FreeFem++ : States and News

F. Hecht
Laboratoire Jacques-Louis Lions
Université Pierre et Marie Curie
Paris, France

http://www.freefem.org mailto:hecht@ann.jussieu.fr

With the support of ANR (French gov.) ANR-07-CIS7-002-01

http://www.freefem.org/ff2a3/ http://www-anr-ci.cea.fr/

Outline

FINANCÉ PAR ANR

Introduction `FreeFem++` is a software to solve numerically

partial differential equations (PDE) in $\mathbb{R}^2$) and in $\mathbb{R}^3$) with finite elements methods. We used a user language to set and control the problem. The `FreeFem++` language allows for a quick specification of linear PDE's, with the variational formulation of a linear steady state problem and the user can write they own script to solve no linear problem and time depend problem. You can solve coupled problem or problem with moving domain or eigenvalue problem, do mesh adaptation , compute error indicator, etc ...

By the way, FreeFem++ is build to play with abstract linear, bilinear form on Finite Element Space and interpolation operator.

FreeFem++ is a freeware and this run on Mac, Unix and Window architecture, in parallel with MPI.

History

1987 MacFem/PCFem les ancêtres (O. Pironneau en Pascal) payant.

1992 FreeFem réécriture de C++ (P1,P0 un maillage) O. Pironneau, D. Bernardi, F. Hecht , C. Prudhomme (adaptation Maillage, bamg).

1996 FreeFem+ réécriture de C++ (P1,P0 plusieurs maillages) O. Pironneau, D. Bernardi, F. Hecht (algèbre de fonction).

1998 FreeFem++ réécriture avec autre noyau élément fini, et un autre langage utilisateur ; F. Hecht, O. Pironneau, K.Ohtsuka.

1999 FreeFem 3d (S. Del Pino) , Une première version de freefem en 3d avec des méthodes de domaine fictif.

2008 FreeFem++ v3 réécriture du noyau élément fini pour prendre en compte les cas multidimensionnels : 1d,2d,3d...

For who, for what !

For what

1. R&D
2. Academic Research ,
3. Teaching of FEM, PDE, Weak form and variational form
4. Algorithmes prototyping
5. Numerical experimentation
6. Scientific computing and Parallel computing

For who : the researcher, engineer, professor, student...

The mailling list `mailto:Freefempp@ljll.math.upmc.fr` with 308 members with a flux of 5-20 messages per day.

The main characteristics I/II (2D)(3D)

- Wide range of finite elements : continuous P1,P2 elements, discontinuous P0, P1, RT0,RT1,BDM1, elements ,Edge element, vectorial element, mini-element, ...
- Automatic interpolation of data from a mesh to an other one (with matrix construction if need), so a finite element function is view as a function of (x, y, z) or as an array.
- Definition of the problem (complex or real value) with the variational form with access to the vectors and the matrix.
- Discontinuous Galerkin formulation (only in 2d to day).
- LU, Cholesky, Crout, CG, GMRES, UMFPack, SuperLU, MUMPS, HIPS, HYPRE, SUPERLU_DIST, PASTIX. ... sparse linear solver ; eigenvalue and eigenvector computation with ARPACK.
- Online graphics with OpenGL/GLUT/VTK, C++ like syntax.
- An integrated development environment FreeFem++-cs

The main characteristics II/II (2D)(3D)

- Analytic description of boundaries, with specification by the user of the intersection of boundaries in 2d.
- Automatic mesh generator, based on the Delaunay-Voronoï algorithm. (2d,3d (tetgen))
- load and save Mesh, solution
- Mesh adaptation based on metric, possibly anisotropic (only in 2d), with optional automatic computation of the metric from the Hessian of a solution. (2d,3d).
- Link with other soft : parview, gmsh , vtk, medit, gnuplot
- Dynamic linking to add plugin.
- Full MPI interface
- Nonlinear Optimisation tools : CG, NLOpt, stochastic
- Wide range of examples : Navier-Stokes 3d, elasticity 3d, fluid structure, eigenvalue problem, Schwarz' domain decomposition algorithm, residual error indicator ...

How to use

on Unix build a "yours.edp" file with your favorite editor : emacs, vi, nedit, etc. Enter`FreeFem++ yours.edp` or `FreeFem++` must be in a directory of your `PATH` shell variable.

on Window, MacOs X build a "yours.edp" file with your favorite text editor (raw text, not word text) : emacs, winedit, wordpad, bbedit, fraise ... and click on the icon of the application `FreeFem++` and load you file via de open file dialog box or drag and drop the icon of your built file on the application `FreeFem++` icon.

The Changes form 12/09 to 12/10

- add some interface with lapack (inverse of full matrix, eigenvalue of full matrix)
- change the version of tetgen to 1.4.3 download (4 time faster)
- add 3d beam examples `e*3d/beam.edp`
- add dynamic load interface with newuoa fortran optimizer without derivative see ffnewuoa.edp example ins examples++-load
- correct problem of free of mesh in case of gluing meshes $\mathrm{Th} = \mathrm{Tha} + \mathrm{Thb};$
- add 3d example and interpolation matrix ($\mathrm{e} * 3\mathrm{d}/\mathrm{mat\_interpole.edp}$)
- writing schwarz-nm-3d.edp examples
- add array of 3d mesh
- add seekp, tellp method on ostream type seekg, teeg method on istream type
- correct ' operator do alway in complex case the conj and trans (Hermissian stuff)
- plot of complex field and 3d vector .. (? ? ?)
- uniformize named parameter in change, movemesh, in load "msh" , glumesh, ...
- change mesh2 tool to make a renumbering of vertex for periodic 3d mesh. $\mathrm{Th} = \mathrm{change}(\mathrm{Th}, \mathrm{renumv} = \mathrm{old2new});$
- correct SubString
- add automatic compilation of download software, (lots of parallel solver)
- Simplify the compilation of dynamics plugin with `ff-c++ -auto`
- correct mistake in mpi
- do asynchronous Isend/Irecv of matrix , meshes.
- 3d mesh adaptation exemple
- correct interface of `mshmet` (compute 3d metric), `freeyams` surface adaptation, `mmg3d` (3d mesh movement)
- add quoting {} in macro argument
- add script to simple the launch of mpi case
- add MPI version on Windows (not tested)
- Compile a version for Windows 64 arch ()

The Changes form 01/11 to 08/11

- Add thresholdings.cpp thresholdings.edp in examples++-load to remove to small coef in a matrix .
- Add lots of DDM Schwarz GMRES preconditioned with a coarse grid solver overlapping :
- Add a true exemple to build mesh form a image (lg.pgm) , Leman-mesh.edp
- Add New syntaxe in macro generation `NewMacro a(i)` .... `EndMacro` with // comment and macro definition.
- Add interface with special function to GSL `http://www.gnu.org/software/gsl/` the full list is in the example `examples++-loal/gsl.edp`
- correct the precond of gmres algo in A^{-1} : (29 mars 2011) to build a correct Navier-Stokes
- add cast TypeOfSolver on int to parametrize the choose of linear solver.
- put comment in the documentation about negative tgv
- correct `mpiReduce` for complex and real data.
- add `mpiAllReduce(umax,dmaxg,comm,mpiMAX);` where for real umax,dmaxg ;
- update the finite element list in documentation
- add (load "Element-Mixte") NEW FINITE ELEMENT 2D TDNSS1 sym matrix 2x2 conforme in $\{H(divdiv)/div(divs)) \in H^{-1}\}$, RT1 and BDM1 conforme in H(div) , RT1 and BDM1ortho conforme in $H(curl)$ Nedelec Finite Elem.
- add matrix multiplication with lapack interface
- add tool to change the region and label number change(Th,fregion= integer function to set the region number)
- nuTriangle now given the tet number in 3D.
- add mpireduce of matrix to build parallel matrix
- correct the Hips interface (not to bad , in test)
- add interface with `MUMPS_4.10.0` version (with automatic download)
- add `P1dc3d` finite element in pluging `"Element_P1dc1"`
- correct bug in gibbs renumbering in 1 case veru sample mesh)
- correct bug in interpolation operator with periodic B.C.
- Add use of `MKL` library, use `mmg3d v4`
- Add interface to NLopt and cmaes (scalar and //)
- correct plugins build for parallel version.

