = b; x = M^{-1}y.$$

Left, right and central preconditioning gives the same spectrum. Yet there are other differences:

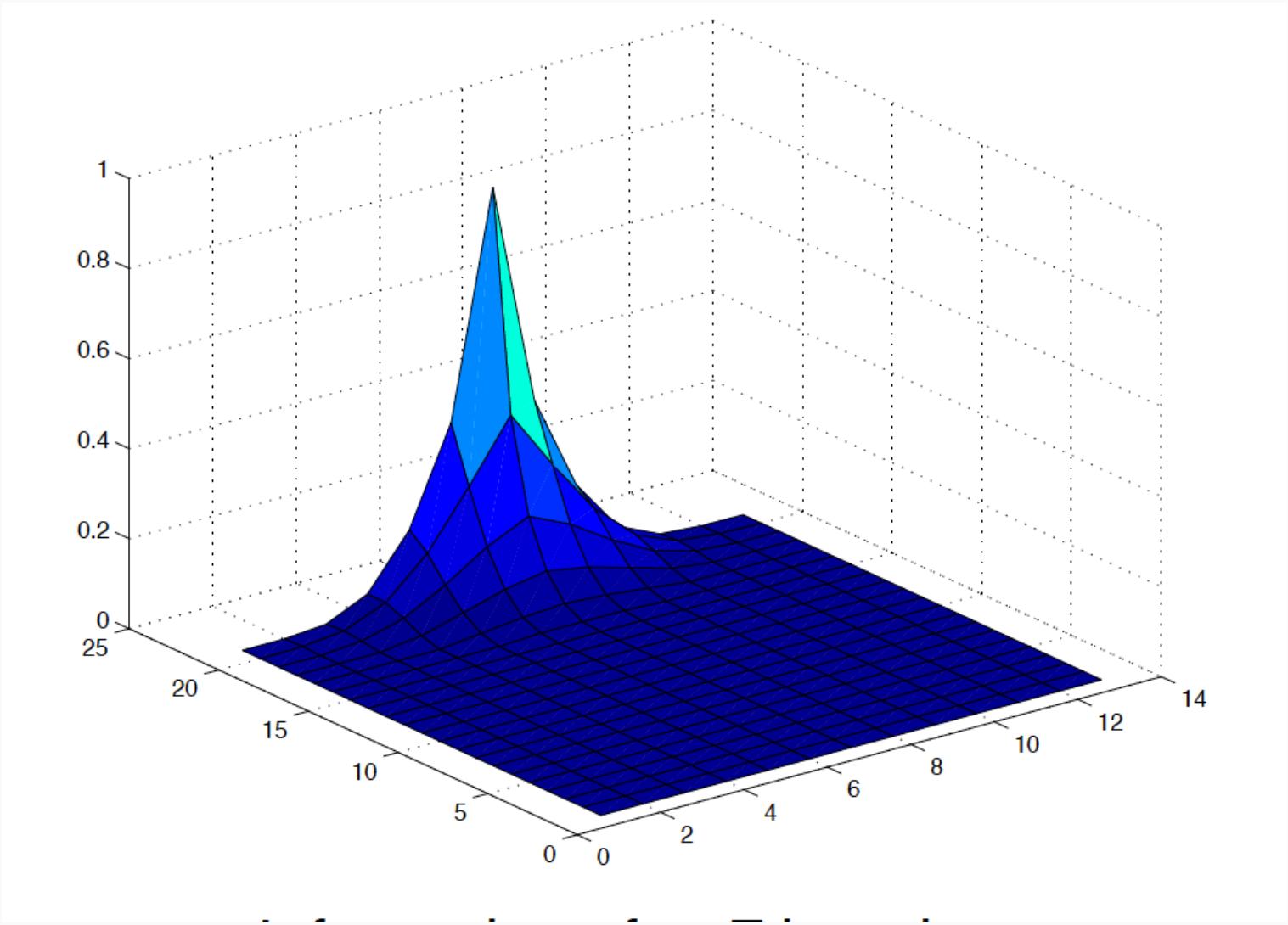
- Left preconditioning is most natural: no extra step is required to compute x;
- Central preconditioning preserves symmetry;
- Right preconditioning does not affect the residual norm

Why preconditioning?



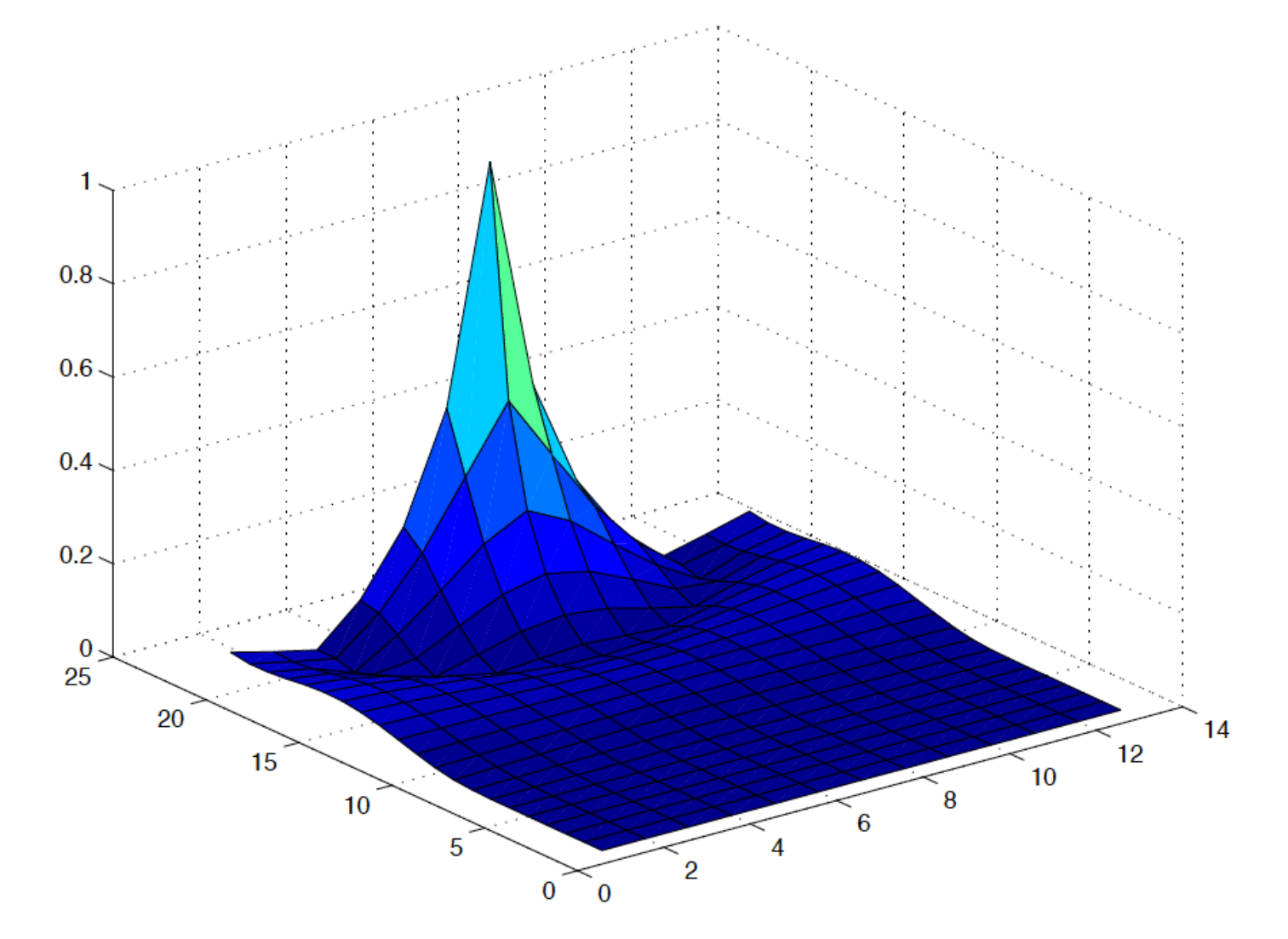
Iteration 1

Why preconditioning?



Iteration 7

Why preconditioning?



Iteration 21

Why preconditioning?

From the previous pictures it is clear that we need $O(1/h) = O(\sqrt{n})$ iterations to move information from one end to the other end of the grid.

So at best it takes $O(n^{3/2})$ operations to compute the solution with an iterative method.

In order to improve this we need a preconditioner that enables fast propagation of information through the mesh.

PETSc

Feel++ uses PETSc as the main backend for algebraic solves

- We call the algebraic interface a `Backend`
- Iterative solvers and preconditioners can be set in `.cfg` file using `ksp-type` and `pc-type` respectively
- idem in command line
- the option naming reflects the PETSc option naming
- use `ksp-monitor` to inspect solver iterations
- use `ksp-view` to inspect solver configuration

```
feelpp_qs_laplacian_2d --help-lib | grep ksp
feelpp_qs_laplacian_2d --help | grep ksp
```

Direct solvers

use a Direct solver interfaced by PETSc such as MUMPS or Umfpack

```
ksp-type=preonly # no iterative solve
pc-type=lu
pc-factor-mat-solver-package-type=mumps # or umfpack in sequential
```

Diagonal scaling

Diagonal scaling or Jacobi preconditioning uses

$$M = \mathrm{diag}(A)$$

as preconditioner. Clearly, this preconditioner does not enable fast propagation through a grid. On the other hand, operations with $\mathrm{diag}(A)$ are very easy to perform and diagonal scaling can be useful as a first step, in combination with other techniques.

```
# diagonal preconditioner
pc-type=jacobi
```

Gauss-Seidel, SOR, SSOR

The Gauss-Seidel preconditioner is defined by

$$M = L + D$$

in which L is the strictly lower-triangular part of A and $D = \mathrm{diag}(A)$. By introducing a relaxation parameter, we get the SOR-preconditioner.

For symmetric problems it is wise to take a symmetric preconditioner. A symmetric variant of Gauss-Seidel is defined by

$$M = (L + D)D^{-1}(L + D)^T$$

By introducing a relaxation parameter we get the so called SSOR-preconditioner.

```
# GS preconditioner
pc-type=sor
```

ILU Preconditioners

ILU-preconditioners are the most popular *black box* preconditioners. They are constructed by making a standard LU-decomposition

$$A = LU$$

However, during the elimination process some nonzero entries in the factors are discarded. This can be done on basis of two criteria:

- Sparsity pattern: e.g. an entry in a factor is only kept if it corresponds to a nonzero entry in A;
- Size: small entries in the decomposition are dropped.

```
# ilu preconditioner
pc-type=ilu
```

ILU Preconditioners

The number of nonzero entries that is maintained in the LU-factors is normally of the order of the number of nonzeros in A.

This means that operations with the LU-preconditioner are approximately as costly as multiplications with A.

For A Symmetric Positive Definite a special variant of ILU exists, called Incomplete Choleski. This preconditioner is based on the Choleski decomposition $A = CC^T$.

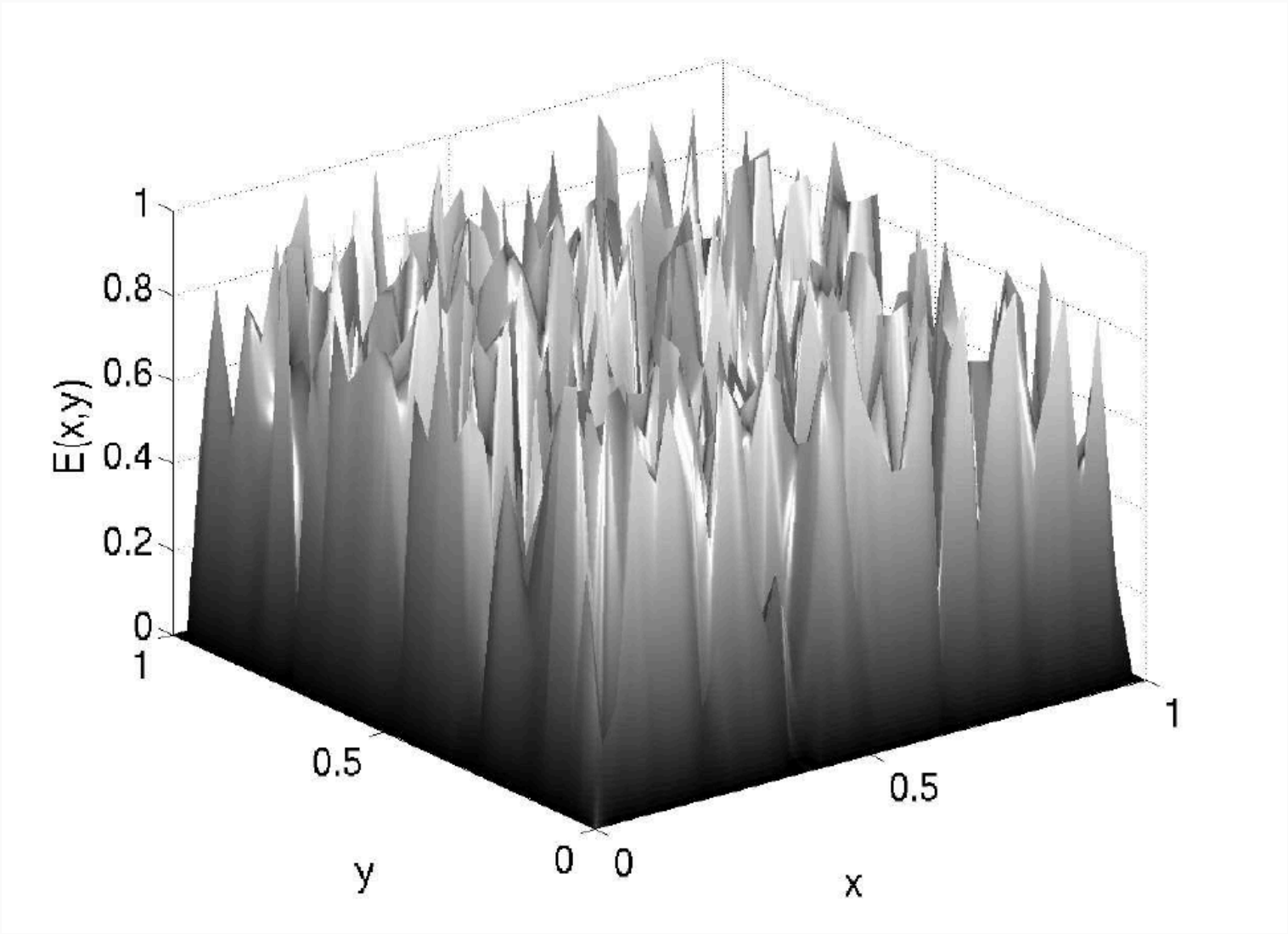
```
1 # icc preconditioner
2 pc-type=icc
```

Basic preconditioners

Although the preconditioners discussed before can considerably reduce the number of iterations, they do not normally reduce the mesh-dependency of the number of iterations.