FINANCÉ PAR ANR

Laplace equation, weak form

Let a domain Ω with a partition of $\partial\Omega$ in Γ_2, Γ_e.
Find u a solution in such that :

$$-\Delta u = 1 \text{ in } \Omega, \quad u = 2 \text{ on } \Gamma_2, \quad \frac{\partial u}{\partial \vec{n}} = 0 \text{ on } \Gamma_e \qquad (1)$$

Denote $V_g = \{v \in H^1(\Omega)/v_{|\Gamma_2} = g\}$.
The Basic variationnal formulation with is : find $u \in V_2(\Omega)$, such that

$$\int_\Omega \nabla u.\nabla v = \int_\Omega 1v + \int_\Gamma \frac{\partial u}{\partial n} v, \quad \forall v \in V_0(\Omega) \qquad (2)$$

The finite element method is just : replace V_g with a finite element space, and the `FreeFem++` code :

Laplace equation in FreeFem++

The finite element method is just : replace V_g with a finite element space, and the FreeFem++ code :

```
mesh3 Th("Th-hex-sph.msh");                    //    read a mesh 3d
fespace Vh(Th,P1);                        //    define the P1 EF space

 Vh u,v;       //    set test and unknow FE function in Vh.
macro Grad(u) [dx(u),dy(u),dz(u)]              //   EOM Grad def
solve laplace(u,v,solver=CG) =
   int3d(Th)(  Grad(u)'*Grad(v)   )
   - int3d(Th) ( 1*v)
   + on(2,u=2);                                //    int on γ2
plot(u,fill=1,wait=1,value=0,wait=1);
```

Run:fish.edp Run:Laplace3d.edp Run:EqPoisson.edp

Mesh adaptation : Metric / unit Mesh

In Euclidean geometry the length $|\gamma|$ of a curve γ of $\mathbb{R}^d$ parametrized by $\gamma(t)_{t=0..1}$ is

$$|\gamma| = \int_0^1 \sqrt{< \gamma'(t), \gamma'(t) >} dt$$

We introduce the metric $\mathcal{M}(x)$ as a field of $d \times d$ symmetric positive definite matrices, and the length ℓ of Γ w.r.t $\mathcal{M}$ is :

$$\ell = \int_0^1 \sqrt{< \gamma'(t), \mathcal{M}(\gamma(t))\gamma'(t) >} dt$$

The key-idea is to construct a mesh where the lengths of the edges are close to 1 accordingly to $\mathcal{M}$.

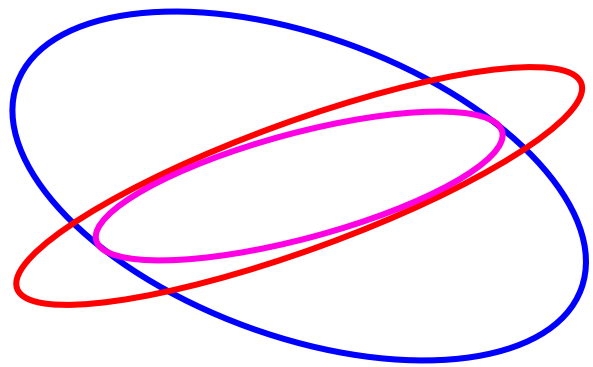
The main IDEA for mesh generation

- The difficulty is to find a tradeoff between the error estimate and the mesh generation, because this two work are strongly different.
- To do that, we propose way based on a metric $\mathcal{M}$ and unit mesh w.r.t $\mathcal{M}$
- The metric is a way to control the mesh size.
- remark : The class of the mesh which can be created by the metric, is very large.

Metrix intersection

The unit ball $\mathcal{BM}$ in a metric $\mathcal{M}$ plot the maximum mesh size on all the direction, is a ellipse.
If you we have two unknowns u and v, we just compute the metric $\mathcal{M}_u$ and $\mathcal{M}_v$, find a metric $\mathcal{M}_{uv}$ call intersection with the biggest ellipse such that :

$$\mathcal{B}(\mathcal{M}_v) \subset \mathcal{B}(\mathcal{M}_u) \cap \mathcal{B}(\mathcal{M}_v)$$

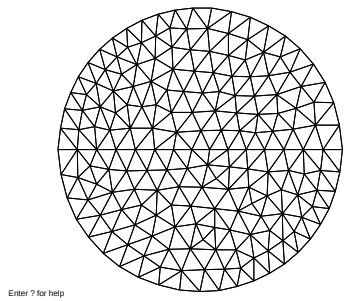


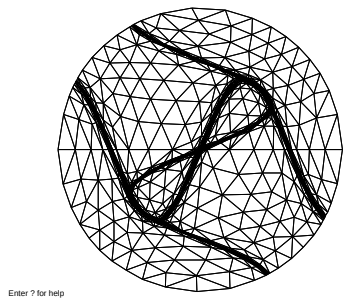
With suppose, if the mesh size decreases in one direction then we get a better approximation.

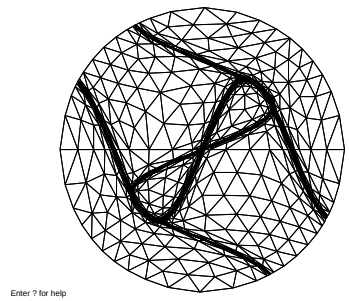
Example of mesh

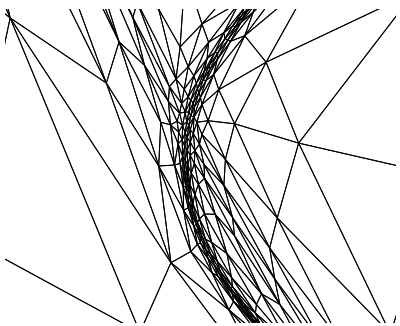
$$u = (10 * x^3 + y^3) + atan2(0.001, (sin(5 * y) - 2 * x))$$

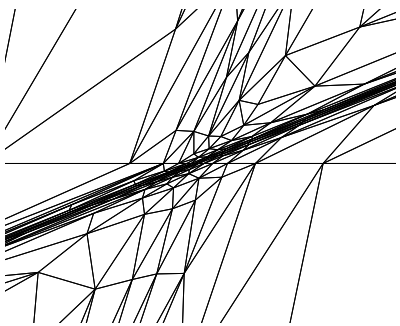
$$v = (10 * y^3 + x^3) + atan2(0.01, (sin(5 * x) - 2 * y)).$$

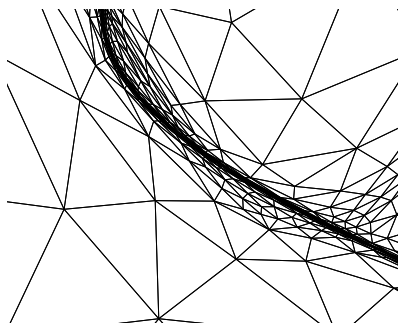












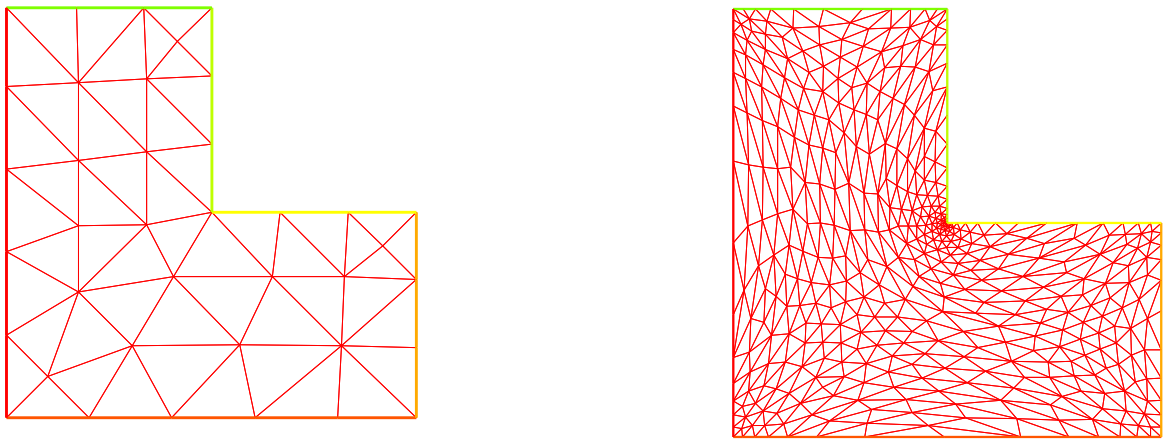
A corner singularity

adaptation with metric

The domain is an L-shaped polygon $\Omega =]0,1[^2 \setminus [\frac{1}{2},1]^2$ and the PDE is

$$\text{Find } u \in H_0^1(\Omega) \text{ such that } \quad -\Delta u = 1 \text{ in } \Omega,$$

The solution has a singularity at the reentrant angle and we wish to capture it numerically.



example of Mesh adaptation

FreeFem++ corner singularity program

```
int[int] lab=[1,1,1,1];
mesh Th = square(6,6,label=lab);
Th=trunc(Th,x<0.5 | y<0.5, label=1);

fespace Vh(Th,P1);          Vh u,v;          real error=0.1;
problem Probem1(u,v,solver=CG,eps=1.0e-6) =
       int2d(Th)(  dx(u)*dx(v) + dy(u)*dy(v))
     - int2d(Th)( v) + on(1,u=0);
for (int i=0;i< 7;i++)
{  Probem1;                                  //    solving the pde
   Th=adaptmesh(Th,u,err=error,nbvx=100000);
                        //    the adaptation with Hessian of u
   plot(Th,u,wait=1,fill=1);                  u=u;
   error = error/ (1000.^(1./7.));   };
```

Run:CornerLap.edp

Poisson with Dirac RHS

$$-\Delta u = \delta_{(\frac{1}{4},\frac{1}{4})} + 2\delta_{(\frac{3}{5},\frac{3}{5})}, \quad u = 0 \text{ on } \Gamma$$