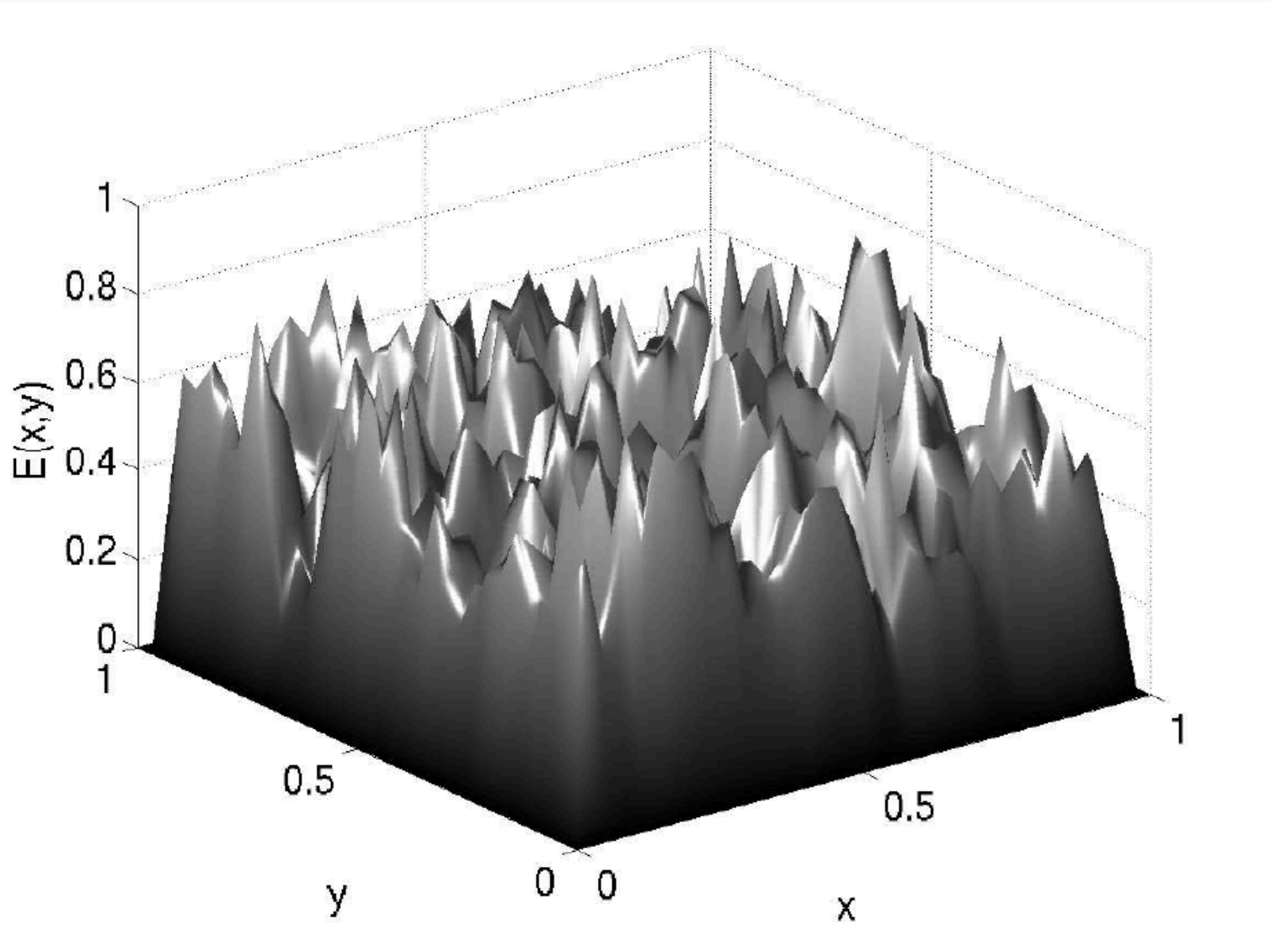
In the next slides we take a closer look at how basic iterative methods reduce the error. From the observations we make, we will develop the idea that is at the basis of one of the fastest techniques: multigrid.

Smoothing



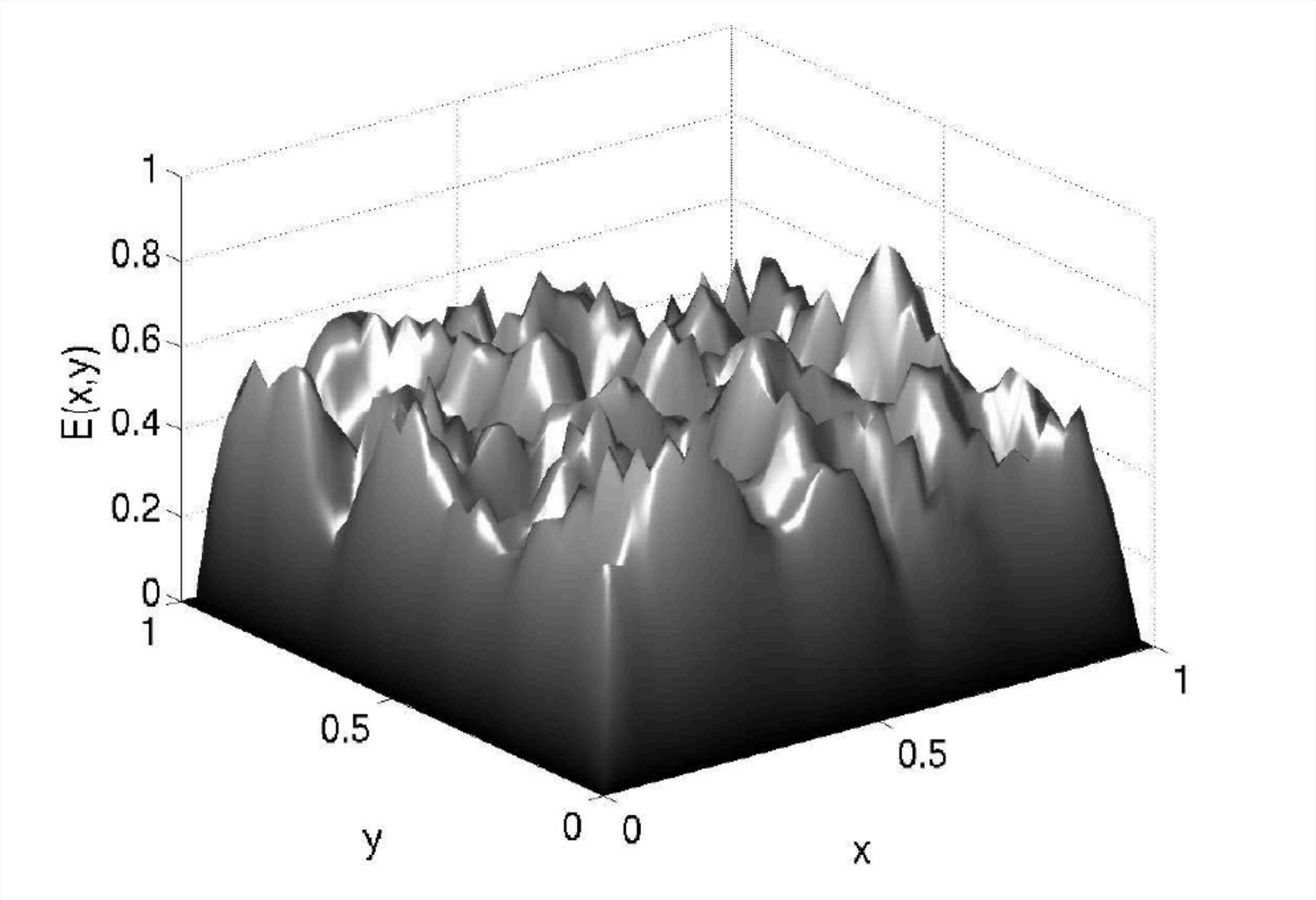
Random initial error

Smoothing



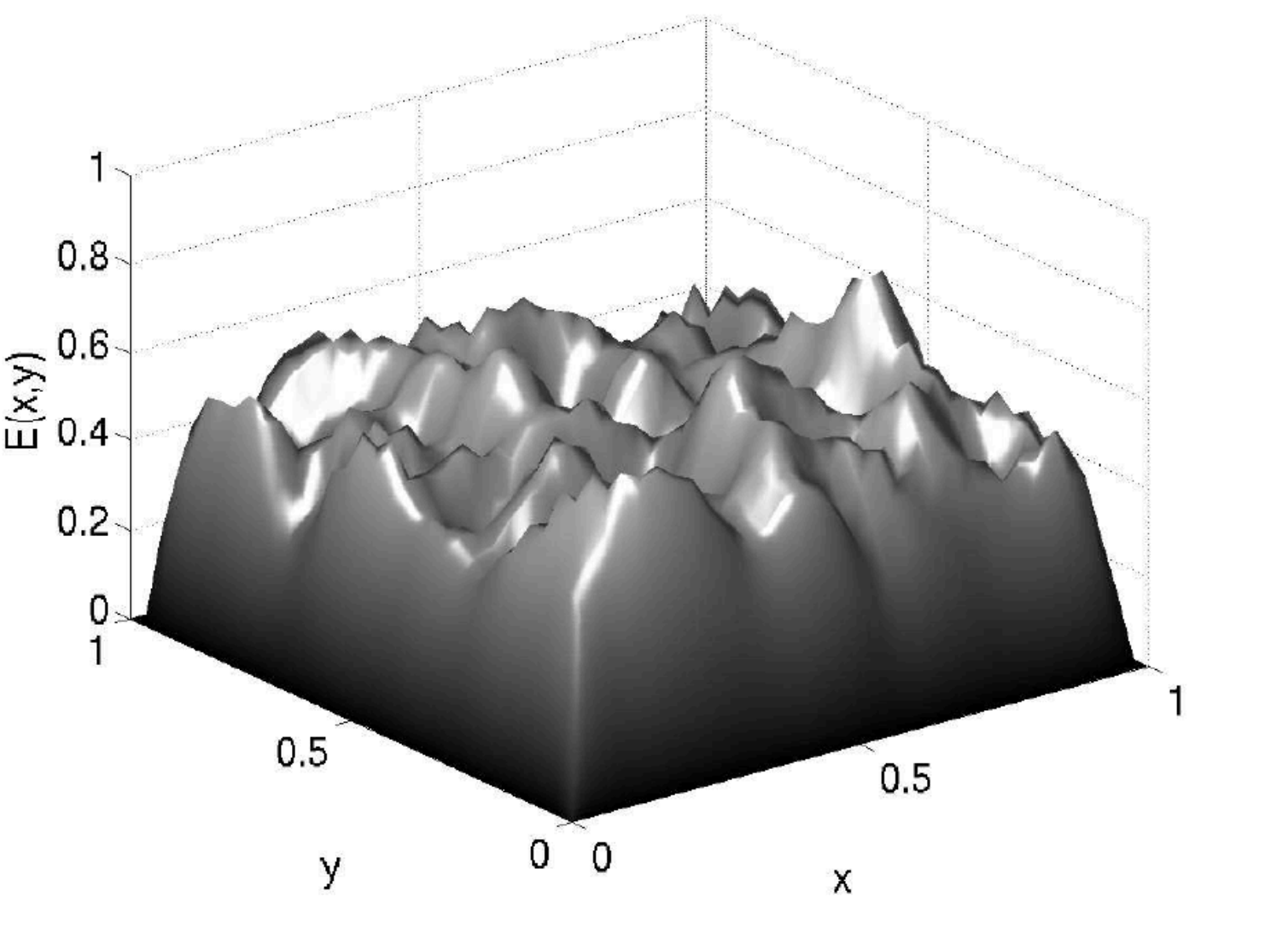
Error after 1 Jacobi ieration

Smoothing



Error after 2 Jacobi ieration

Smoothing



Error after 3 Jacobi ieration

Complementarity

- Error after a few weighted Jacobi iterations has structure, this is the same for the other basic iterative methods.
- Instead of discarding the method, look to complement its failings

How can we best correct errors that are slowly reduced by basic iterative method?

Complementarity

- Error after a few weighted Jacobi iterations has structure, this is the same for the other basic iterative methods.
- Instead of discarding the method, look to complement its failings

How can we best correct errors that are slowly reduced by basic iterative method?

- Slow-to-converge errors are smooth
- Smooth vectors can be accurately represented using fewer degrees of freedom

Coarse Grid Correction

Smooth vectors can be accurately represented using fewer degrees of freedom

- Idea: transfer job of resolving smooth components to a coarser grid version of the problem
- Need:
  - Complementary process for resolving smooth components of the error on the coarse grid
  - Way to combine the results of the two processes

Multigrid

Relaxation is the name for applying one or a few basic iteration steps.

- Idea is to correct the approximation after relaxation, $x^{(1)}$, from a coarse-grid version of the problem
- Need interpolation map, P, from coarse grid to fine grid
- Corrected approximation will be $x^{(2)} = x^{(1)} + Px_c$
- x_c is the solution of the coarse-grid problem and satisfies

$$(P^T AP)x_c = P^T A(x - x^{(1)}) = P^T r^{(1)}$$

Two-grid cycle

Components

- Relaxation Relax: $x^{(1)} = x^{(0)} + M^{-1} r^{(0)}$

Two-grid cycle

Components

- Relaxation Relax: $x^{(1)} = x^{(0)} + M^{-1} r^{(0)}$
  - Use a smoothing process (such as Jacobi or Gauss-Seidel) to eliminate oscillatory errors
  - Remaining error satisfies $Ae^{(1)} = r^{(1)} = b - Ax^{(1)}$

Two-grid cycle

Components

- Relaxation Relax: $x^{(1)} = x^{(0)} + M^{-1} r^{(0)}$
- Restriction
  - Transfer residual to coarse grid
  - Compute $P^T r^{(1)}$

Two-grid cycle

Components

- Relaxation Relax: $x^{(1)} = x^{(0)} + M^{-1} r^{(0)}$
- Restriction
- Coarse Grid Correction : Solve: $P^T A P x_c = P^T r^{(1)}$
  - Use coarse-grid correction to eliminate smooth errors
  - Best correction x_c satisfies

$$P^T A P x_c = P^T r^{(1)}$$

Two-grid cycle

Components

- Relaxation Relax: $x^{(1)} = x^{(0)} + M^{-1} r^{(0)}$
- Restriction
- Coarse Grid Correction : Solve: $P^T A P x_c = P^T r^{(1)}$
- Interpolation : Solve: $P^T A P x_c = P^T r^{(1)}$
  - Transfer correction to fine grid
  - Compute $x^{(2)} = x^{(1)} + P x_c$

Two-grid cycle

Components

- Relaxation Relax: $x^{(1)} = x^{(0)} + M^{-1} r^{(0)}$
- Restriction
- Coarse Grid Correction : Solve: $P^T A P x_c = P^T r^{(1)}$
- Interpolation : Solve: $P^T A P x_c = P^T r^{(1)}$
- Relaxation
  - Relax once again to remove oscillatory error introduced in coarse-grid correction

Two-grid cycle

Components

- Relaxation Relax: $x^{(1)} = x^{(0)} + M^{-1} r^{(0)}$
- Restriction
- Coarse Grid Correction : Solve: $P^T A P x_c = P^T r^{(1)}$
- Interpolation : Solve: $P^T A P x_c = P^T r^{(1)}$
- Relaxation

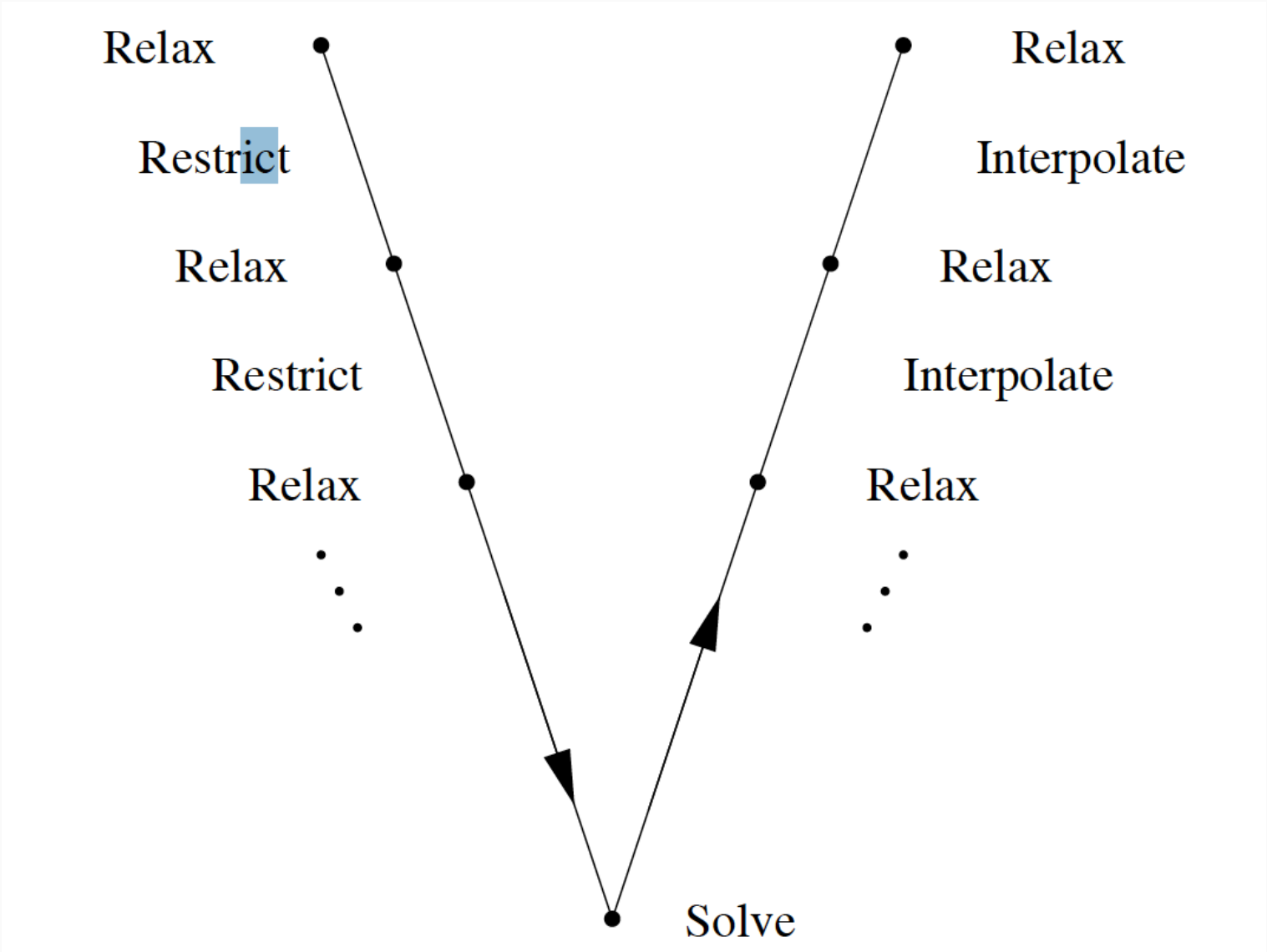
Be careful with solution of coarse-grid problem !

Two-grid cycle

Components

- Relaxation Relax: $x^{(1)} = x^{(0)} + M^{-1} r^{(0)}$
- Restriction
- Coarse Grid Correction : Solve: $P^T A P x_c = P^T r^{(1)}$
- Interpolation : Solve: $P^T A P x_c = P^T r^{(1)}$
- Relaxation

Use Recursion ! apply same methodology to solve coarse grid problem -> V-Cycle

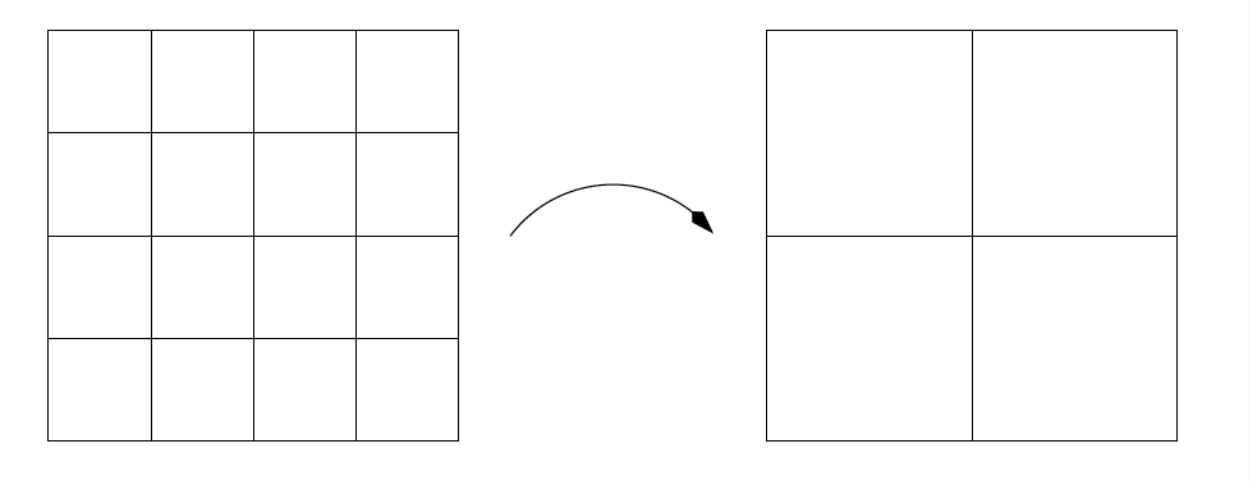

Relax
Restrict
Relax
Restrict
Relax
Solve
Relax
Interpolate
Relax
Interpolate
Relax

Properties of Effective Cycles

- Fast convergence
  - Effective reduction of all error components
  - On each level, coarse-grid correction must effectively reduce exactly those errors that are slow to be reduced by relaxation alone
  - Hierarchy of coarse-grid operators resolves relevant physics at each scale
- Low iteration cost
  -
  - Simple relaxation scheme (cheap computation of $M^{-1}r$ on all levels)
  - Sparse coarse-grid operators
  - Sparse interpolation/restriction operations

Choosing coarse grids

- No best strategy to choose coarse grids
- Operator dependent, but also machine dependent



- For structured meshes, often use uniform de-refinement approach
- For unstructured meshes, various weighted independent set algorithms are often used.

What is still missing ?

- How do we choose P?
  - Number of columns
  - Sparsity structure
  - Non-zero values
- Choices depend closely on the properties of the relaxation method

Remarks on multigrid

Multigrid works well if the problem

- is grid-based. However, matrix-based multigrid methods (*Algebraic Multigrid*) do exist and are often successful;
- has a smooth solution. An underlying assumption is that the solution can be represented on a coarser grid. Multigrid works particularly well for Poisson-type problems. For these problems the number of operations is $O(n)$.

Multigrid can be used as a separate solver, but is often used as a preconditioner for a Krylov-type method.

```
pc-type=gamg # can use of hypre if available
pc-mg-levels=4 # number of levels
gamg-nsmooths=1 # number of smoothign steps
```

Domain decomposition preconditioners

A standard way to parallelise a grid-based computation is to split the domain into p subdomains, and to map each subdomain on a processor.

It is a natural idea to solve the subdomain problems independently and to iterate to correct for the error. This idea has given rise to the family of domain decomposition preconditioners.

The theory for domain decomposition preconditioners is vast,

Domain Decomposition

The domain decomposition decomposes the system $Ax = b$ into blocks. For two subdomains one obtains

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

x_1 and x_2 represent the subdomain unknowns, A_{11} and A_{22} the subdomain discretization matrices and A_{12} and A_{21} the coupling matrices between subdomains.

Domain Decomposition Preconditioners

It is a natural idea to solve a linear system $Ax = b$ by solving the subdomain problems independently and to iterate to correct for the error.

This idea has given rise to the family of domain decomposition preconditioners.

The theory for domain decomposition preconditioners is vast, here we only discuss some important ideas.

Block Jacobi Preconditioners

A simple domain decomposition preconditioner is defined by

$$M = \begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} \end{pmatrix} \text{(Block-Jacobi preconditioner).}$$

or, in general by

$$M = \begin{pmatrix} A_{11} & & \\ & \ddots & \\ & & A_{MM} \end{pmatrix}$$

The subdomain systems can be solved in parallel.

```
# global preconditioner block jacobi
pc-type=bjacobi
# preconditioner in the subdomains
sub-pc-type=lu
```

Solution of the Subdomain Problems

There are several ways to solve the subdomain problems:

- Exact solves. This is in general (too) expensive.
- Inexact solves using an incomplete decomposition (block-ILU).
- Inexact solves using an iterative method to solve the subproblems. Since in this case the preconditioner is variable, the outer iteration should be flexible, for example GCR.

On the Scalability of Block-Jacobi

Without special techniques, the number of iterations increases with the number of subdomains. The algorithm is not scalable.

To overcome this problem, techniques can be applied to enable the exchange of information between subdomains.

Improving the Scalabiliy

Two popular techniques that improve the flow of information are:

- Use an overlap between subdomains. Of course one has to ensure that the value of unknowns in gridpoints that belong to multiple domains is unique.
- Use a coarse grid correction. The solution of the coarse grid problem is added to the subdomain solutions. This idea is closely related to multigrid, since the coarse-grid solution is the non-local, smooth part of the error that cannot be represented on a single subdomain.

```
# Additive Schwarz with coarse preconditioner
pc-type=gasm # or asm
pc-gasm-overlap=1 # more means more communications
# direct solves in the subdomaine
sub-pc-type=lu
```

KKT or saddle point systems

KKT systems frequently arise in optimisation. KKT systems are also know as saddle-point systems. A KKT system has the following block structure

$$A = \begin{pmatrix} A & B^T \\ B & -C \end{pmatrix}$$

with F and C symmetric pos. def matrices. Systems with such a matrix arise in many other applications, for example in CFD (Stokes problem) and in structural engineering.

Preconditioning a KKT-system

Many preconditonres have been developed that make use of the fact that of the system matrix

$$A = \begin{pmatrix} F & B^T \\ B & -C \end{pmatrix}$$

a block LU decomposition can be made:

$$\begin{pmatrix} F & B^T \\ B & -C \end{pmatrix} = \begin{pmatrix} I & O^T \\ BF^{-1} & I \end{pmatrix} \begin{pmatrix} F & B^T \\ O & -M_S \end{pmatrix}$$

Here $M_S = BF^{-1}B^T + C$ is the Schur complement.

Preconditioning a KKT-system

Idea (e.g. Elman, Silvester, Wathen): take

$$P = \begin{pmatrix} F & B^T \\ O & -M_S \end{pmatrix}$$

as (right) preconditioner:

$$AP^{-1} = \begin{pmatrix} I & O^T \\ BF^{-1} & I \end{pmatrix}$$

has only eigenvalue 1: GMRES ready in 2 iterations. **BUT**

- Preconditioner is nonsymmetric
- Schur complement is too expensive to compute and has to be approximated

Preconditioning a KKT-system

An SPD block-diagonal preconditioner:

$$P = \begin{pmatrix} F & O^T \\ O & M_S \end{pmatrix}$$

Can be used with MINRES (short recurrences) Preconditioned matrix has three distinct eigenvalues!

MINRES needs three iterations.

The main question is again how to make a cheap approximation to the Schur complement.

Block preconditioners: PCD I

```
[fluid]
pc-type=blockns
ksp-type=fgmres # fgmres,gcr
ksp-use-initial-guess-nonzero=0
gcr-restart=100
fgmres-restart=100

[fluid.blockns]
type=PCD #PMM,PCD

[blockns]
cd=true
pcd=true
```

Block preconditioners: PCD II

```
[blockns.pmm]
# consider diagonal pressure mass matrix
diag=0

[blockns.pcd]
# CL at inflow of pressure
inflow=Robin#Dirichlet#Robin
# CL at outflow of pressure
outflow=Dirichlet#Neumann
# Mp Fp^-1 Ap : 1, other Ap Fp^-1 Mp
order=1
#diffusion=BTBt#Laplacian or BTBt

# Ap : diffusion operator
[Ap]
```

Block preconditioners: PCD III

```
#reuse-prec=1
ksp-type=preonly
pc-type=lu
pc-factor-mat-solver-package-type=mumps
pc-factor-mumps.icntl-14=60

# velocity bloc
[Fu]
ksp-type=preonly
pc-type=lu
pc-factor-mat-solver-package-type=mumps
pc-factor-mumps.icntl-14=60

# pressure mass matrix
[Mp]
```

Block preconditioners: PCD IV

```
#reuse-prec=1
ksp-type=preonly
pc-type=lu
pc-factor-mat-solver-package-type=mumps
pc-factor-mumps.icntl-14=60
```

Block preconditioner: SIMPLE I

```
[fluid]
ksp-type=fgmres#gcr#fgmres
fgmres-restart=100

# fieldsplit
pc-type=fieldsplit
fieldsplit-type=schur #additive, multiplicative, schur
fieldsplit-schur-fact-type=upper#full#upper#full #diag, low

fieldsplit-schur-inner-solver.use-outer-solver=0
fieldsplit-schur-inner-solver.pc-type=jacobi
fieldsplit-schur-inner-solver.ksp-type=preonly
fieldsplit-schur-upper-solver.use-outer-solver=0
fieldsplit-schur-upper-solver.pc-type=jacobi
```

Block preconditioner: SIMPLE II

```
fieldsplit-schur-upper-solver.ksp-type=preonly

# Schur complement
fieldsplit-schur-precondition=user

# block velocity
[fluid.fieldsplit-0]
ksp-type=preonly
pc-type=lu
pc-factor-mat-solver-package-type=mumps
pc-factor-mumps.icntl-14=60

[fluid.fieldsplit-1]
ksp-type=preonly#gmres
ksp-maxit=15
```

Block preconditioner: SIMPLE III

```
ksp-rtol=1e-3
pc-type=lu
pc-factor-mat-solver-package-type=mumps
pc-factor-mumps.icntl-14=60
```

Part III

Solving PDEs with Feel++

Exercising with the Toolboxes

Dropbox Link for the exercises without the solution (but with the geometry or the mesh) `https://www.dropbox.com/sh/ce1xlktwe6sptz3/AACi3uzIoiPkfQuP98GM_q0ca?dl=0`

Toolbox Framework

CFD Toolbox

Computation Solid Mechanics toolbox

Fluid Structure Interaction toolbox

Heat Transfer toolbox

Thermo Electric toolbox

Model Properties: Why?

We have a nice DSEL that allows to write variational formulations, we have the cfg files as well as Ginac. However everyone comes up with a different solution, we need to have a more unified way to describe our models.

Introduce Model Properties Framework

Model Properties: What?

What are model properties?

- Parameters: fixed, variables, formulas
- Materials
- Model variants
- Boundary Conditions
- Post Processing

and possibly more ...

Model Properties: JSON

From `json.org`: “json is lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript”

```
// -*- mode: javascript -*-
{
    "Name": "Turek",
    "Description":"Fluid flow around a cylinder",
    "Model":"Navier_Stokes",
    "Parameters": {}.
    "Materials": {},
    "BoundaryConditions": {},
    "PostProcess":{}
}
```

- Use `boost::property_tree`

Model Properties: Parameters

Parameters allow to define constants, variables and formulas

```
"Parameters": {
  "Um":{
     "type":"variable",
     "value":0.3,
     "min":1e-4,
     "max":10
   },
   "H": {
     "type":"constant",
     "value":0.41
   }}
```

- can be used in boundary conditions, material properties or in formulas
- we should be able to define different kind of parameters and customize them (e.g. random variables...)