Remark, The Classical Weak formulation is incorrect because $\delta \notin H^{-1}$ but :

```
real[int] xdelta = [0.25,0.6], ydelta = [0.25,0.6]
          , cdelta=[1.,2.];
mesh Th=square(10,10);                verbosity=0;
for(int iter=0;iter < 20;iter++)
{ fespace Vh(Th,P1);                          //    P1 FE space
 Vh uh,vh;                        //    unkown and test function.
 matrix D = interpolate(Vh,xdelta,ydelta);
 real[int]  b= D*cdelta;  b= -b;
 solve laplace(uh,vh) =
     int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) )  + b[]
     + on(1,2,3,4,uh=0) ;
 plot(uh,wait=1,dim=3,fill=1);
 Th=adaptmesh(Th,uh,nbvx=100000,err= 0.01*1.2^-iter); }
```

Run:LaplaceDirac.edp

Poisson equation with 3d mesh adaptation

```
load "msh3" load "tetgen" load "mshmet" load "medit"
int nn  = 6;
int[int] l1111=[1,1,1,1],l01=[0,1],l11=[1,1];                    //   label numbering
mesh3 Th3=buildlayers(square(nn,nn,region=0,label=l1111),
      nn,  zbound=[0,1], labelmid=l11,labelup = l01,labeldown = l01);
Th3=trunc(Th3,(x<0.5)|(y < 0.5)|(z < 0.5) ,label=1);              //   remove ]0.5, 1[^3

fespace Vh(Th3,P1);                    Vh u,v;     //   FE. space definition
macro Grad(u) [dx(u),dy(u),dz(u)]                                  //    EOM
problem Poisson(u,v,solver=CG) =
  int3d(Th3)( Grad(u)'*Grad(v) )  -int3d(Th3)( 1*v ) + on(1,u=0);

real errm=1e-2;                                          //    level of error
for(int ii=0; ii<5; ii++)
{
  Poisson;         Vh h;
  h[]=mshmet(Th3,u,normalization=1,aniso=0,nbregul=1,hmin=1e-3,
             hmax=0.3,err=errm);
  errm*= 0.8;                                  //    change the level of error
  Th3=tetgreconstruction(Th3,switch="raAQ",sizeofvolume=h*h*h/6.);
  medit("U-adap-iso-"+ii,Th3,u,wait=1);
}
```

Run:Laplace-Adapt-3d.edp

Build Matrix and vector of problem

The 3d FreeFem++ code :

```
mesh3 Th("dodecaedre.mesh");    //    read a mesh from file
fespace Vh(Th,P1);              //    define the P1 FE space

macro Grad(u) [dx(u),dy(u),dz(u)] //    End of Macro

varf vlaplace(u,v,solver=CG) =
   int3d(Th)( Grad(u)'*Grad(v) )  +  int3d(Th) ( 1*v)
  + on(2,u=2);                                  //    on γ2

matrix A= vlaplace(Vh,Vh,solver=CG);    //    bilinear part
real[int] b=vlaplace(0,Vh);             //    // linear part
Vh u;
u[] = A^-1*b;                   //    solve the linear system
```

Remark on varf

The functions appearing in the variational form are formal and local to the `varf` definition, the only important think is the order in the parameter list, like in

```
varf vb1([u1,u2],[q]) = int2d(Th)( (dy(u1)+dy(u2))*q) + int2d(Th)(1*q);
varf vb2([v1,v2],[p]) = int2d(Th)( (dy(v1)+dy(v2))*p) + int2d(Th)(1*p);
```

To build matrix A from the bilinear part and the "on" part, the variational form a of type `varf` do simply

```
  matrix B1 = vb1(Vh,Wh [, ...] );
  matrix<complex> C1 = vb1(Vh,Wh [, ...] );
//    where the fespace have the correct number of comp.
 //     Vh is "fespace" for the unknown fields with 2 comp.
 //     ex fespace Vh(Th,[P2,P2]); or fespace Vh(Th,RT);
 //     Wh is "fespace" for the test fields with 1 comp.
```

To build a vector form the linear part and the "on" part , put $u1 = u2 = 0$ by setting 0 of on unknown part.

```
  real[int]     b = vb2(0,Wh);
  complex[int]  c = vb2(0,Wh);
```

Remark : In this case the mesh use to defined , $\int, u, v$ can be different.

The boundary condition terms

The only label of the edge (2d) or the faces (3d) is used.

- An "on" scalar form (for Dirichlet) : `on(1, u = g )`

  The meaning is for all degree of freedom i of this associated boundary, the diagonal term of the matrix $a_{ii} = tgv$ with the *terrible giant value* `tgv` (=10^{30} by default) and the right hand side $b[i] = "(\Pi_h g)[i]" \times tgv$, where the $"(\Pi_h g)g[i]"$ is the boundary node value given by the interpolation of g.

- An "on" vectorial form (for Dirichlet) : `on(1,u1=g1,u2=g2)` If you have vectorial finite element like `RT0`, the 2 components are coupled, and so you have : $b[i] = "(\Pi_h(g1, g2))[i]" \times tgv$, where Π_h is the vectorial finite element interpolant.

- a linear form on Γ or on $\Gamma_1 \cup \Gamma_4$ (for Neumann in 2d)

  `-int1d(Th)( f*w)` or `-int1d(Th,1,4)( f*w)`

- a bilinear form on Γ or Γ_2 (for Robin in 2d)

  `int1d(Th)( K*v*w)` or `int1d(Th,2)( K*v*w)`.

- a linear form on Γ or on $\Gamma_2 \cup \Gamma_3$ (for Neumann in 3d)

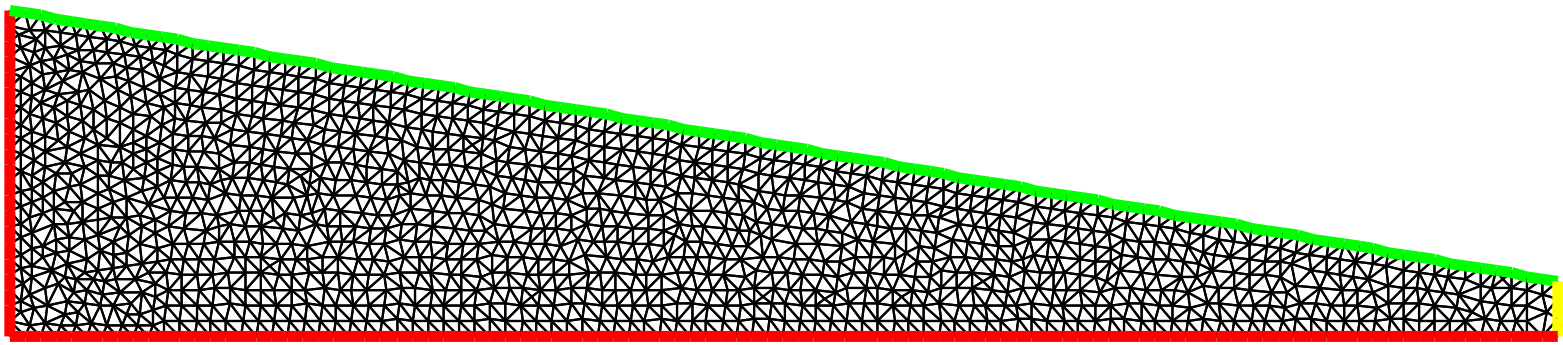
  `-int2d(Th)( f*w)` or `-int2d(Th,2,3)( f*w)`

A Free Boundary problem , (phreatic water)

Let a trapezoidal domain Ω defined in FreeFem++ :

```
real L=10;                                          //    Width
real h=2.1;                                        //    Left height
real h1=0.35;                                      //    Right height
border a(t=0,L){x=t;y=0;label=1;}; //    impermeable Γ_a
border b(t=0,h1){x=L;y=t;label=2;};    //    the source Γ_b
border f(t=L,0){x=t;y=t*(h1-h)/L+h;label=3;}; //    Γ_f
border d(t=h,0){x=0;y=t;label=4;};    //    Left impermeable Γ_d
int n=10;
mesh Th=buildmesh (a(L*n)+b(h1*n)
                   +f(sqrt(L^2+(h-h1)^2)*n)+d(h*n));
plot(Th,ps="dTh.eps");
```

The initial mesh



The problem is : find p and Ω such that :

$$\begin{cases} -\Delta p = 0 & \text{in } \Omega \\ p = y & \text{on } \Gamma_b \\ \dfrac{\partial p}{\partial n} = 0 & \text{on } \Gamma_d \cup \Gamma_a \\ \dfrac{\partial p}{\partial n} = \frac{q}{K} n_x & \text{on } \Gamma_f \quad (Neumann) \\ p = y & \text{on } \Gamma_f \quad (Dirichlet) \end{cases}$$

where the input water flux is $q = 0.02$, and $K = 0.5$. The velocity u of the water is given by $u = -\nabla p$.

algorithm

We use the following fix point method : let be, $k = 0$, $\Omega^k = \Omega$.
First step, we forgot the Neumann BC and we solve the problem :
Find p in $V = H^1(\Omega^k)$, such $p = y$ on $\Gamma_b^k \cup \Gamma_f^k$

$$\int_{\Omega^k} \nabla p \nabla p' = 0, \quad \forall p' \in V \text{ with } p' = 0 \text{ on } \Gamma_b^k \cup \Gamma_f^k$$

With the residual of the Neumann boundary condition we build a domain transformation $\mathcal{F}(x, y) = [x, y - v(x)]$ where v is solution of : $v \in V$, such than $v = 0$ on Γ_a^k (bottom)

$$\int_{\Omega^k} \nabla v \nabla v' = \int_{\Gamma_f^k} (\frac{\partial p}{\partial n} - \frac{q}{K} n_x) v', \quad \forall v' \in V \text{ with } v' = 0 \text{ sur } \Gamma_a^k$$

remark : we can use the previous equation to evaluate

$$\int_{\Gamma^k} \frac{\partial p}{\partial n} v' = - \int_{\Omega^k} \nabla p \nabla v'$$

The new domain is : $\Omega^{k+1} = \mathcal{F}(\Omega^k)$ Warning if is the movement is too large we can have triangle overlapping.

```
problem Pp(p,pp,solver=CG) =
    int2d(Th)( dx(p)*dx(pp)+dy(p)*dy(pp))
  + on(b,f,p=y) ;
problem Pv(v,vv,solver=CG) =
    int2d(Th)( dx(v)*dx(vv)+dy(v)*dy(vv))
  + on (a, v=0)
  + int1d(Th,f)(vv*((Q/K)*N.y- (dx(p)*N.x+dy(p)*N.y)));
while(errv>1e-6)
{   j++;  Pp;  Pv;    errv=int1d(Th,f)(v*v);
     coef = 1;
//    Here french cooking if overlapping see the example
    Th=movemesh(Th,[x,y-coef*v]);          //    deformation
}
```

Execute freeboundary.edp

Eigenvalue/ Eigenvector example

The problem, Find the first λ, u_λ such that :

$$a(u_\lambda, v) = \int_\Omega \nabla u_\lambda \nabla v = \lambda \int_\Omega u_\lambda v = \lambda b(u_\lambda, v)$$

the boundary condition is make with exact penalization : we put $1e30 = tgv$ on the diagonal term of the lock degree of freedom. So take Dirichlet boundary condition only with a variational form and not on b variational form , because we compute eigenvalue of

$$w = A^{-1} B v$$

Otherwise we get spurious mode.

Eigenvalue/ Eigenvector example code

```
...
fespace Vh(Th,P1);
macro Grad(u) [dx(u),dy(u),dz(u)]                    //    EOM
varf  a(u1,u2)= int3d(Th)(  Grad(u1)'*Grad(u2) +
on(1,u1=0) ;
varf b([u1],[u2]) = int3d(Th)(  u1*u2 ) ;          //    no BC
matrix A= a(Vh,Vh,solver=UMFPACK),
        B= b(Vh,Vh,solver=CG,eps=1e-20);

int nev=40; //    number of computed eigenvalue close to 0
real[int] ev(nev);                 //    to store nev eigenvalue
Vh[int] eV(nev);                   //    to store nev eigenvector
int k=EigenValue(A,B,sym=true,value=ev,vector=eV);
k=min(k,nev);
for (int i=0;i<k;i++)
   plot(eV[i],cmm="ev  "+i+" v =" + ev[i],wait=1,value=1);
```

Execute Lap3dEigenValue.edp Execute LapEigenValue.edp

Linear elasticty equation, weak form Let a

domain $\Omega \subset \mathbb{R}^d$ with a partition of $\partial\Omega$ in Γ_d, Γ_n.
Find the displacement $\boldsymbol{u}$ field such that :

$$-\nabla.\sigma(\boldsymbol{u}) = \boldsymbol{f} \text{ in } \Omega, \quad \mathbf{u = 0} \text{ on } \mathbf{\Gamma_d}, \quad \sigma(\mathbf{u}).\mathbf{n = 0} \text{ on } \mathbf{\Gamma_n} \quad (3)$$

Where $\varepsilon(\boldsymbol{u}) = \frac{1}{2}(\nabla \boldsymbol{u} + {}^t\nabla \boldsymbol{u})$ and $\sigma(\boldsymbol{u}) = \boldsymbol{A}\varepsilon(\boldsymbol{u})$ with $\boldsymbol{A}$ the linear positif operator on symmetric $d \times d$ matrix corresponding to the material propriety. Denote $V_{\boldsymbol{g}} = \{\boldsymbol{v} \in H^1(\Omega)^d / \boldsymbol{v}_{|\Gamma_d} = \boldsymbol{g}\}$.
The Basic displacement variational formulation with is : find $\boldsymbol{u} \in V_0(\Omega)$, such that

$$\int_\Omega \varepsilon(\boldsymbol{v}) : \boldsymbol{A}\varepsilon(\boldsymbol{u}) = \int_\Omega \boldsymbol{v}.\boldsymbol{f} + \int_\Gamma ((\boldsymbol{A}\varepsilon(\boldsymbol{u})).n).v, \quad \forall \boldsymbol{v} \in V_0(\Omega) \quad (4)$$

Optimization examples

The 2 membrane V.I. is find $(u^-, u^+) \in H^1(\Omega)^2$
$u^-_{|\Gamma} = g^-, u^+_{|\Gamma} = g^+$

$$\underset{u^- < u^+}{\text{argmin}} \ \frac{1}{2} \int_\Omega |\nabla u^-|^2 + |\nabla u^+|^2 - \int_\Omega u^- f^- + u^+ f^-$$

with $f^\pm, g^\pm$ are given function.
To solve just make a change of variable $u = u^+ - u^-, u > 0$ and $v = u^+ + u^-$, and we get a classical VI problem on u and and the Poisson on v.

So we use the algorithm of Primal-Dual Active set strategy as a semi smooth Newton Method HinterMuller , K. Ito, K. Kunisch SIAM J. Optim. V 13, I 3, 2002.

Run:VI-2-membrane-adap.edp

NLopt interface

```
load  "ff-NLopt"
...
if(kas==1)
   mini = nloptAUGLAG(J,start,grad=dJ,lb=lo,ub=up,IConst=IneqC,
            gradIConst=dIneqC,subOpt="LBFGS",stopMaxFEval=10000,
            stopAbsFTol=starttol);
else if(kas==2)
  mini = nloptMMA(J,start,grad=dJ,lb=lo,ub=up,stopMaxFEval=10000,
          stopAbsFTol=starttol);
else if(kas==3)
  mini = nloptAUGLAG(J,start,grad=dJ,IConst=IneqC,gradIConst=dIneqC,
          EConst=BC,gradEConst=dBC,
          subOpt="LBFGS",stopMaxFEval=200,stopRelXTol=1e-2);
else if(kas==4)
  mini = nloptSLSQP(J,start,grad=dJ,IConst=IneqC,gradIConst=dIneqC,
         EConst=BC,gradEConst=dBC,
         stopMaxFEval=10000,stopAbsFTol=starttol);
```

Run:VarIneq2.edp

Stochastic interface This algorithm works with a normal multivariate

distribution in the parameters space and try to adapt its covariance matrix using the information provides by the successive function evaluations. Syntaxe : `cmaes(J,u[],..)`

`seed=` Seed for random number generator

`initialStdDev=0.3` Value for the standard deviations of the initial covariance matrix (`val` is a real). the initial covariance matrix will be set to σI. initial distance between initial $|X - argmin| \sim sinitialStdDev$

`initialStdDevs=` Same as above except that the argument is an array allowing to set a value of the initial standard deviation for each parameter.

`stopTolFun=` Stops the algorithm if differences are smaller than ., default is 10^{-12}.

`stopTolFunHist=` Stops the algorithm if differences of the best values are smaller. 0=> (unused).

`stopTolX=.` Stopping criteria triggered if step sizes are smaller than , default is 0.

`stopTolXFactor=` Stopping criteria triggered when the standard deviation increases more than this value. The default value is 10^3.

`stopMaxFunEval=.` Stops the algorithm when the function . have been done. Set to $900(n+3)^2$ by default, n is dimension .

`stopMaxIter=.` Integer stopping the search, Unused by default.

`popsize=.` the population size. The default is $4 + \lfloor 3\ln(n) \rfloor$, Increasing this usually improves the global search capabilities.

`paramFile=` This filename `string` to pass all the parameters using this file.

Stochastic Exemple

```
load  "ff-cmaes"

real mini = cmaes(J,start,stopMaxFunEval=10000*(al+1),
                  stopTolX=1.e-4/(10*(al+1)),
                  initialStdDev=(0.025/(pow(100.,al))));
SSPToFEF(best1[],best2[],start);
```

Run:cmaes-VarIneq.edp Run:cmaes-oven.edp

```
load  "mpi-cmaes"

real mini = cmaesMPI(J,start,stopMaxFunEval=10000*(al+1),
                  stopTolX=1.e-4/(10*(al+1)),
                  initialStdDev=(0.025/(pow(100.,al))));
SSPToFEF(best1[],best2[],start);
```

remark, the FreeFem `mpicommworld` is used by default. The user can specify his own MPI communicator with the named parameter "`comm=`", see the MPI section of this manual for more informations about communicators in `FreeFem++`.