Model Properties: Materials

Materials allow to define area with specific properties

```
"Materials":{
  "fluid":{
    // density
    "rho": "1",
    // dynamic_viscosity
    "mu": "0.001"
  }},
```

- Currently support coefficients required by fluid and thermal flows with default values
- the name of the material is associated directly to a physical region in Gmsh

Model Properties: Boundary Conditions

See `feel/feelpde/boundaryconditions.hpp`

```
"BoundaryConditions":{
  "velocity": {
    "Dirichlet": {
      "inflow": {
        "expr":"{4*Um*y*( H-y )/H^2,0}:y:Um:H"
      },
      "wall": {
        "expr":"{0,0}"
      },
      "cylinder":{
        "expr":"{0,0}"
      } },
    "Neumann": {
      "outlet": {
        "expr":"{0,0,0,0}"
      }
    }}}, // BoundaryConditions
```

- Support 1 or two expressions (Dirichlet, Neumann, Robin)
- Support create of scalar, vector or matrix fields

```
// retrieve a list of all vector field
//associated to velocity and Dirichlet
dirichlet_cond = bc.getVectorFields<di
        ( "velocity", "Dirichlet" );

// retrieve a list of all matrix field
// associated to velocity and neumann
neumann_cond = bc.getMatrixFields<dim>
        ( "velocity", "Neumann" );
```

Model Properties: Post Process

Define outputs for the model to be computed

```
"PostProcess":
{
  "Force":["cylinder"],
  "PressureDifference":
  {
    "x1":"{0.15,0.2}",
    "x2":"{0.25,0.2}"
  }}
```

It is up to the model to decide the type of postprocessing it supports

- Compute drag and lift (Force applied to the structure)
- Compute fields at points
- Compute flow rates
- ...

Toolbox Framework

CFD Toolbox

Computation Solid Mechanics toolbox

Fluid Structure Interaction toolbox

Heat Transfer toolbox

Thermo Electric toolbox

CFD Toolbox

https://book.feelpp.org/content/en/03-modeling/Fluid/

CFD - TurekHron I

Run interactive feel++ docker container

```
docker run -it -v $HOME/feel:/feel \
       feelpp/feelpp/feelpp-toolboxes
```

You are now in the FEEL++ toolbox docker container.

Place yourself to CFD TurekHron test case directory then execute the CFD toolbox with given configs

```
cd Testcases/CFD/TurekHron/

mpirun -np 4 /usr/local/bin/feelpp_toolbox_fluid_2d \
       --config-file cfd1.cfg
```

Take a look to the toolbox config **./cfd1.cfg** and model config **cfd1.json**

For more possible options

CFD - TurekHron II

```
feelpp_toolbox_fluid_2d --help
```

CFD - TurekHron 1 - cfd1.cfg I

Config file **~/Testcases/CFD/TurekHron/cfd1.cfg**

```
#fe-approximation=P1P1

[fluid]
geofile=$cfgdir/cfd.geo
[fluid.gmsh]
hsize=0.03

[fluid]
filename=$cfgdir/cfd1.json

solver=Newton #Picard,Newton
#verbose_solvertimer=1

linearsystem-cst-update=false
jacobian-linear-update=false
```

CFD - TurekHron 1 - cfd1.cfg II

```
#start-by-solve-stokes-stationary=1
#reuse-prec=true
#reuse-jac=true
#reuse-jac.rebuild-at-first-newton-step=true
#reuse-prec.rebuild-at-first-newton-step=true

# ksp-converged-reason=true
# snes-converged-reason=true
snes-monitor=true
# ksp-monitor=true
snes-maxit=100
snes-maxit-reuse=100
snes-ksp-maxit=1000
snes-ksp-maxit-reuse=100

pc-type=lu #gasm,lu,fieldsplit,ilu

[exporter]
directory=applications/models/fluid/TurekHron/cfd1/$fluid_tag
```

CFD - TurekHron 1 - .json I

Model config file **~/Testcases/CFD/TurekHron/cfd1.json**

```
{
    "Name": "Fluid Mechanics",
    "ShortName":"Fluid",
    "Model":"Navier-Stokes",
    "Parameters":
    {
        "ubar":"0.2"
    },
    "Materials":
    {
        "Fluid":{
            "name":"myFluidMat",
            "rho":"1.0e3",
            "mu":"1.0"
        }
    },
```

CFD - TurekHron 1 - .json II

```
    "BoundaryConditions":
    {
        "velocity":
        {
            "Dirichlet":
            {
                "inlet":
                {
                    "expr":"{ 1.5*ubar*(4./0.1681)*y*(0.41-y),0
                        }:ubar:y"
                },
                "wall1":
                {
                    "expr":"{0,0}"
                },
                "wall2":
                {
                    "expr":"{0,0}"
                }
            }
```

CFD - TurekHron 1 - .json III

```
        },
        "fluid":
        {
            "outlet":
            {
                "outlet":
                {
                    "expr":"0"
                }
            }
        }
    },
    "PostProcess":
    {
        "Fields":["velocity","pressure","pid"],
        "Measures":
        {
            "Forces":"wall2",
            "Points":
            {
```

CFD - TurekHron 1 - .json IV

```
                    "pointA":
                    {
                        "coord":"{0.6,0.2,0}",
                        "fields":"pressure"
                    },
                    "pointB":
                    {
                        "coord":"{0.15,0.2,0}",
                        "fields":"pressure"
                    }
                }
            }
        }

}
```

CFD - TurekHron 2 - cfd2.cfg I

Config file **~/Testcases/CFD/TurekHron/cfd2.cfg**

```
#fe-approximation=P1P1

[fluid]
geofile=$cfgdir/cfd.geo
[fluid.gmsh]
hsize=0.03

[fluid]
filename=$cfgdir/cfd2.json

solver=Newton #Picard,Newton
#verbose_solvertimer=1

linearsystem-cst-update=false
jacobian-linear-update=false
```

CFD - TurekHron 2 - cfd2.cfg II

```
#start-by-solve-stokes-stationary=1
#reuse-prec=true
#reuse-jac=true
#reuse-jac.rebuild-at-first-newton-step=true
#reuse-prec.rebuild-at-first-newton-step=true

# ksp-converged-reason=true
# snes-converged-reason=true
snes-monitor=true
# ksp-monitor=true
snes-maxit=100
snes-maxit-reuse=100
snes-ksp-maxit=1000
snes-ksp-maxit-reuse=100

pc-type=lu #gasm,lu,fieldsplit,ilu

[exporter]
directory=applications/models/fluid/TurekHron/cfd2/$fluid_tag
```

CFD - TurekHron 2 - .json I

Model config file **~/Testcases/CFD/TurekHron/cfd2.json**

```
{
    "Name": "Fluid Mechanics",
    "ShortName":"Fluid",
    "Model":"Navier-Stokes",
    "Parameters":
    {
        "ubar":"1.0"
    },
    "Materials":
    {
        "Fluid":{
            "name":"myFluidMat",
            "rho":"1.0e3",
            "mu":"1.0"
        }
    },
```

CFD - TurekHron 2 - .json II

```
    "BoundaryConditions":
    {
        "velocity":
        {
            "Dirichlet":
            {
                "inlet":
                {
                    "expr":"{ 1.5*ubar*(4./0.1681)*y*(0.41-y),0
                        }:ubar:y"
                },
                "wall1":
                {
                    "expr":"{0,0}"
                },
                "wall2":
                {
                    "expr":"{0,0}"
                }
            }
```

CFD - TurekHron 2 - .json III

```
        },
        "fluid":
        {
            "outlet":
            {
                "outlet":
                {
                    "expr":"0"
                }
            }
        }
    },
    "PostProcess":
    {
        "Fields":["velocity","pressure","pid"],
        "Measures":
        {
            "Forces":"wall2",
            "Points":
            {
```

CFD - TurekHron 2 - .json IV

```
                "pointA":
                {
                    "coord":"{0.6,0.2,0}",
                    "fields":"pressure"
                },
                "pointB":
                {
                    "coord":"{0.15,0.2,0}",
                    "fields":"pressure"
                }
            }
        }
    }

}
```

CFD - TurekHron 3 - cfd3.cfg I

Config file **~/Testcases/CFD/TurekHron/cfd3.cfg**

```
#fe-approximation=P1P1


[fluid]
geofile=$cfgdir/cfd.geo
[fluid.gmsh]
hsize=0.03#0.02

[fluid]
filename=$cfgdir/cfd3.json

solver=Newton #Oseen,Picard,Newton

linearsystem-cst-update=false
jacobian-linear-update=false
```

CFD - TurekHron 3 - cfd3.cfg II

```
#start-by-solve-stokes-stationary=1
# reuse-prec=true
# ksp-maxit-reuse=5
# reuse-jac=true
# reuse-jac.rebuild-at-first-newton-step=true
# reuse-prec.rebuild-at-first-newton-step=true

# ksp-converged-reason=true
# snes-converged-reason=true
# snes-monitor=true
# snes-maxit=100
# snes-maxit-reuse=10
# snes-ksp-maxit=1000
# snes-ksp-maxit-reuse=10

pc-type=lu #gasm,lu,fieldsplit,ilu

#verbose_solvertimer=1
# marked-zones.internal-faces=1
# #marked-zones.expressions=1
# # stabilisation-cip-convection=1
```

CFD - TurekHron 3 - cfd3.cfg III

```
# stabilisation-cip-pressure=1
# # stabilisation-cip-divergence=1


[fluid.bdf]
order=2
#strategy-high-order-start=1

[ts]
time-step=0.01
time-final=10
#restart=true
restart.at-last-save=true
#time-initial=0.0002
#save.freq=2

[exporter]
directory=applications/models/fluid/TurekHron/cfd3/$fluid_tag
```

CFD - TurekHron 3 - .json I

Model config file **~/Testcases/CFD/TurekHron/cfd3.json**

```
// -*- mode: javascript -*-
{
    "Name": "Fluid Mechanics",
    "ShortName":"Fluid",
    "Model":"Navier-Stokes",
    "Parameters":
    {
        "ubar":"2",
        "chi":"t<2:t"
    },
    "Materials":
    {
        "Fluid":{
            "name":"myFluidMat",
            "rho":"1.0e3",
            "mu":"1.0"
```

CFD - TurekHron 3 - .json II

```
        }
    },
    "BoundaryConditions":
    {
        "velocity":
        {
            "Dirichlet":
            {
                "inlet":
                {
                    "expr":"{ 1.5*ubar*(4./0.1681)*y*(0.41-y)*(
                        ((1-cos(pi*t/2))/2)*chi + (1-chi) ),0}:
                       ubar:y:t:chi"
                },
                "wall1":
                {
                    "expr":"{0,0}"
                },
                "wall2":
                {
```

CFD - TurekHron 3 - .json III

```
                    "expr":"{0,0}"
                }
            }
        },
        "fluid":
        {
            "outlet":
            {
                "outlet":
                {
                    "expr":"0"
                }
            }
        }
    },
    "PostProcess":
    {
        "Fields":["velocity","pressure","pid"],
        "Measures":
        {
```

CFD - TurekHron 3 - .json IV

```
            "Forces":"wall2",
            "Points":
            {
                "pointA":
                {
                    "coord":"{0.6,0.2,0}",
                    "fields":"pressure"
                },
                "pointB":
                {
                    "coord":"{0.15,0.2,0}",
                    "fields":"pressure"
                }
            }
        }
    }

}
```

CFD - Cavity I

Run interactive feel++ docker container

```
docker run -it -v $HOME/feel:/feel \
       feelpp/feelpp/feelpp-toolboxes
```

You are now in the FEEL++ toolbox docker container.

Place yourself to CFD Lid-driven-cavity test case directory then execute the CFD toolbox with given configs

```
cd Testcases/CFD/Lid-driven-cavity/

mpirun -np 4 /usr/local/bin/feelpp_toolbox_fluid_2d \
       --config-file cavity2d.cfg
```

CFD - Cavity II

Take a look to the toolbox config **./cavity2d.cfg** and model config **cavity2d.json**

For more possible options

```
feelpp_toolbox_fluid_2d --help
```

You can test also the 3d case.

Toolbox Framework

CFD Toolbox

Computation Solid Mechanics toolbox

Fluid Structure Interaction toolbox

Heat Transfer toolbox

Thermo Electric toolbox

CSM Toolbox

https://book.feelpp.org/content/en/03-modeling/Solid/

CSM - TurekHron I

Run interactive feel++ docker container

```
docker run -it -v $HOME/feel:/feel \
       feelpp/feelpp/feelpp-toolboxes
```

You are now in the FEEL++ toolbox docker container.

Place yourself to CSM TurekHron test case directory then execute the solid toolbox with given configs

```
cd Testcases/CSM/TurekHron/

mpirun -np 4 /usr/local/bin/feelpp_toolbox_solid_2d \
       --config-file csm3.cfg
```

Take a look to the toolbox config **./csm3.cfg** and model config **csm3.json**

For more possible options

CSM - TurekHron II

```
feelpp_toolbox_solid_2d --help
```

CSM - TurekHron - csm3.cfg I

Config file **~/Testcases/solid/TurekHron/csm3.cfg**

```
fe-approximation=P1 #P1,P2

[solid]
filename=$toolboxes_srcdir/solid/TurekHron/csm3.json

material_law=StVenantKirchhoff# StVenantKirchhoff, NeoHookean

# use density and material coeff cst in appli
jacobian-linear-update=false
linearsystem-cst-update=false

# snes and ksp config
#reuse-prec=true#false
#reuse-jac=true#false
reuse-jac.rebuild-at-first-newton-step=true
reuse-prec.rebuild-at-first-newton-step=true
```

CSM - TurekHron - csm3.cfg II

```
snes-maxit=500
snes-maxit-reuse=10
snes-ksp-maxit=1000
snes-ksp-maxit-reuse=100

# precondtioner config
pc-type=lu #lu,gasm,ml
ksp-converged-reason=1
geofile=$toolboxes_srcdir/solid/TurekHron/csm.geo


[solid.gmsh]
hsize=0.004

[ts]
time-step=0.01
time-final=10.01
#restart=true
restart.at-last-save=true
#save.freq=2
```

CSM - TurekHron - csm3.cfg III

```
[exporter]
directory=applications/models/solid/TurekHron/csm3/$solid_tag
freq=1
#format=ensightgold
```

CSM - TurekHron - .json I

Model config file **~/Testcases/solid/TurekHron/csm3.json**

```
// -*- mode: javascript -*-
{
    "Name": "Solid Mechanics ",
    "ShortName":"Solid",
    "Model":"Hyper-Elasticity",
    "Parameters":
    {
        "gravity":
        {
            "value":"2"
        }
    },
    "Materials":
    {
        "beam":{
            "name":"solid",
```

CSM - TurekHron - .json II

```
            "E":"1.4e6",
            "nu":"0.4",
            "rho":"1e3"
        }
    },
    "BoundaryConditions":
    {
        "displacement":
        {
            "Dirichlet":
            {
                "fixed-wall":
                {
                    "expr":"{0,0}"
                }
            },
            "Neumann_scalar":
            {
                "free-wall":
                {
```

CSM - TurekHron - .json III

```
                    "expr":"0"
                }
            },
            "VolumicForces":
            {
                "":
                {
                    "expr":"{0,-gravity*1e3}:gravity"
                }
            }
        }
    },
    "PostProcess":
    {
        "Fields":["displacement"],
        "Measures":
        {
            "Points":
            {
                "pointA":
```

CSM - TurekHron - .json IV

```
                    {
                        "coord":"{0.6,0.2,0}",
                        "fields":["displacement","velocity"]
                    }
                },
                "Maximum":
                {
                    "free-wall":
                    {
                        "markers":"free-wall",
                        "fields":["displacement","velocity"]
                    }
                }
            }
        }

}
```

CSM - TorsionBar I

Run interactive feel++ docker container

```
docker run -it -v $HOME/feel:/feel \
       feelpp/feelpp/feelpp-toolboxes
```

You are now in the FEEL++ toolbox docker container.