How to build matrix and RHS in //

```
int[int] refm=[1,1,2,1,3,1,4,1];
int[int] refu=[0,1];
mesh3 Th=buildlayers(square(nn,nn),nn,zbound=[0.,1.],labelmid=refm,
        labelup=refu,labeldown=refu);
                                //    to slip integral on each processor

//    change the region numbering to compute int on each proc
//    such that the number of element in one region is load balance
Th=change(Th,fregion= min(mpisize-1,int(nuTriangle* ccc+1e-10)));
real ccc = mpisize/ real(Th.nt) ;
int reg= mpirank;          //    set a reg int domain to the current proc.
//    end of trick
//    compute only on region reg = processor number
varf vlaplace(uh,vh) =                          //    definition de problem
      int3d(Th,reg)( uh*vh+ dt*(Grad(uh)'*Grad(vh)) )     //    bil. form
    + int3d(Th,reg)( dt*vh*f) + on(1,uh=g) ;
varf vmasse(u,v) = int3d(Th,reg)(u*v);
varf von1(uh,vh) = on(1,uh=1) ;
matrix AA = vlaplace(Vh,Vh,tgv=ttgv) ;
                          //    reduce the matrice to get the full matrice
                //    warning the tgv number is now mpisize*tgv for B.C.
```

The Loop & communication

```
matrix A;
mpiAllReduce(AA,A,mpiCommWorld,mpiSUM);
set(A,solver=sparsesolver,tgv=ttgv,sparams=ssparams);     //    factorize

matrix M = vmasse(Vh,Vh);
real [int] b(A.n),bb(A.n);
real[int] bcl= vlaplace(0,Vh,tgv=ttgv);                   //    The B.C vector
real[int] Gamma = von1(0,Vh,tgv=1);
for(int i=0;i<imax;i++)
{
  bb = M*uh[];
  bb += bcl;
  mpiAllReduce(bb,b,mpiCommWorld,mpiSUM);
  uh[] = A^-1*b;                                          //    resolution
  if(mpirank==0)
        cout « "  - time " « (i+1)*dt « " ||uh|| "
             « uh[].linfty « " ||b|| " « b.linfty « endl;
}
```

Run:Heat3D-mumps.edp Run:Heat3D-hips.edp

Poisson equation with Schwarz method

The sketch of the method is :

- Construction of the local partition of the unity
- Construction of the overlapping,
- the Schwarz alternating method,
- asynchronous Send/Receive (Hard)
- an GMRES version for Schwarz alternating method,
- Parallel visualisation for debugging

The explanation of the two script `examples++-mpi/MPIGMRES[23]D.edp`, a schwarz parallel with a complexity in $\sqrt[d]{n_p}$ of the number of process n_p :

Poisson equation with Schwarz method

- Introduction `Freefem++`
- new parallel solver
- Schwarz algorithm to solve Poisson
  - Construction of the local partition of the unity
  - Construction of the overlapping,
  - the Schwarz alternating method,
  - asynchronous Send/Receive (Hard)
  - an GMRES version for Schwarz alternating method,
  - Parallel visualisation for debugging

Poisson equation with Schwarz method

To solve the following Poisson problem on domain Ω with boundary Γ in $L^2(\Omega)$:

$$-\Delta u = f, \text{ in } \Omega, \text{ and } u = g \text{ on } \Gamma,$$

where f and g are two given functions of $L^2(\Omega)$ and of $H^{\frac{1}{2}}(\Gamma)$,
Let introduce $(\pi_i)_{i\in I}$ a regular partition of the unity of Ω in $N_p = \#I < +\infty$,

$$\text{q-e-d :} \qquad \pi_i \in \mathcal{C}^0(\Omega): \quad 0 \le \pi_i \le 1 \text{ and } \sum_{i\in I} \pi_i = 1.$$

Denote Ω_i the sub domain which is the support of π_i function and also denote Γ_i the boundary of Ω_i.
The parallel Schwarz method is Let $\ell = 0$ the iterator and a initial guest u^0 respecting the boundary condition (i.e. $u^0_{|\Gamma} = g$).

$$\forall i \in I \quad -\Delta u_i^\ell = f, \text{ in } \Omega_i, \text{ and } u_i^\ell = u^\ell \text{ on } \Gamma_i \setminus \Gamma, \; u_i^\ell = g \text{ on } \Gamma_i \cap \Gamma \tag{5}$$

$$u^{\ell+1} = \sum_{i\in I} \pi_i u_i^\ell \tag{6}$$

After discretization with the Lagrange finite element method, with a compatible mesh $\mathcal{T}_{hi}$ of Ω_i, i. e., the exist a global mesh $\mathcal{T}_h$ such that $\mathcal{T}_{hi}$ is include in $\mathcal{T}_h$.
Let us denote :

- V_{hi} the finite element space corresponding to domain Ω_i,
- $\mathcal{N}_{hi}$ is the set of the degree of freedom σ_i^k,
- $\mathcal{N}_{hi}^{\Gamma_i}$ is the set of the degree of freedom of V_{hi} on the boundary Γ_i of Ω_i,
- $\sigma_i^k(v_h)$ is the value the degree of freedom k,
- $V_{0hi} = \{v_h \in V_{hi} : \forall k \in \mathcal{N}_{hi}^{\Gamma_i}, \quad \sigma_i^k(v_h) = 0\}$,
- the conditional expression $a \ ? \ b : c$ is defined like in `C` of `C++` language by

$$a?b:c \equiv \begin{cases} \text{if } a \text{ is true then return } b \\ \text{else return } c \end{cases} .$$

Remark we never use finite element space associated to the full domain Ω because it to expensive.

We have to defined to operator to build the previous algorithm :
We denote $u^{\ell}_{h\,|i}$ the restriction of u^{ℓ}_{h} on $V_{h\,i}$, so the discrete problem on Ω_i of problem (5) is find $u^{\ell}_{h\,i} \in V_{h\,i}$ such that :

$$\forall v_{h\,i} \in V_{0i} : \int_{\Omega_i} \nabla v_{h\,i}.\nabla u^{\ell}_{h\,i} = \int_{\Omega_i} f v_{h\,i}, \quad \forall k \in \mathcal{N}^{\Gamma_i}_{hi} \ : \ \sigma^k_i(u^{\ell}_{h\,i}) = (k \in \Gamma) \ ? \ g^k_i : \sigma^k_i(u^{\ell}_{h\,|i})$$

where g^k_i is the value of g associated to the degree of freedom $k \in \mathcal{N}^{\Gamma_i}_{hi}$.

In FreeFem++, it can be written has with U is the vector corresponding to $u^{\ell}_{h\,|\,i}$ and the vector U1 is the vector corresponding to $u^{\ell}_{h\,i}$ is the solution of :

```
real[int] U1(Ui.n)   ,    b= onG .* U;
b  = onG? b : Bi ;        U1 = Ai^-1*b;
```

where onG$[i] = (i \in \Gamma_i \setminus \Gamma)?1 : 0$, and Bi the right of side of the problem, are defined by

```
fespace Whi(Thi,P2);                                          //    def of the Finite element space.
varf vPb(U,V)= int3d(Thi)(grad(U)'*grad(V)) + int3d(Thi)(F*V) +on(1,U=g) + on(10,U=G);
varf vPbon(U,V)=on(10,U=1);
matrix Ai = vPb(Whi,Whi,solver=sparsesolver);
real[int] onG = vPbon(0,Whi);
real[int] Bi=vPb(0,Whi);
```

where the freefem++ label of Γ is 1 and the label of $\Gamma_i \setminus \Gamma$ is 10.

To build the transfer/update part corresponding to (6) equation on process i, let us call njpart the number the neighborhood of domain of Ω_i (i.e : π_j is none 0 of Ω_i), we store in an array jpart of size njpart all this neighborhood. Let us introduce two array of matrix, Smj[j] to defined the vector to send from i to j a neighborhood process, and the matrix $rMj[j]$ to after to reduce owith neighborhood j domain.
So the tranfert and update part compute $v_i = \pi_i u_i + \sum_{j \in J_i} \pi_j u_j$ and can be write the freefem++ function Update :

```
func bool Update(real[int] &ui, real[int] &vi)
{ int n= jpart.n;
  for(int j=0;j<njpart;++j)  Usend[j][]=sMj[j]*ui;
  mpiRequest[int] rq(n*2);
  for (int j=0;j<n;++j) Irecv(processor(jpart[j],comm,rq[j  ]), Ri[j][]);
  for (int j=0;j<n;++j) Isend(processor(jpart[j],comm,rq[j+n]), Si[j][]);
  for (int j=0;j<n*2;++j) int k= mpiWaitAny(rq);
  vi = Pii*ui;//    set to πi ui // apply the unity local partition .
   for(int j=0;j<njpart;++j)  vi += rMj[j]*Vrecv[j][];                    //    add πj uj
 return true; }
```

where the buffer are defined by :

```
InitU(njpart,Whij,Thij,aThij,Usend)  //    defined the send buffer
InitU(njpart,Whij,Thij,aThij,Vrecv)  //    defined the revc buffer
```

with the following macro definition :

```
macro InitU(n,Vh,Th,aTh,U) Vh[int] U(n); for(int j=0;j<n;++j) {Th=aTh[j];  U[j]=0;}      //   EOM
```

Finally you can easily accelerate the fixe point algorithm by using a parallel GMRES algorithm after the introduction the following affine $\mathcal{A}_i$ operator sub domain Ω_i.

```
func real[int] Ai(real[int]& U) {
 real[int] V(U.n)    ,  b= onG .* U;
  b  = onG? b : Bi;
  V = Ai^-1*b;
  Update(V,U);
  V -= U;   return V; }
```

Where the parallel MPIGMRES or MPICG algorithm is just a simple way to solve in parallel the following $A_i x_i = b_i, i = 1, .., N_p$ by just changing the dot product by reduce the local dot product of all process with the following MPI code :

```
template<class R> R ReduceSum1(R s,MPI_Comm * comm)
{   R r=0;
    MPI_Allreduce( &s, &r, 1 ,MPI_TYPE<R>::TYPE(),   MPI_SUM,  *comm );
    return r; }
```

This is done in MPIGC dynamics library tool.

We have all ingredient to solve in parallel if we have et the partitions of the unity. To build this partition we do : the initial step on process 1 to build a coarse mesh, ${\mathcal{T}_h}^*$ of the full domain, and build the partition π function constant equal to i on each sub domain $\mathcal{O}_i, i = 1, .., N_p$, of the grid with the Metis graph partitioner and on each process i in $1.., N_p$ do

1. Broadcast from process 1, the mesh $\mathcal{T}_h{}^*$ (call `Thii` in freefem++ script), and π function,
2. remark that the characteristic function $\mathbb{1}_{\mathcal{O}_i}$ of domain $\mathcal{O}_i$, is defined by $(\pi = i)?1 : 0$,
3. let us call Π_P^2 (resp. Π_V^2) the L^2 project on P_h^* (resp. V_h^* the space P1 one $\mathcal{T}_h{}^*$) where P_h^* (resp. V_h^*) is F.E. space P0 (reps. P1) on $\mathcal{T}_h{}^*$. Let us introduce $\pi_i^* = (\Pi_V^2 \Pi_C^2)^m \mathbb{1}_{O_i}$ with m is a the overlaps size on the coarse mesh, now

$$\mathcal{O}_i = supp(\pi_i^*),$$

and so the partition of the unity is simply defined by

$$\pi_i = \pi_i^* \big(\sum_{j \in I} \pi_j^*\big)^{-1}.$$

The set J_i of neighborhood of the domain Ω_i, V_{hi}, `jpart` and `njpart` can be defined with :

```
Vhi pii=π_i^* ;  Vhi[int] pij(npij);     //    π_j|Ω_i = pii + Σ_j pij[j]
int[int] jpart(npart);   int njpart=0;     Vhi sumphi =  π_i^*  ;
for (int i=0;i<npart;++i)
  if(i != ipart ) {
     if(int3d(Thi)( π_j^*)>0) {  pij[njpart]=π_j^*;
       sumphi[] += pij[njpart][];   jpart[njpart++]=i;}}}
     pii[]=pii[] ./ sumphi[];
for (int j=0;j<njpart;++j) pij[j][] = pij[j][] ./ sumphi[];
jpart.resize(njpart);
```

1. We call $\mathcal{T}_{h_{ij}}^*$ the sub mesh part of $\mathcal{T}_{hi}$ where π_j are none zero. and tank to the function `trunc`,

```
for(int jp=0;jp<njpart;++jp)
   aThij[jp] =trunc(Thi,pij[jp]>1e-10,label=10);
```

2. At this step we have all on the coarse mesh , so we can build the final fine mesh by splitting all meshes : `Thi`, `Thij[j]`,`Thij[j]` with freefem++ `trunc` mesh function.

3. The construction of the send/recv matrices `sMj` and `rMj` : can done with this code :

```
mesh3 Thij=Thi;
fespace Whij(Thij,Pk);                          //    var. fespace meshes..
matrix Pii; Whi wpii=pii; Pii = wpii[];         //    Diag. matrix ×π_i
matrix[int] sMj(njpart), rMj(njpart);           //    M send/rend case..
 for(int jp=0;jp<njpart;++jp)
   { int j=jpart[jp]; Thij = aThij[jp];   //    set mesh to change Whij,Whij
     matrix I = interpolate(Whij,Whi);            //    Whij ← Whi
     sMj[jp] = I*Pii;
     rMj[jp] = interpolate(Whij,Whi,t=1); }}     //    Whij to Whi
```

To build a not to bad application, I have add code tout change variable from parameter value with the following code

```
include "getARGV.idp"
verbosity=getARGV("-vv",0);
...
bool gmres=getARGV("-gmres",1);
bool dplot=getARGV("-dp",0);
```

And small include to make graphic in parallel of distribute solution of vector u on mesh T_h with the following interface :

```
include "MPIplot.idp"
func bool  plotMPIall(mesh3 &Th,real[int] & u,string  cm)
{ PLOTMPIALL(mesh3,Pk, Th, u,{cmm=cm,nbsio=3});            //    MPI plot
 return 1;}
```

A Parallel Numerical experiment on laptop

We consider first example in an academic situation to solve Poisson Problem on the cube $\Omega =]0,1[^3$

$$-\Delta u = 1, \text{ in } \Omega; \qquad u = 0, \text{ on } \partial\Omega. \tag{7}$$

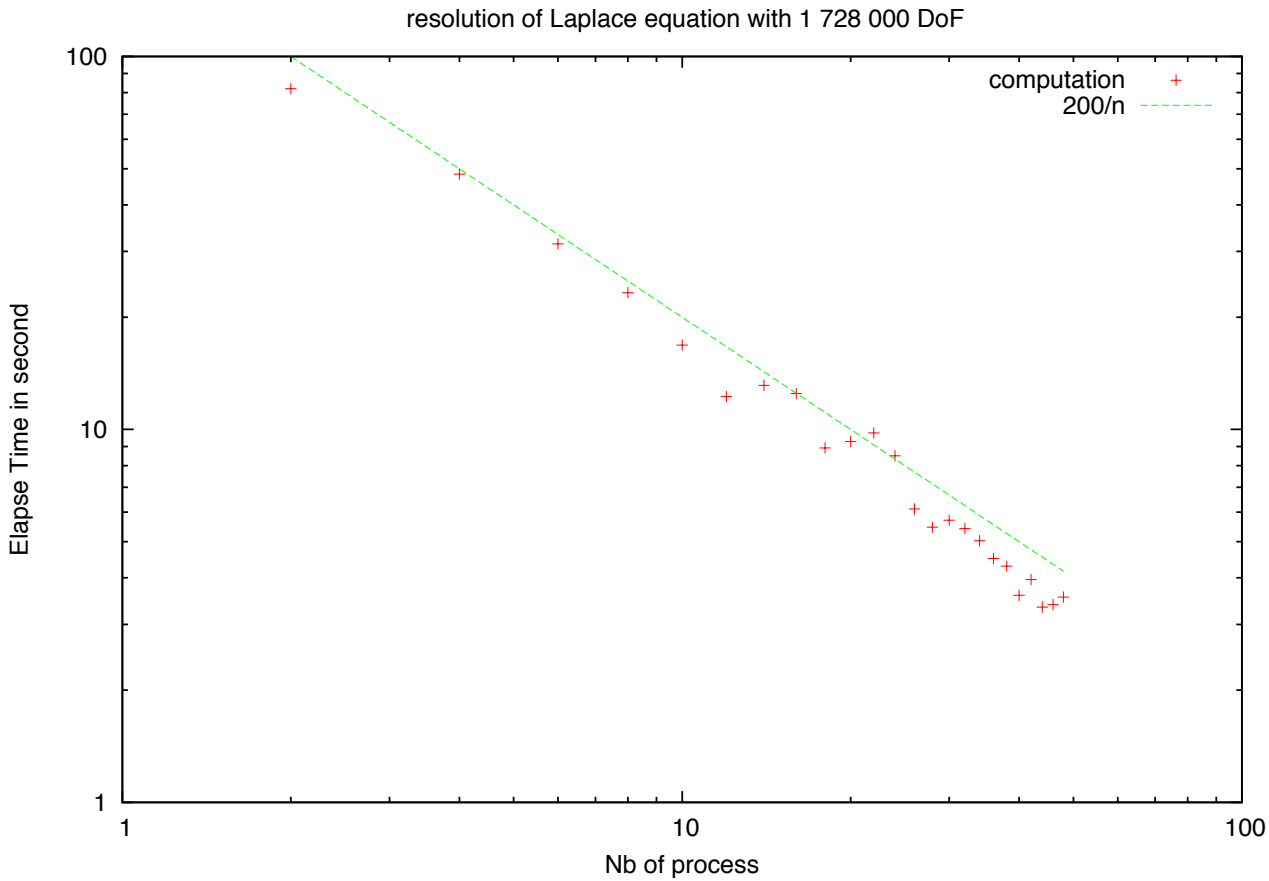
With a cartesian meshes $\mathcal{T}_{hn}$ of Ω with $6n^3$ tetrahedron, the coarse mesh is $\mathcal{T}^*_{hm}$, and m is a divisor of n.
We do the validation of the algorithm on a Laptop Intel Core i7 with 4 core at 2.66 Ghz with $8Go$ of RAM DDR3 at 1067 Mhz,