Place yourself to CSM TurekHron test case directory then execute the solid toolbox with given configs

```
cd Testcases/CSM/torsionbar

mpirun -np 4 /usr/local/bin/feelpp_toolbox_stress_2d \
       --config-file torsionbar.cfg
```

CSM - TorsionBar II

Take a look to the toolbox config **./torsionbar.cfg** and model config **torsionbar.json**

For more possible options

```
feelpp_toolbox_stress_2d --help
```

CSM - TorsionBar - .cfg I

Config file **~/Testcases/CSM/torsionbar/torsionbar.cfg**

```
fe-approximation=P1 #P1,P2

solve-quasi-static=1
[solve-quasi-static]
variable-step=0.01#0.025#0.05
variable-symbol=rotation

[solid]
geofile=$cfgdir/torsionbar.geo
[solid.gmsh]
hsize=0.08#0.1


[solid]
filename=$cfgdir/torsionbar.json
```

CSM - TorsionBar - .cfg II

```
material_law=NeoHookean# StVenantKirchhoff, NeoHookean
#use-incompressibility-constraint=1

#verbose=1
verbose_solvertimer=1

on.type=elimination_symmetric

# use density and material coeff cst in appli
jacobian-linear-update=false
linearsystem-cst-update=false

# snes and ksp config
#reuse-prec=true#false
reuse-prec.rebuild-at-first-newton-step=true
#reuse-jac=false
#reuse-jac.rebuild-at-first-newton-step=true

ksp-converged-reason=1
#ksp-monitor=1
```

CSM - TorsionBar - .cfg III

```
snes-monitor=1

snes-maxit=500
snes-maxit-reuse=50#10
snes-ksp-maxit=1000
snes-ksp-maxit-reuse=100

# preconditioner config
pc-type=gasm


[ts]
#restart=true


[exporter]
directory=applications/models/solid/torsionbar/$solid_tag
```

CSM - TorsionBar - .json I

Model config file **~/Testcases/CSM/torsionbar/torsionbar.json**

```
// -*- mode: javascript -*-
{
    "Name": "Solid Mechanics ",
    "ShortName":"Solid",
    "Model":"Hyper-Elasticity",
    "Materials":
    {
        "OmegaSolid":{
            "name":"copper",
            //"E":"1.4e6",
            "E":"124e6",
            "nu":"0.33",
            "rho":"8920"
        }
    },
    "Parameters":
```

CSM - TorsionBar - .json II

```
    {
        "rotation":
        {
            //"value":"8"
            "value":"3"
        }
    },
    "BoundaryConditions":
    {
        "displacement_y":
        {
            "Dirichlet":
            {
                "Torsion":
                {
                    "expr":"0.5 + (y - 0.5)*cos(rotation) - (z-
                       0.5)*sin(rotation) - y :y:z:rotation"
                }
            }
        },
```

CSM - TorsionBar - .json III

```
        "displacement_z":
        {
            "Dirichlet":
            {
                "Torsion":
                {
                    "expr":"0.5 + (y - 0.5)*sin(rotation) + (z-
                       0.5)*cos(rotation) - z :y:z:rotation"
                }
            }
        },
        "displacement" :
        {
            "Dirichlet":
            {
                "Fixed":
                {
                    "expr":"{0,0,0}"
                }
            },
```

CSM - TorsionBar - .json IV

```
            "Neumann_scalar" :
            {
                "BoundaryForce":
                {
                    "expr":"0"
                }
            }
        }
    },
    "PostProcess":
    {
        "Fields":["displacement","pressure","pid","Von-Mises"],
        "Measures":
        {
            "VolumeVariation":""
        }

    }
}
```

CSM - Solenoid I

Run interactive feel++ docker container

```
docker run -it -v $HOME/feel:/feel \
       feelpp/feelpp/feelpp-toolboxes
```

You are now in the FEEL++ toolbox docker container.

Place yourself to CSM Solenoid test case directory then execute the solid toolbox with given configs

```
cd Testcases/CSM/Solenoid

mpirun -np 4 /usr/local/bin/feelpp_toolbox_solenoid_3d \
       --config-file sol.cfg
```

Take a look to the toolbox config **./sol.cfg** and model config **sol.json**

For more possible options

CSM - Solenoid II

```
feelpp_toolbox_solenoid_3d --help
```

CSM - Solenoid - .cfg I

Config file **~/Testcases/CSM/Solenoid/sol.cfg**

```
[solid]
filename=$cfgdir/sol.json

on.type=elimination_symmetric

# # precondtioner config
pc-type=gamg #lu,gasm,ml
ksp-monitor=1
# ksp-converged-reason=1

geofile=$cfgdir/sol.geo
[solid.gmsh]
hsize=0.1

[ts]
```

CSM - Solenoid - .cfg II

```
steady=true

[exporter]
directory=applications/models/solid/Solenoid/sol/$solid_tag
```

CSM - Solenoid - .json I

Model config file **~/Testcases/CSM/Solenoid/sol.json**

```
{
    "Name": "Solid Mechanics ",
    "ShortName":"Solid",
    "Model":"Elasticity",
    "Materials":
    {
        "Omega":
        {
            "name":"solid",
            "E":"2.1e6",
            "nu":"0.33"
        }
    },
    "BoundaryConditions":
    {
        "displacement":
```

CSM - Solenoid - .json II

```
        {
            "Dirichlet":
            {
                "Top":
                {
                    "expr":"{0,0,0}"
                },
                "Bottom":
                {
                    "expr":"{0,0,0}"
                }
            },
            "Neumann_scalar":
            {
                "Rint":
                {
                    "expr":"0"
                },
                "Rext":
                {
```

CSM - Solenoid - .json III

```
                    "expr":"0"
                }
            },
            "VolumicForces":
            {
                "":
                {
                    "expr":"{x/sqrt(x^2+y^2)*5000*(10+20*(sqrt(
                       x^2+y^2)-1)),y/sqrt(x^2+y^2)*5000*(10+20
                       *(sqrt(x^2+y^2)-1)),0.0}:x:y:z"
                }
            }
        }
    }
}
```

CSM - NAFEMS-LE1 I

Run interactive feel++ docker container

```
docker run -it -v $HOME/feel:/feel \
       feelpp/feelpp/feelpp-toolboxes
```

You are now in the FEEL++ toolbox docker container.

Place yourself to CSM NAFEMS-LE1 test case directory then execute the solid toolbox with given configs

```
cd Testcases/CSM/NAFEMS-LE1

mpirun -np 4 /usr/local/bin/feelpp_toolbox_stress_2d \
       --config-file le1.cfg
```

Take a look to the toolbox config **./le1.cfg** and model config **le1.json**

For more possible options

CSM - NAFEMS-LE1 II

```
feelpp_toolbox_stress_2d --help
```

CSM - NAFEMS-LE1 - .cfg I

Config file **~/Testcases/CSM/NAFEMS-LE1/le1.cfg**

```
fe-approximation=P1 #P1,P2

[solid]
filename=$cfgdir/le1.json

on.type=elimination_symmetric

# # precondtioner config
pc-type=gamg #lu,gasm,ml
ksp-monitor=1
# ksp-converged-reason=1

geofile=$cfgdir/le1.geo
[solid.gmsh]
hsize=0.05
```

CSM - NAFEMS-LE1 - .cfg II

```
[ts]
steady=true

[exporter]
directory=applications/models/solid/NAFEMS-LE1/le1/$solid_tag
```

CSM - NAFEMS-LE1 - .json I

Model config file **~/Testcases/CSM/NAFEMS-LE1/le1.json**

```
{
    "Name": "Solid Mechanics ",
    "ShortName":"Solid",
    "Model":"Elasticity",
    "Materials":
    {
        "Omega":
        {
            "name":"solid",
            "E":"210e9",
            "nu":"0.3",
            "rho":"7800"
        }
    },
    "BoundaryConditions":
    {
        "displacement_x":
```

CSM - NAFEMS-LE1 - .json II

```
        {
            "Dirichlet":
            {
                "AB":
                {
                    "expr":"0"
                }
            }
        },
        "displacement_y":
        {
            "Dirichlet":
            {
                "CD":
                {
                    "expr":"0"
                }
            }
        },
        "displacement":
```

CSM - NAFEMS-LE1 - .json III

```
        {
            "Neumann_scalar":
            {
                "BC":
                {
                    "expr":"1e7"
                }
            }
        }
    },
    "PostProcess":
    {
        "Fields":["displacement","Von-Mises","tresca","
           principal-stresses"],
        "Measures":
        {
            "Points":
            {
                "pointD":
                {
```

CSM - NAFEMS-LE1 - .json IV

```
                    "coord":"{2,0,0}",
                    "fields":["stress_yy"]
                }
            }
        }
    }

}
```

CSM - NAFEMS-LE10 I

Run interactive feel++ docker container

```
docker run -it -v $HOME/feel:/feel \
       feelpp/feelpp/feelpp-toolboxes
```

You are now in the FEEL++ toolbox docker container.

Place yourself to CSM NAFEMS-LE10 test case directory then execute the solid toolbox with given configs

```
cd Testcases/CSM/NAFEMS-LE10

mpirun -np 4 /usr/local/bin/feelpp_toolbox_stress_2d \
       --config-file le10.cfg
```

Take a look to the toolbox config **./le10.cfg** and model config **le10.json**

For more possible options

CSM - NAFEMS-LE10 II

```
feelpp_toolbox_stress_2d --help
```

CSM - NAFEMS-LE10 - .cfg I

Config file ~/Testcases/CSM/NAFEMS-LE10/le10.cfg

```
fe-approximation=P1 #P1,P2

[solid]
filename=$cfgdir/le10.json

on.type=elimination_symmetric

# # precondtioner config
pc-type=gamg #lu,gasm,ml
ksp-monitor=1
# ksp-converged-reason=1

geofile=$cfgdir/le10.geo
[solid.gmsh]
hsize=0.06

[ts]
```

CSM - NAFEMS-LE10 - .cfg II

```
steady=true

[exporter]
directory=applications/models/solid/NAFEMS-LE10/le10/$solid_tag
```

CSM - NAFEMS-LE10 - .json I

Model config file **~/Testcases/CSM/NAFEMS-LE10/le10.json**

```
{
    "Name": "Solid Mechanics ",
    "ShortName":"Solid",
    "Model":"Elasticity",
    "Materials":
    {
        "Omega":
        {
            "name":"solid",
            "E":"210e9",
            "nu":"0.3",
            "rho":"7800"
        }
    },
    "BoundaryConditions":
    {
        "displacement_x":
```

CSM - NAFEMS-LE10 - .json II

```
        {
            "Dirichlet":
            {
                "ABAB":
                {
                    "expr":"0"
                },
                "BCBC":
                {
                    "expr":"0"
                }
            }
        },
        "displacement_y":
        {
            "Dirichlet":
            {
                "DCDC":
                {
                    "expr":"0"
```

CSM - NAFEMS-LE10 - .json III

```
                },
                "BCBC":
                {
                    "expr":"0"
                }
            }
        },
        "displacement_z":
        {
            "Dirichlet":
            {
                "EE":
                {
                    "expr":"0"
                }
            }
        },
        "displacement":
        {
            "Neumann_scalar":
```

CSM - NAFEMS-LE10 - .json IV

```
            {
                "Top":
                {
                    "expr":"-1e6"
                }
            }
        }
    },
    "PostProcess":
    {
        "Fields":["displacement","Von-Mises","tresca","
           principal-stresses"],
        "Measures":
        {
            "Points":
            {
                "pointD":
                {
                    "coord":"{2,0,0.6}",
                    "fields":["stress_yy"]
```

CSM - NAFEMS-LE10 - .json V

```
                    }
                }
            }
        }

}
```

Toolbox Framework

CFD Toolbox

Computation Solid Mechanics toolbox

Fluid Structure Interaction toolbox

Heat Transfer toolbox

Thermo Electric toolbox

FSIB Toolbox

https://book.feelpp.org/content/en/03-modeling/FSI/

FSI - TurekHron I

Run interactive feel++ docker container

```
docker run -it -v $HOME/feel:/feel \
       feelpp/feelpp/feelpp-toolboxes
```

You are now in the FEEL++ toolbox docker container.

Place yourself to FSI TurekHron test case directory then execute the FSI toolbox with given configs

```
cd Testcases/FSI/TurekHron/

mpirun -np 4 /usr/local/bin/feelpp_toolbox_fsi_2d \
       --config-file fsi3.cfg
```

FSI - TurekHron II

Take a look to the toolbox config **`./fsi3.cfg`** and model config **`fsi3_solid.json`** and **`fsi3_fluid.json`**

For more possible options

```
feelpp_toolbox_fsi_2d --help
```

FSI - TurekHron - fsi3.cfg I

Config file **~/Testcases/FSI/TurekHron/fsi3.cfg**

```
#fe-approximation=P1P1

[fsi]
geofile=$toolboxes_srcdir/fsi/TurekHron/fsi.geo
fluid-mesh.markers=Fluid
solid-mesh.markers=Solid
hsize=0.05 #  M1=0.05; M2=0.025; M3=0.0125
mesh-save.tag=M1

conforming-interface=true

fixpoint.tol=1e-6#1e-8
fixpoint.initialtheta=0.98#0.1#99#0.05
fixpoint.min_theta=1e-12#0.005#1e-8#1e-4
#fixpoint.maxit=10#13#100#3#1
```

FSI - TurekHron - fsi3.cfg II

```
coupling-type=Semi-Implicit #Semi-Implicit

coupling-bc=dirichlet-neumann
#coupling-bc=nitsche
#coupling-bc=robin-robin
#coupling-bc=robin-neumann
#coupling-bc=robin-robin-genuine
coupling-nitsche-family.gamma=600
#coupling-nitsche-family.gamma0=0.5
#coupling-bc=robin-neumann-generalized

# optimisations
solid.reuse-prec.activated-after-n-fsi-it=2
solid.reuse-prec.activated-only-if-greater-than-tol=0.5#

[fluid]
filename=$toolboxes_srcdir/fsi/TurekHron/fsi3_fluid.json

solver=Oseen#Newton #Oseen,Picard,Newton
```

FSI - TurekHron - fsi3.cfg III

```
#hovisu=true

reuse-prec=true
reuse-prec.rebuild-at-first-newton-step=true
reuse-jac=true
reuse-jac.rebuild-at-first-newton-step=true

#ksp-converged-reason=true
#snes-converged-reason=true
pc-type=lu#gasm #gasm#lu #asm#fieldsplit #ilu
#ksp-maxit=30
#ksp-maxit-reuse=10

[fluid.alemesh]
type=harmonic
pc-type=lu
ksp-maxit=30
reuse-prec=true
[fluid.alemesh.ho]
pc-type=lu
ksp-maxit=30
```