Run:DDM-Schwarz-Lap-3d.edp Run:DDM-Schwarz-Lap-2dd.edp
Run:DDM-Schwarz-Lame-2d.edp Run:DDM-Schwarz-Lame-3d.edp

A Parallel Numerical experiment, scalability

We consider first example in an academic situation to solve Poisson Problem on the cube $\Omega =]0,1[^3$

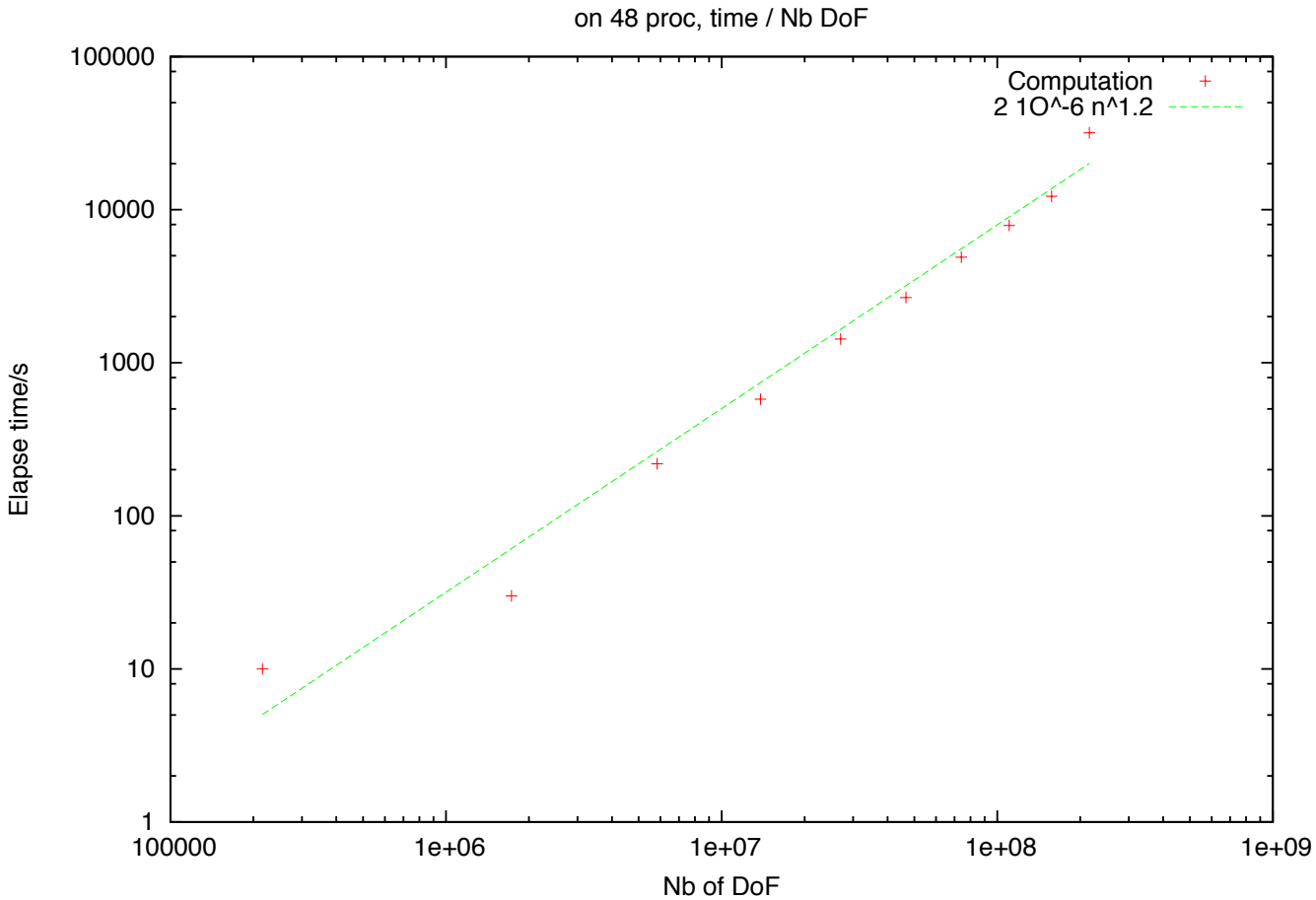
$$-\Delta u = 1, \text{ in } \Omega; \qquad u = 0, \text{ on } \partial\Omega. \tag{8}$$



A Parallel Numerical experiment, complexity

$$-\Delta u = 1, \text{ in } \Omega; \qquad u = 0, \text{ on } \partial\Omega. \tag{9}$$

On 48 proc, test on number of Dof from $216\,10^3$ to $216\,10^6$



Linear Lame equation, weak form

Let a domain $\Omega \subset \mathbb{R}^d$ with a partition of $\partial\Omega$ in Γ_d, Γ_n.
Find the displacement $\boldsymbol{u}$ field such that :

$$-\nabla.\sigma(\boldsymbol{u}) = \boldsymbol{f} \text{ in } \Omega, \quad \mathbf{u = 0} \text{ on } \mathbf{\Gamma_d}, \quad \sigma(\mathbf{u}).\mathbf{n = 0} \text{ on } \mathbf{\Gamma_n} \quad (10)$$

Where $\varepsilon(\boldsymbol{u}) = \frac{1}{2}(\nabla \boldsymbol{u} + {}^t\nabla \boldsymbol{u})$ and $\sigma(\boldsymbol{u}) = \boldsymbol{A}\varepsilon(\boldsymbol{u})$ with $\boldsymbol{A}$ the linear positif operator on symmetric $d \times d$ matrix corresponding to the material propriety. Denote $V_{\boldsymbol{g}} = \{\boldsymbol{v} \in H^1(\Omega)^d / \boldsymbol{v}_{|\Gamma_d} = \boldsymbol{g}\}$.
The Basic displacement variational formulation is : find $\boldsymbol{u} \in V_0(\Omega)$, such that

$$\int_\Omega \varepsilon(\boldsymbol{v}) : \boldsymbol{A}\varepsilon(\boldsymbol{u}) = \int_\Omega \boldsymbol{v}.\boldsymbol{f} + \int_\Gamma ((\boldsymbol{A}\varepsilon(\boldsymbol{u})).n).v, \quad \forall \boldsymbol{v} \in V_0(\Omega) \quad (11)$$

Linear elasticty equation, in FreeFem++ The

finite element method is just : replace V_g with a finite element space, and the FreeFem++ code :

```
load "medit"   include "cube.idp"
int[int]  Nxyz=[20,5,5];
real [int,int]  Bxyz=[[0.,5.],[0.,1.],[0.,1.]];
int [int,int]  Lxyz=[[1,2],[2,2],[2,2]];
mesh3 Th=Cube(Nxyz,Bxyz,Lxyz);
                                                    //    Alu ...
real rhoAlu = 2600, alu11=  1.11e11 , alu12 = 0.61e11 ;
real alu44= (alu11-alu12)*0.5;
func Aalu = [  [alu11, alu12,alu12,   0.   ,0.   ,0.  ],
               [alu12, alu11,alu12,   0.   ,0.   ,0.   ],
               [alu12, alu12,alu11,   0.   ,0.   ,0.   ],
               [0. ,   0. ,   0. ,    alu44,0.   ,0.   ],
               [0. ,   0. ,   0. ,    0.   ,alu44,0.   ],
               [0. ,   0. ,   0. ,    0.   ,0.   ,alu44]
];
real gravity = -9.81;
```