FSI - TurekHron - fsi3.cfg IV

```
reuse-prec=true
[fluid.bdf]
order=2
[fluid.alemesh.bdf]
order=2

[solid]
filename=$toolboxes_srcdir/fsi/TurekHron/fsi3_solid.json

material_law=StVenantKirchhoff # StVenantKirchhoff, NeoHookean

pc-type=lu

# reuse prec/jac config
reuse-prec=true
reuse-prec.rebuild-at-first-newton-step=true
snes-maxit-reuse=10
#snes-ksp-maxit=1000
snes-ksp-maxit-reuse=10
#ksp-maxit=10
#ksp-converged-reason=true
```

FSI - TurekHron - fsi3.cfg V

```
[ts]
#restart=true
time-step=0.005#0.01 #0.01,0.005,0.001
time-final=10.0
restart.at-last-save=true
#time-initial=3.01 #3.04

[exporter]
directory=applications/models/fsi/TurekHron/fsi3/P2P1G1-P1G1

[fluid]
# verbose=1
# verbose_solvertimer=1
[solid]
# verbose=1
# verbose_solvertimer=1
[fsi]
# verbose=1
```

FSI - TurekHron - fsi3.cfg VI

```
verbose_solvertimer=1
```

FSI - TurekHron - .json I

Model config file **~/Testcases/FSI/TurekHron/fsi3_solid.json**

```
// -*- mode: javascript -*-
{
    "Name": "2D beam ",
    "ShortName":"beam2d",
    "Model":"Hyper-Elasticity",
    "Materials":
    {
        "Solid": {
            "name":"solidFSI3",
            "E":"5.6e6",
            "nu":"0.4",
            "rho":"1e3"
        }
    },
    "BoundaryConditions":
    {
```

FSI - TurekHron - .json II

```
        "displacement":
        {
            "Dirichlet":
            {
                "solid-fixed":
                {
                    "expr":"{0,0}"
                }
            },
            "interface_fsi":
            {
                "fsi-wall":
                {
                    "expr":"0"
                }
            }
        }
    },
    "PostProcess":
    {
```

FSI - TurekHron - .json III

```
        "Fields":["displacement","velocity","pid"],
        "Measures":
        {
            "Points":
            {
                "pointA":
                {
                    "coord":"{0.6,0.2,0}",
                    "fields":["displacement","velocity"]
                }
            }
        }

    }

}
```

FSI - TurekHron - .json I

Model config file **~/Testcases/FSI/TurekHron/fsi3_fluid.json**

```
// -*- mode: javascript -*-
{
    "Name": "Fluid fsi",
    "ShortName":"FluidFSI",
    "Model":"Navier-Stokes",
    "Parameters":
    {
        "ubar":"2",
        "chi":"t<2:t"
    },
    "Materials":
    {
        //"Fluid":{ "name":"Glycerine","filename":"
            $feelpp_srcdir/databases/materials/pure/copper.json"
             }
        "Fluid": {
```

FSI - TurekHron - .json II

```
            "name":"fluidFSI3",
            "rho":"1.0e3",
            "mu":"1.0"
        }
    },
    "BoundaryConditions":
    {
        "velocity":
        {
            "Dirichlet":
            {
                "fluid-inlet":
                {
                    "expr":"{ 1.5*ubar*(4./0.1681)*y*(0.41-y)*(
                        ((1-cos(pi*t/2))/2)*chi + (1-chi) ),0}:
                       ubar:y:t:chi"
                },
                "fluid-wall":
                {
                    "expr":"{0,0}"
```

FSI - TurekHron - .json III

```
            },
            "fluid-cylinder":
            {
                "expr":"{0,0}"
            }
        },
        "interface_fsi":
        {
            "fsi-wall":
            {
                "expr":"0"
            }
        }
    },
    "fluid":
    {
        "outlet":
        {
            "fluid-outlet":
            {
```

FSI - TurekHron - .json IV

```
                    "expr":"0"
                }
            }
        }
    },
    "PostProcess":
    {
        "Fields":["velocity","pressure","displacement","pid"],
        "Measures":
        {
            "Forces":["fsi-wall","fluid-cylinder"],
            "Points":
            {
                "pointB":
                {
                    "coord":"{0.15,0.2,0}",
                    "fields":"pressure"
                }
            }
        }
```

```
        }
}
```

FSI - WavePressure 2D I

Run interactive feel++ docker container

```
docker run -it -v $HOME/feel:/feel \
       feelpp/feelpp/feelpp-toolboxes
```

You are now in the FEEL++ toolbox docker container.

Place yourself to FSI WavePressure test case directory then execute the FSI toolbox with given configs

```
cd Testcases/FSI/wavepressure2d/

mpirun -np 4 /usr/local/bin/feelpp_toolbox_fsi_2d \
       --config-file fsi3.cfg
```

FSI - WavePressure 2D II

Take a look to the toolbox config **./wavepressure2d.cfg** and model config **wavepressure2d_solid.json** and **wavepressure2d_fluid.json**

For more possible options

```
feelpp_toolbox_fsi_2d --help
```

FSI - WavePressure 2D - .cfg I

Config file **~/Testcases/FSI/wavepressure2d/wavepressure2d.cfg**

Config file **~/Testcases/FSI/wavepressure2d/wavepressure2d.json**

```
#fe-approximation=P1P1

[fsi]
fixpoint.tol=1e-6#1e-8
fixpoint.initialtheta=0.05
conforming-interface=true
coupling-type=Semi-Implicit #Semi-Implicit
#coupling-bc=nitsche
#coupling-bc=robin-neumann
coupling-bc=robin-neumann-generalized
fixpoint.maxit=1#20#1
solid-mesh.extract-1d-from-fluid-mesh=1

[fluid]
```

FSI - WavePressure 2D - .cfg II

```
filename=$toolboxes_srcdir/fsi/wavepressure2d/
    wavepressure2d_fluid.json

gmsh.hsize=0.05#0.1#0.05#0.075#0.05
geofile=$toolboxes_srcdir/fsi/wavepressure2d/
    wavepressure2d_fluid.geo

solver=Oseen #Oseen,Picard,Newton

stabilisation-convection-energy=true
stabilisation-cip-convection=true
marked-zones.expressions=(x<0.3)+(x>(6-0.3)):x

# marked-zones.internal-faces=1
# stabilisation-cip-pressure=1
# stabilisation-cip-pressure-gamma=0.001
# #stabilisation-cip-divergence=1

#hovisu=true
#use-cst-matrix=false
#use-cst-vector=false
```

FSI - WavePressure 2D - .cfg III

```
#ksp-monitor=1
reuse-prec=true
#reuse-jac=true
reuse-jac.rebuild-at-first-newton-step=false
#residual-uselinearjac=true
#ksp-converged-reason=true
#snes-converged-reason=true


#####################################
pc-type=lu #gasm#lu #asm#fieldsplit #ilu
ksp-maxit=30

[fluid.alemesh]
type=harmonic
pc-type=lu
ksp-maxit=30
reuse-prec=true
[fluid.alemesh.ho]
pc-type=lu
ksp-maxit=30
```

FSI - WavePressure 2D - .cfg IV

```
reuse-prec=true
[fluid.bdf]
order=2
[fluid.alemesh.bdf]
order=2

[solid]
filename=$toolboxes_srcdir/fsi/wavepressure2d/
    wavepressure2d_solid.json
rho=1.1
coeffpoisson=0.5
youngmodulus=0.75e6
model=Generalised-String
solver=LinearSystem
pc-type=lu
pc-factor-mat-solver-package-type=mumps #umfpack
ksp-maxit=10
#ksp-converged-reason=true

[ts]
#restart=true
```

FSI - WavePressure 2D - .cfg V

```
#time-initial=0.0044
time-step=0.0001
time-final=1.0
restart.at-last-save=true
file-format=hdf5

[exporter]
#directory=applications/models/fsi/wavepressure2d/P2P1G1-P1G1
directory=applications/models/fsi/wavepressure2d/$fluid_tag-
    $solid_tag


[fluid]
# verbose=1
# verbose_solvertimer=1
[solid]
# verbose=1
# verbose_solvertimer=1
[fsi]
# verbose=1
```

FSI - WavePressure 2D - .cfg VI

```
verbose_solvertimer=1
```

FSI - WavePressure 2D - .json I

Model config file

~/Testcases/FSI/wavepressure2d/wavepressure2d_solid.json

```
// -*- mode: javascript -*-
{
    "Name": "2D beam ",
    "ShortName":"beam2d",
    "Model":"hyperelastic",
    // "Materials":
    // {
    //     "beam":{ "name":"Copper","filename":"$feelpp_srcdir/
        databases/materials/pure/copper.json" }
    // },
    "BoundaryConditions":
    {
        "displacement":
        {
```

FSI - WavePressure 2D - .json II

```
            "Dirichlet":
            {
                "Fixe":
                {
                    "expr":"{0,0}"
                }
            },
            "interface_fsi":
            {
                "fsi-wall":
                {
                    "expr":"0"
                }
            }
        }
    },
    "PostProcess":
    {
        "Fields":["displacement"]
    }
```

FSI - WavePressure 2D - .json III

```
}
```

FSI - WavePressure 2D - .json I

Model config file

`~/Testcases/FSI/wavepressure2d/wavepressure2d_fluid.json`

```
// -*- mode: javascript -*-
{
    "Name": "fluid fsi",
    "ShortName":"Fluidfsi",
    "Model":"Navier-Stokes",
    "Parameters":
    {
        "pIn":
        {
            "value":"2.0e4"
        },
        "pTimeMax":
        {
            "value":"0.005"
        }
```

FSI - WavePressure 2D - .json II

```
    },
    "Materials":
    {
        "Fluid":{
            "name":"myFluidMat",
            "rho":"1",
            "mu":"0.035"
        }
    },
    "BoundaryConditions":
    {
        "velocity":
        {
            "Neumann_scalar":
            {
                "fluid-inlet":
                {
                    "expr": "-(pIn/2.)*(1-cos(pi*t/(pTimeMax/2
                        .)))*(t<pTimeMax):pIn:pTimeMax:t",
```

FSI - WavePressure 2D - .json III

```
                    "alemesh_bc":"fixed"                    // fixed,
                        free [default=fixed]
                }
            },
            "interface_fsi":
            {
                "fsi-wall":{}
            }
        },
        "fluid":
        {
            "outlet":
            {
                "fluid-outlet":
                {
                    "number":1,                             // number
                        of outlet [default=1]
                    "alemesh_bc":"fixed",                   // fixed,
                        free [default=fixed]
```

FSI - WavePressure 2D - .json IV

```
                    "model":"windkessel",                   // free,
                        windkessel [default=free]
                    "windkessel_coupling":"implicit", //
                        explicit, implicit [default=implicit]
                    "windkessel_Rd":6.2e3,                  //
                        resistance distal [default=1.0]
                    "windkessel_Rp":400,                    //
                        resistance proximal [default=1.0]
                    "windkessel_Cd":2.72e-4                 //
                        capacitance [default=1.0]
                }
            }
        }
    },
    "PostProcess":
    {
        "Fields":["velocity","pressure","displacement"]
    }

}
```

FSI - WavePressure 2D - .json V

FSI - WavePressure 3D I

Run interactive feel++ docker container

```
docker run -it -v $HOME/feel:/feel \
       feelpp/feelpp/feelpp-toolboxes
```

You are now in the FEEL++ toolbox docker container.

Place yourself to FSI WavePressure test case directory then execute the FSI toolbox with given configs

```
cd Testcases/FSI/wavepressure3d/

mpirun -np 4 /usr/local/bin/feelpp_toolbox_fsi_2d \
       --config-file fsi3.cfg
```

FSI - WavePressure 3D II

Take a look to the toolbox config **./wavepressure3d.cfg** and model config **wavepressure3d_solid.json** and **wavepressure3d_fluid.json**

For more possible options

```
feelpp_toolbox_fsi_2d --help
```

FSI - WavePressure 3D - .cfg I

Config file **~/Testcases/FSI/wavepressure3d/wavepressure3d.cfg**

```
[fsi]
geofile=$cfgdir/straightpipe.geo
fluid-mesh.markers=Blood
solid-mesh.markers=ArterialWall
hsize=0.1 #M0=0.3 M1=0.1 M2=0.05, M3=0.025
mesh-save.tag=M1
mesh-save.directory=meshfiles/wavepressure3d

fixpoint.maxit=3#10#3#5#1#3
fixpoint.tol=1e-6#1e-8
#fixpoint.initialtheta=0.05
conforming-interface=true
coupling-type=Semi-Implicit
coupling-bc=robin-neumann-generalized
#coupling-bc=robin-neumann-genuine
```

FSI - WavePressure 3D - .cfg II

```
#coupling-bc=robin-neumann#robin-neumann#robin-neumann-genuine
#coupling-bc=robin-robin-genuine#robin-robin
#coupling-bc=nitsche
#coupling-bc=robin-robin
#transfert-velocity-F2S.use-extrapolation=false
coupling-robin-neumann-generalized.use-interface-operator=false
coupling-robin-neumann-generalized.manual-scaling=0.1

[fluid]
filename=$cfgdir/wavepressure3d_fluid.json

solver=Oseen #Oseen,Picard,Newton

stabilisation-convection-energy=true
stabilisation-cip-convection=true

#marked-zones.expressions=(x<0.3)+(x>(6-0.3)):x
# marked-zones.elements-from-markedfaces=inletBlood
# marked-zones.elements-from-markedfaces=outletBlood
marked-zones.markedfaces=inletBlood
```

FSI - WavePressure 3D - .cfg III

```
marked-zones.markedfaces=outletBlood

#fluid-outlet.type=windkessel#free
fluid-outlet.number=1
fluid-outlet.windkessel.coupling=implicit
fluid-outlet.windkessel.Rd0=6.2e3
fluid-outlet.windkessel.Rp0=400
fluid-outlet.windkessel.Cd0=2.72e-4

#hovisu=true
#use-cst-matrix=false
#use-cst-vector=false

reuse-prec=true
#reuse-jac=true
reuse-jac.rebuild-at-first-newton-step=false
#residual-uselinearjac=true
#ksp-converged-reason=true
#snes-converged-reason=true

pc-type=gasm#lu
```

FSI - WavePressure 3D - .cfg IV

```
#ksp-maxit=30

[fluid.alemesh]
type=harmonic
ksp-rtol=1e-8
ksp-type=gmres
#ksp-maxit=30
reuse-prec=true
#ksp-monitor=1
ksp-converged-reason=1
pc-type=fieldsplit
fieldsplit-use-components=1
fieldsplit-type=additive
[fluid.alemesh.fieldsplit-0]
pc-type=gasm#gamg#lu
ksp-type=gmres#preonly
ksp-rtol=1e-3
ksp-maxit=50
#pc-gamg-threshold=0.02
mg-coarse.redundant.pc-factor-mat-solver-package-type=petsc
#pc-gamg-coarse-eq-lim=8000
```

FSI - WavePressure 3D - .cfg V

```
[fluid.alemesh.fieldsplit-1]
pc-type=gasm#gamg#lu
ksp-type=gmres#preonly
ksp-rtol=1e-3
ksp-maxit=50
#pc-gamg-threshold=0.02
mg-coarse.redundant.pc-factor-mat-solver-package-type=petsc
#pc-gamg-coarse-eq-lim=8000
[fluid.alemesh.fieldsplit-2]
pc-type=gasm#gamg#lu
ksp-type=gmres#preonly
ksp-rtol=1e-3
ksp-maxit=50
#pc-gamg-threshold=0.02
mg-coarse.redundant.pc-factor-mat-solver-package-type=petsc
#pc-gamg-coarse-eq-lim=8000


[fluid.alemesh.ho]
pc-type=gasm
ksp-maxit=30
```

FSI - WavePressure 3D - .cfg VI

```
reuse-prec=true

[fluid.bdf]
order=2
[fluid.alemesh.bdf]
order=2

[solid]
filename=$cfgdir/wavepressure3d_solid.json

material_law=StVenantKirchhoff # StVenantKirchhoff, NeoHookean

pc-type=gasm#gamg#lu
pc-gamg-coarse-eq-lim=8000
#pc-gamg-threshold=0.02
mg-coarse.redundant.pc-factor-mat-solver-package-type=petsc

reuse-prec=1
#ksp-maxit=10
#ksp-converged-reason=true
```

FSI - WavePressure 3D - .cfg VII

```
[ts]
#restart=true
#time-initial=0.0013
time-step=0.0001
time-final=1.0
restart.at-last-save=true

[exporter]
directory=applications/models/fsi/wavepressure3d/
    $fluid_tag_$solid_tag_M1


[fluid]
scalability-save=1
# verbose=1
verbose_solvertimer=1
[solid]
scalability-save=1
# verbose=1
verbose_solvertimer=1
```

FSI - WavePressure 3D - .cfg VIII

```
[fsi]
# verbose=1
verbose_solvertimer=1
```

FSI - WavePressure 3D - .json I

Model config file

`~/Testcases/FSI/wavepressure3d/wavepressure3d_solid.json`

```
// -*- mode: javascript -*-
{
    "Name": "Arterial wall",
    "ShortName":"ArterialWall",
    "Model":"Elasticity",
    "Materials":
    {
        "ArterialWall":{
            "name":"solid",
            "E":"3e6",
            "nu":"0.3",
            "rho":"1.2"
        }
    },
    "BoundaryConditions":
```

FSI - WavePressure 3D - .json II

```
    {
        "displacement":
        {
            "Dirichlet":
            {
                "inletRing":
                {
                    "expr":"{0,0,0}"
                }
            },
            "Neumann_scalar":
            {
                "outletRing":
                {
                    "expr":"0"
                }
            },
            "interface_fsi":
            {
                "fsiWall":
```

FSI - WavePressure 3D - .json III

```
                {
                    "expr":"0"
                }
            },
            "robin":
            {
                "exterior":
                {
                    "expr1":"{1e4,0,0}",
                    "expr2":"{0,0,0}"
                }
            }
        }
    },
    "PostProcess":
    {
        "Fields":["displacement","pid"],
        "Measures":
        {
            "Maximum":
```

FSI - WavePressure 3D - .json IV

```
            {
                "fsiWall":
                {
                    "markers":"fsiWall",
                    "fields":["displacement"]
                }
            }
        }

}
```

FSI - WavePressure 3D - .json I

Model config file

`~/Testcases/FSI/wavepressure3d/wavepressure3d_fluid.json`

```
// -*- mode: javascript -*-
{
    "Name": "Blood fluid mechanics",
    "ShortName":"Blood",
    "Model":"Navier-Stokes",
    "Parameters":
    {
        "pIn":
        {
            "value":"1.3332e4"
        },
        "pTimeMax":
        {
            "value":"0.003"
        }
```

FSI - WavePressure 3D - .json II

```
    },
    "Materials":
    {
        "Blood":{
            "name":"myFluidMat",
            "rho":"1.0",
            "mu":"0.03"
        }
    },
    "BoundaryConditions":
    {
        "velocity":
        {
            "Neumann_scalar":
            {
                "inletBlood":
                {
                    "expr": "-(pIn/2.)*(1-cos(pi*t/(pTimeMax/2
                        .)))*(t<pTimeMax):pIn:pTimeMax:t"
                }
```

FSI - WavePressure 3D - .json III

```
        },
        "interface_fsi":
        {
            "fsiWall":
            {
                "expr":"0"
            }
        }
    },
    "fluid":
    {
        "outlet":
        {
            "outletBlood":
            {
                "expr":"0",                                  // avoid
                   to have a warning in output
                "number":1,                                  // number
                    of outlet [default=1]
```

FSI - WavePressure 3D - .json IV

```
                "alemesh_bc":"free",                    // fixed,
                    free [default=fixed]
                "model":"windkessel",                   // free,
                    windkessel [default=free]
                "windkessel_coupling":"explicit", //
                    explicit, implicit [default=implicit]
                "windkessel_Rd":6.2e3,                  //
                    resistance distal [default=1.0]
                "windkessel_Rp":400,                    //
                    resistance proximal [default=1.0]
                "windkessel_Cd":2.72e-4                 //
                    capacitance [default=1.0]
            }
        }
    }
},
"PostProcess":
{
    "Fields":["velocity","pressure","displacement","pid"],
    "Measures":
```

FSI - WavePressure 3D - .json V

```
        {
            "FlowRate":
            {
                "inlet":
                {
                    "markers":"inletBlood",
                    "direction":"interior_normal"
                },
                "outlet":
                {
                    "markers":"outletBlood",
                    "direction":"exterior_normal"
                }
            }
        }
    }
}
```

Toolbox Framework

CFD Toolbox

Computation Solid Mechanics toolbox

Fluid Structure Interaction toolbox

Heat Transfer toolbox

Thermo Electric toolbox

Heat Transfer Toolbox

https://book.feelpp.org/content/en/04-learning/HeatTransfer/

HeatTransfer - Thermo 2D I

Run an interactive feel++ docker container

```
docker run -it -v $HOME/feel:/feel \
       feelpp/feelpp/feelpp-toolboxes
```

You are now in the FEEL++ toolbox docker container.

Place yourself to HeatTransfer Thermo 2d test case directory then execute the HeatTransfer toolbox with given configs

```
cd Testcases/HeatTransfer/thermo2d/

mpirun -np 4 /usr/local/bin/feelpp_toolbox_thermodyn_2d \
       --config-file fsi3.cfg
```

HeatTransfer - Thermo 2D II

Take a look to the toolbox config **./thermo2d.cfg** and model config **thermo2d.json**.

For more possible options

```
feelpp_toolbox_thermodyn_2d --help
```

HeatTransfer - Thermo 2D - .cfg I

Config file **~/Testcases/HeatTransfer/thermo2d/thermo2d.cfg**

```
[thermo]
geofile=$cfgdir/thermo2d.geo
[thermo.gmsh]
hsize=0.04#0.05

[thermo]
filename=$cfgdir/thermo2d.json

initial-solution.temperature=293

velocity-convection={-30*(y+-0.7)*(y-0.5)*(y>0.5)*(y<0.7),0}:y
#velocity-convection={-30*(y+sin(pi*t/3.)-0.7)*(y+sin(pi*t/3.)
    -0.5)*((y+sin(pi*t/3.))>0.5)*((y+sin(pi*t/3.))<0.7),0}:y:t

do_export_all=true
```

HeatTransfer - Thermo 2D - .cfg II

```
#verbose=1
#verbose_solvertimer=1
reuse-prec=1
pc-type=lu

[thermo.bdf]
order=2

[ts]
time-step=0.1#0.01
time-final=20
restart.at-last-save=true


[exporter]
directory=applications/models/thermodyn/P1G1
```

HeatTransfer - Thermo 2D - .json I

Model config file ~/Testcases/HeatTransfer/thermo2d/thermo2d.json

```
// -*- mode: javascript -*-
{
    "Name": "Thermo dynamics",
    "ShortName":"ThermoDyn",
    "Model":"mythermicmodel",
    // "Parameters":
    // {
    //     "gravity":
    //     {
    //         "value":"-1"
    //     }
    // },
    "Materials":
    {
        //"Omega":{ "name":"Copper","filename":"$feelpp_srcdir/
            databases/materials/pure/copper.json" }
```

HeatTransfer - Thermo 2D - .json II

```
        "Omega":
        {
            "name":"mymat",
            "k11":"10",//[ W/(m*K) ]
            "Cp":"1000.005", //[ J/(kg*K) ]
            "rho":"1.205" //[ kg/(m^3) ]
        }
    },
    "BoundaryConditions":
    {
        "temperature":
        {
            "Dirichlet":
            {
                "MarkerDirichlet":
                {
                    "expr":"293-10*(y-1)*(y+1):y"
                },
                "wall":
                {
```

HeatTransfer - Thermo 2D - .json III

```
                    "expr":"293+0.38*t:t"
                }
            },
            "Neumann":
            {
                "MarkerNeumann":
                {
                    "expr":"0:x"
                }
            },
            "Robin":
            {
                "MarkerRobin":
                {
                    "expr1":"16.667",// h coeff
                    "expr2":"287"// temperature exterior
                }
            }
        }
    },
```

HeatTransfer - Thermo 2D - .json IV

```
    "PostProcess":
    {
        "Fields":["temperature"]
    }

}
```

Toolbox Framework

CFD Toolbox

Computation Solid Mechanics toolbox

Fluid Structure Interaction toolbox

Heat Transfer toolbox

Thermo Electric toolbox

Thermoelectric Toolbox

https://book.feelpp.org/content/en/03-modeling/ThermoElectric/

ThermoElectric - Test 2D I

Run an interactive feel++ docker container

```
docker run -it -v $HOME/feel:/feel \
       feelpp/feelpp/feelpp-toolboxes
```

You are now in the FEEL++ toolbox docker container.

Place yourself to ThermoElectric test 2d test case directory then execute the ThermoElectric toolbox with given configs

```
cd Testcases/ThermoElectric/test/

mpirun -np 4 /usr/local/bin/feelpp_toolbox_thermoelectric_2d \
       --config-file test-thermoelectric.cfg
```

ThermoElectric - Test 2D II

Take a look to the toolbox config **./test-thermoelectric.cfg** and model config **test-thermoelectric.json**.

For more possible options

```
feelpp_toolbox_thermoelectric_2d --help
```

ThermoElectric - Test 2D - .cfg I

Config file **~/Testcases/ThermoElectric/test/test-thermoelectric.cfg**

```
fe-approximation=P1 #P1,P2,P3

[thermo-electric]
geofile=$cfgdir/test-thermoelectric.geo
gmsh.hsize=0.05
filename=$cfgdir/test-thermoelectric.json
#verbose=1
verbose_solvertimer=1
pc-type=lu
ksp-monitor=1
snes-monitor=1

[thermo-electric.thermo]
filename=$cfgdir/test-thermoelectric.json
#initial-solution.temperature=293
#verbose=1
```

ThermoElectric - Test 2D - .cfg II

```
verbose_solvertimer=1
pc-type=lu
ksp-monitor=1

[thermo-electric.electric]
pc-type=lu
ksp-monitor=1


[exporter]
directory=applications/models/thermoelectric/test
```

ThermoElectric - Test 2D - .json I

Model config file

`~/Testcases/ThermoElectric/test/test-thermoelectric.json`

```
// -*- mode: javascript -*-
{
    "Name":"myThermoElectricName",
    "ShortName":"myThermoElectricShortName",
    "Model":"ThermoElectric-nonlinear",//"ThermoElectric-
       nonlinear" or "ThermoElectric-linear"
    "Materials":
    {
        //"Omega":{ "name":"Copper","filename":"$feelpp_srcdir/
           databases/materials/pure/copper.json" }
        "Omega":
        {
            "name":"mymat",
            "k11":"370",//[ W/(m*K) ]
```

ThermoElectric - Test 2D - .json II

```
            //"Cp":"1000.005", //[ J/(kg*K) ]
            // "rho":"1.205", //[ kg/(m^3) ]
            "sigma":"53e6"
        }
    },
    "BoundaryConditions":
    {
        "temperature":
        {
            "Dirichlet":
            {
                "top":
                {
                    "expr":"10"
                },
                "bottom":
                {
                    "expr":"0"
                }
            },
```

ThermoElectric - Test 2D - .json III

```
            "Neumann":
            {
                "cylinder":
                {
                    "expr":"0"
                }
            }
        },
        "electric-potential":
        {
            "Dirichlet":
            {
                "right":
                {
                    "expr":"3"
                },
                "left":
                {
                    "expr":"1"
                }
```

ThermoElectric - Test 2D - .json IV

```
            }
        }
    },
    "PostProcess":
    {
        "Fields":["temperature", "electric-potential","electric
            -field","pid"]
    }

}
```

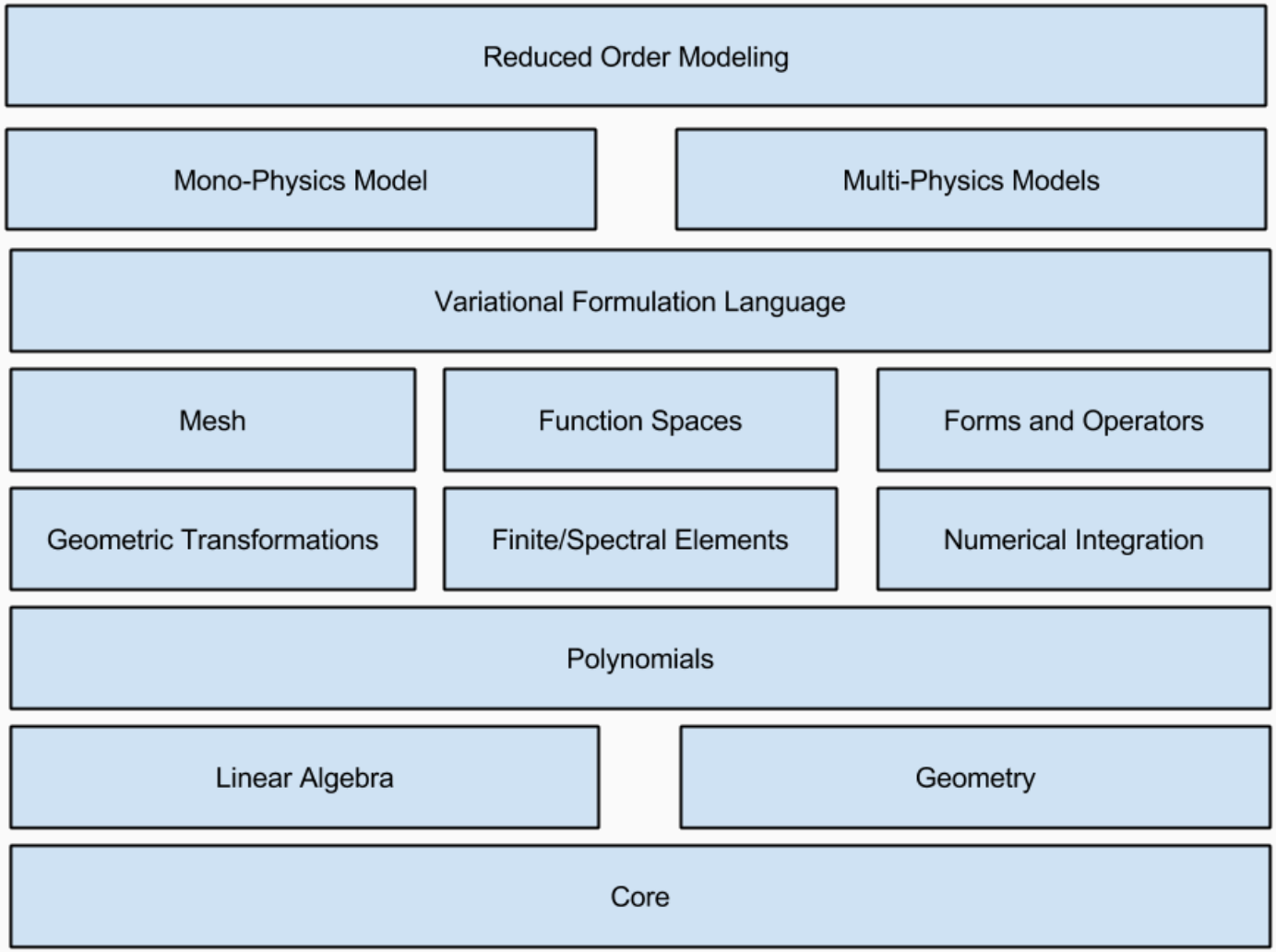
Part IV

Programming with Feel++

Programming with Feel++

- Mesh
- Function Spaces
- Operators and Forms
- Pre/Post-Processing
- Language

Architecture



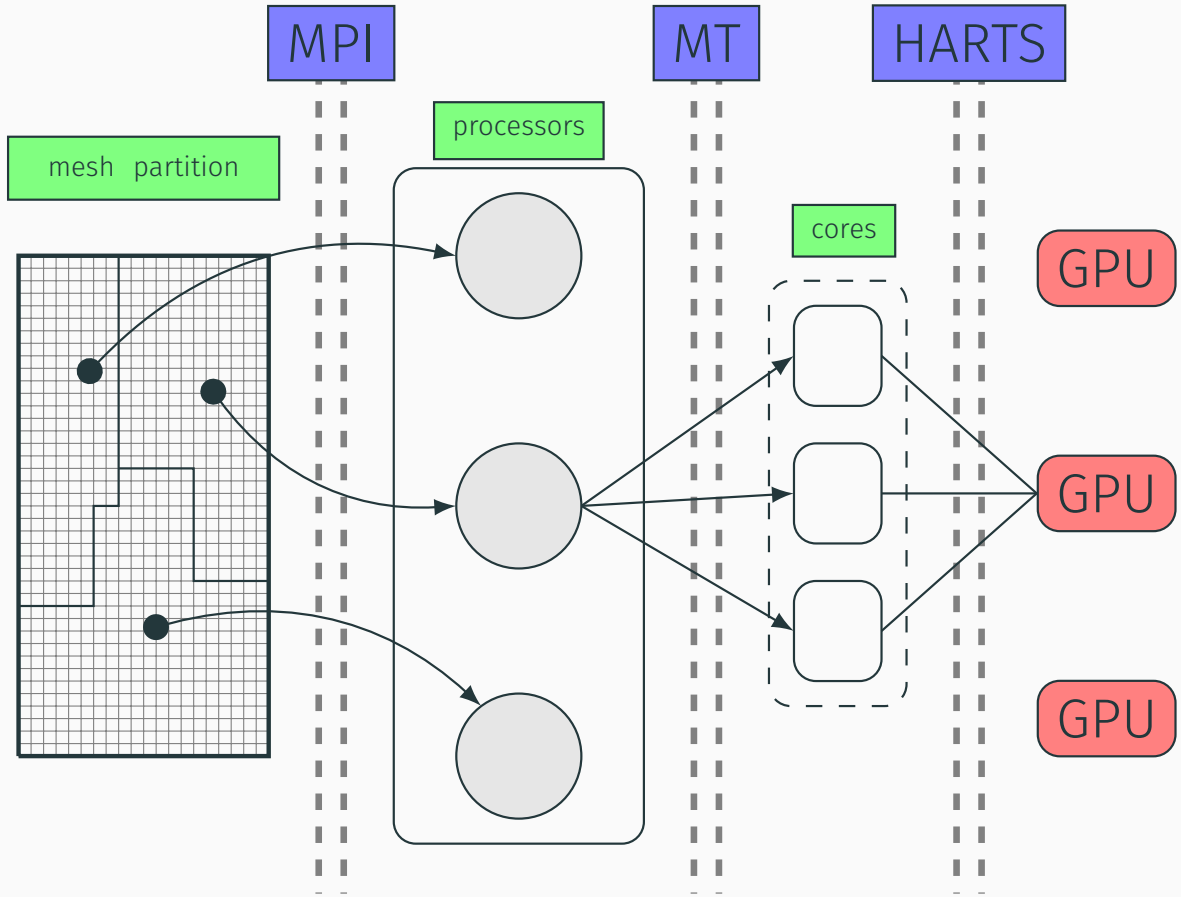
Seamless parallelisation

Hybrid architectures

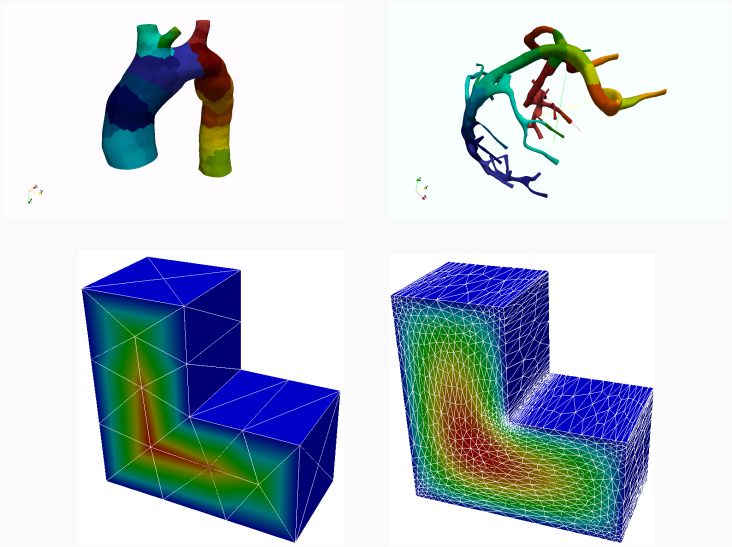
- many nodes, many cores (one day: hybrid nodes)
- MPI, multi-threads (one day: GPU)

MPI implementation :

- mesh partitioning
- dof table partitioning
- PETSc interface



Mesh(Parallel) I



Mesh adaptation on 3D L-shape

- Convexes and associated geometric transformation ($\mathbb{P}_{\mathbb{N}}, N = 1, 2, 3, 4, 5...$)
- 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 extraction (mesh of faces, mesh of edges,...) in parallel
- Mesh adaptation (isotropic versus anisotropic) in sequential

Mesh(Parallel) II

Example

```
1 elements(mesh [, processid]);
2 markedelements(mesh, {marker1, marker2,...}, [, processid]);
3 boundaryfaces(mesh, [, processid]);
4 markedfaces(mesh, {marker1, marker2,...}, [, processid]);
5 markededges(mesh, {marker1, marker2,...}, [, processid]);
6 markedpoints(mesh, {marker1, marker2,...}, [, processid]);
7 auto trace_mesh = mesh->trace( markedfaces(mesh,marker1) );
8 auto submesh = createSubmesh( mesh,
9                               markedelements(mesh,marker2));
```

▸ Learn more

Feel++ keeps mesh relationships I

Extraction issue

If we extract sub-meshes from a parent mesh and wish to manipulate functions defined on both meshes (parent and children), it has to be efficient. Feel++ keeps tracks for you of the relationships (child/parent and siblings) automatically.

Function Spaces(Parallel) I

- 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 (only 2D) and non-periodic spaces
- Support mortar function spaces (2D)
- Seamless parallel support (except for periodic and different mesh types)

Function Spaces(Parallel) II

```
auto P1h = Pch<1>( mesh ); // $\mathbb{P_1}$ space
auto v = P1h->element(); // a $C^0$ piecewise linear function
auto Xh = THch<2>( mesh ); // $[\mathbb{P}_{N+1}]^d \times \mathbb{P}_{N}
auto Uh = Xh->functionSpace<0>();
auto x = Xh->element();
auto p = x.element<1>(); // view
auto Mh = Moch<1>( mesh ); // Mortar space
auto Yh=product(P1h,4); // Yh cartesian space
auto Zh=product(P1h,Uh); // Zh cartesian space
```

▸ Learn more

Operators and Forms (Parallel) I

- 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
  - Shell matrices
- The link between the variational expression and the algebraic representation
- Supports cartesian product spaces with same and difference space types (dynamic for the former, static for the later)

Operators and Forms (Parallel) II

Example

```
auto Xh = Pch<5>(mesh); auto Vh = Pch<3>(mesh);
auto u = Xh->element(); auto v = Vh->element();
// operator $T: X_1 \rightarrow X_2$
auto T = opLinear( Xh, Vh [, backend] );
T = integrate(elements(mesh), id(u)*idt(v) );
// linear functional $f: X_2 \rightarrow \mathbb{R}$
auto f = funLinear( Vh [, backend] );
T.apply( u, f ); f.apply( v );
auto opP1 = lagrangeP1( Xh, mesh );
```

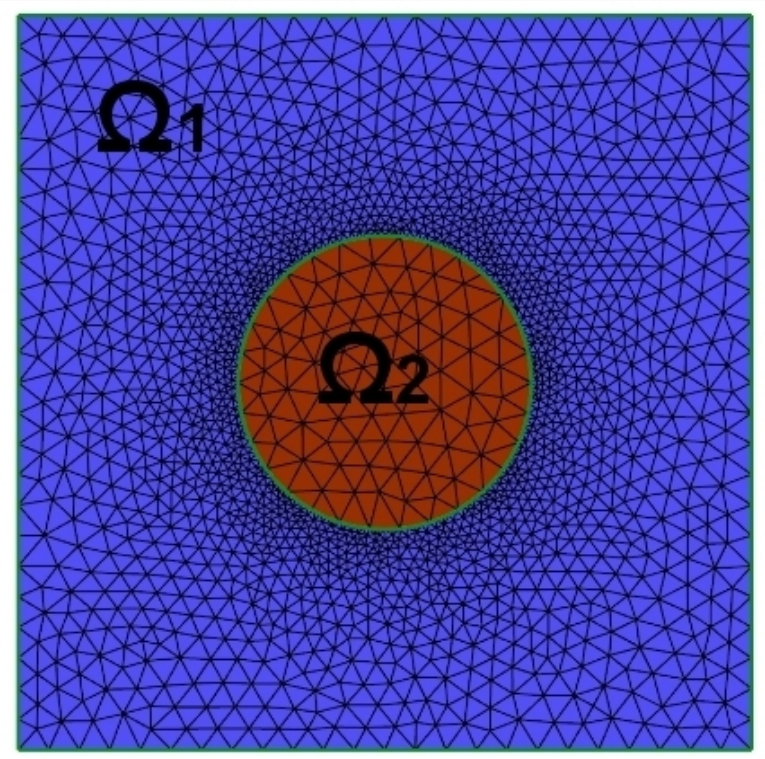
Operators and Forms (Parallel) III

```
auto ps = product(Xh,Wh);
auto u = Xh->element();
auto v = Wh->element();
auto bbf = blockform2( ps );
bbf( 0_c, 0_c ) = integrate( _range=elements(mesh),
                             _expr=id(u)*idt(u) );
bbf( 0_c, 1_c ) += integrate( _range=elements(mesh),
                              _expr=id(u)*(trans(idt(v))*one() ) ) ;
bbf( 1_c, 0_c ) += integrate( _range=elements(mesh),
                              _expr=1./2*(trans(id(v))*one())*idt(u) );
bbf( 1_c, 1_c ) += integrate( _range=elements(mesh),
                              _expr=trans(id(v))*idt(v) );
```

Seamless Interpolation Tool(Parallel)

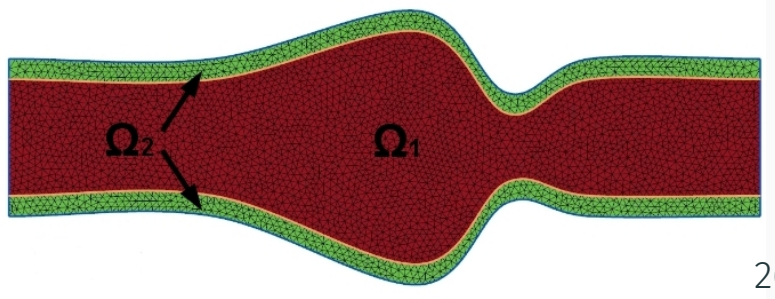
Motivations

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



Some Applications

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



Post-Processing I

Exporter

- Support for Ensight format (ensight and ensighgold): parallel, 1D-3D, works well but order 1 geometry
- Support for Gmsh format : sequential, 1D-3D, high order geometry support
- Time stepping support
- Seamless interpolation support
- Support ghost cells (ensight gold)
- Xdmf/HDF5 and VTK support
- Altair/Hypermesh

Post-Processing II

```
{
    using namespace Feel;
    Environment env( _argc=argc, _argv=argv,
                     _desc=feel_options(),
                     _about=about(_name="exporters",
                                  _author="christophe prud'
                                  _email="christophe.prudho
    // ``order 2'' circle
    auto mesh = unitCircle<2>();
    // $\mathbb{p}_2$ space
    auto Xh = Pch<2>( mesh );
    auto v = project( _space=Xh,
                      _range=elements(mesh),
                      _expr=sin(pi*Px()));
```

Post-Processing III

```
auto exhi = exporter( _mesh=mesh,
                      _name="exhi" );
exhi->add( "vhi", v );

// ``order 1'' sphere
auto meshp1 = unitCircle<1>();

auto exlo = exporter( _mesh=meshp1,
                      _name="exlo" );
exlo->add( "vlo", v );

// P2isoP1 mesh
auto lagmesh=lagrangeP1(_space=Xh)->mesh();
auto exhilo = exporter( _mesh=lagmesh,
                        _name="exhilo");
```

Post-Processing IV

```
    exhilo->add( "vhilo", v );
    exhi->save();
    exlo->save();
    exhilo->save();
}
```

Feel++ Keywords

▸ List of keywords

some keywords

- math functions
- geometry
- finite element, galerkin keywords
- some useful expression

Some Language Additions

- `mean(_range=elements(mesh),_expr=sin(pi*Px()));`:
  $\left(\int_\Omega \sin \pi x\right) / \int_\Omega 1$
- `normL2(_range=elements(mesh),_expr=sin(pi*Px()));`:
  $\left(\int_\Omega (\sin \pi x)^2\right)^{1/2}$
- `normL2Squared(_range=elements(mesh),_expr=sin(pi*Px()));`:
  $\int_\Omega (\sin \pi x)^2$
- `normLinf(_range=elements(mesh),_expr=sin(pi*Px()));`:
  $sup_{x\in\Omega} |\sin \pi x|$
- meta-expression:

```
auto I = integrate(_range=elements(mesh),_expr=_e1 );
auto v1 = I( idv(v) ).evaluate();
auto v2 = I( cst(1) ).evaluate();
```

Evaluation of an expression at a set of points

Problem

We wish to compute repeatedly an expression at a (small) set of points in parallel (point wise measures, interpolation ...)

```
// define mesh here
auto Xh = Pch<3>( mesh );
auto ctx = Xh->context();
// accumulate points
ctx->add( pt1 );
ctx->add( pt2 );
// ...
//  evaluation e.g. u at the set of points
// call repeatedly this function after updating u for example
evaluateFromContext( _expr=idv(u), _context=ctx );
```

Symbolic Computing and the link with Numerical Computing

- Improved support for GiNaC (`http://www.ginac.de`
- Added some functions for scalar and vector fields: **`laplacian`**, **`grad`**, **`div`**
  more can be easily added
- integration to the variational language
- good basis for scripting interfaces, first steps in **`.cfg`** files

```
mu=2.3
beta=4
e=t+sin(mu*pi*x)*cos(beta*pi*z)*cos(pi*y):x:y
      :z
```