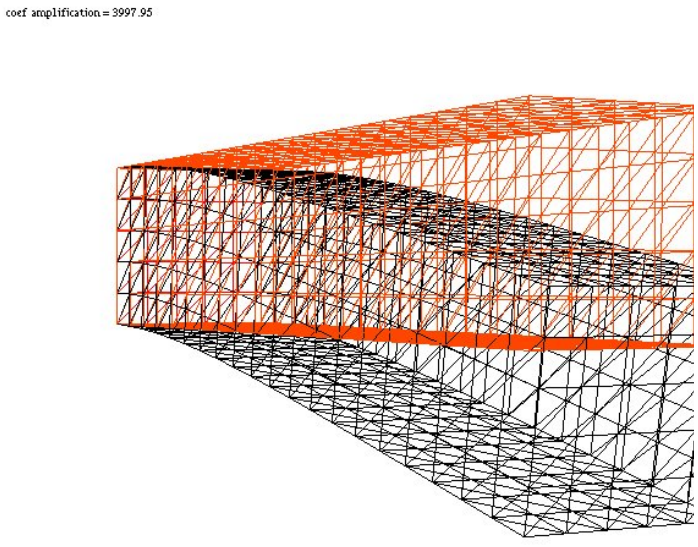
Linear elasticty equation, in FreeFem++

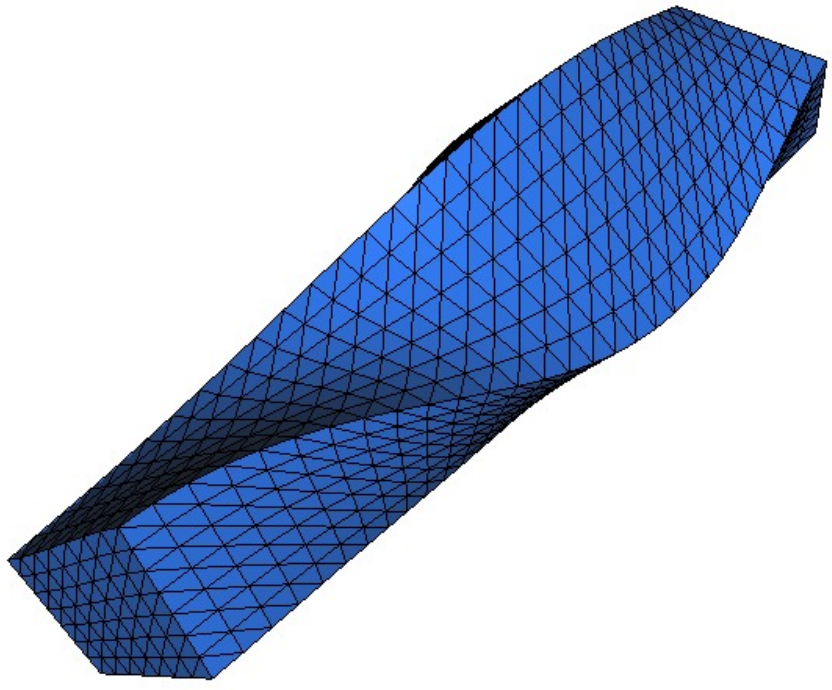
The problem code :

```
fespace Vh(Th,[P1,P1,P1]);
Vh [u1,u2,u3], [v1,v2,v3];
macro Strain(u1,u2,u3) [ dx(u1), dy(u2),dz(u3), (dz(u2)
  +dy(u3)), (dz(u1)+dx(u3)), (dy(u1)+dx(u2)) ]    //    EOM
solve Lame([u1,u2,u3],[v1,v2,v3])=
  int3d(Th)(  Strain(v1,v2,v3)'*(Aalu*Strain(u1,u2,u3))  )
            - int3d(Th) ( rhoAlu*gravity*v3)
            + on(1,u1=0,u2=0,u3=0) ;

real coef= 0.1/u1[].linfty;  int[int] ref2=[1,0,2,0];
mesh3 Thm=movemesh3(Th,
     transfo=[x+u1*coef,y+u2*coef,z+u3*coef],
     label=ref2);
plot(Th,Thm, wait=1,cmm="coef  amplification = "+coef );
medit("Th-Thm",Th,Thm);
```

Lame equation / figure





Run:beam-3d.edp Run:beam-EV-3d.edp Run:beam-3d-Adapt.edp

The formulation Primitive equation, toy Problem.

Let ω a domain of $\mathbb{R}^2$ and Ω of $\mathbb{R}^2$ such that $\Omega = \{(x,y,z)/(x,y) \in \omega, z_0(x,y) < z < z_d(x,y)\}$ where z_0,z_d are the z of top, and the z of the deep of the domain Ω, corresponding to Γ_0 (the top) and Γ_d (the deep), and Γ_v is the vertical boundary.

Denote, u_H is the horizontal velocity, ∇_H horizontal gradient ; the mean value in z : $\overline{p}^z = \int_z p / \int_z 1$, ...

The Stokes Primitive equation in $\Omega \subset \mathbb{R}^3$ find $\boldsymbol{u}$, velocity, p the pressure, such that p is independent of z and :

$$\nu_H \Delta_H \boldsymbol{u}_H + \nu_z \Delta_z \boldsymbol{u}_H - \overline{\nabla_H p}^z = \mathbf{f}$$

$$\nabla . \boldsymbol{u} = 0$$

The formulation Primitive equation, BC

B.C :

- $u_3 = 0$ on Γ_0 and Γ_d
- $\boldsymbol{u}_H$ are given on Γ_v and Γ_d .
- we use a wall law at the Γ_0 .

remark, due to B.C. on u_3, we have $\overline{\partial_z u_3}^z = 0$ so the equation is well pose , because $\partial_z(u_3) = -\nabla_H \boldsymbol{u}_H$ and u_3 know on z_d, gives $\overline{\nabla_H . \boldsymbol{u}_H}^z = 0$,

The formulation Primitive equation, variational form

find $\boldsymbol{u}_H \in \boldsymbol{V}_g$ and $\overline{p}^z \in W$ such that $\forall \boldsymbol{v}_H \in \boldsymbol{V}_0, \overline{q}^z \in W$:

$$\int_\Omega \nu_H \nabla_H \boldsymbol{u}_H . \nabla_H \boldsymbol{v}_H + \nu_z \nabla_z \boldsymbol{u}_H . \nabla_z \boldsymbol{v}_H$$
$$- \int_\omega \overline{\nabla_H . \boldsymbol{u}_H}^z \overline{q}^z$$
$$- \int_\omega \overline{\nabla_H . \boldsymbol{v}_H}^z \overline{p}^z \quad = \quad - \int_\Omega f \boldsymbol{v}_H + ...$$

where Ω is the horizontal trace of Ω. and u_3 is reconstruct from $\boldsymbol{u}_H$ with :

$$\forall v_3 \in V_0, \quad \int_\Omega \partial_z u_3 \partial_z v_3 = - \int_\Omega \nabla_H . \boldsymbol{u}_H \partial_z v_3$$

with $V_0 = \{v \in H^1(\Omega), v_{|\Gamma_d} = 0, v_{|\Gamma_0} = 0\}$.

The formulation Primitive equation, FreeFem++

$\boldsymbol{V}$ is `fespace(Th,[P2,P2])`,
W is `fespace(Thp,[P1],periodic[[G0,x,y],[[Gd,x,y]))`,
where Thp is a mesh with one layer to build a 2d FE element space because No interpolation between 2d and 3d `fespace`.

```
mesh Th2D=square(nn,nn);
int[int] refm=[1,1,2,1,3,10,4,1], refu=[0,3], refd=[0,2];
fespace Vh2(Th2D,P2);
mesh3 Th=buildlayers(Th2D,nn,zbound=[-1.,0.],
          labelmid=refm,labelup=refu,labeldown=refd);
mesh3 Thp=buildlayers(Th2D,1,zbound=[-1.,0.],
          labelmid=refm,labelup=refu,labeldown=refd);

fespace Vh(Th,[P2,P2,P1]);                    //    for u1,u2, p
fespace Uh(Th,[P2]);
fespace Wh(Th,[P2,P2]);                         //    for u1,u2
fespace Ph2(Thp,[P1],periodic=[[2,x,y],[3,x,y]]);          //   2d
```

Injection , FreeFem++

The Injection $I_h : (u_1, u_2, \overline{p}^z) \mapsto (u_1, u_2, p)$

```
matrix Ih;     //    matrix du go from 3d,3d,2d -> 3d,3d,3d
{
                          //    numbering first u1,u2 second p.
int[int] V2Pc=[-1,-1,0],  V2Wc=[0,1,-1];       //    u1, u2

matrix IPh =interpolate(Vh,Ph2,U2Vc=V2Pc);
                                                //   ΠVh(v ∈ Ph2)
matrix IWh =interpolate(Vh,Wh,U2Vc=V2Wc);
int n = IPh.n;
int m =  IWh.m+IPh.m;
int[int] nuV=0:n-1;                                  //    id
int[int] nuW=0:IWh.m-1;                             //    u1,u2
int[int] nuP=IWh.m:m-1;                             //    p (2d)
matrix JJh = IPh(nuV^-1,nuP^-1);
matrix IIh = IWh(nuV^-1,nuW^-1);
Ih =  JJh + IIh;
}
```

Matrix , FreeFem++

```
varf vPrimitive([u1,u2,p3],[v1,v2,q3]) =
      int3d(Th)(
          nuH * ( GradH(u1)'*GradH(v1)
                + GradH(u2)'*GradH(v2)  )
        + nuz * (dz(u1)*dz(v1) + dz(u2)*dz(v2) )
        - div2(u1,u2)* q3                          //   - du3*q3
        - div2(v1,v2)* p3                          //   - dv3*p3
        - 1e-6*p3*q3     //    mean value for p3 is zero ..
      )                                            //    bil. form
    + int2d(Th,3) ( (u1*v1 + u2*v2)*0.00001)
    + int2d(Th,3)  ( v1*0.0001 )                          //
    + on(10,u1=x*(1-x),u2=0)+ on(1,u1=0,u2=0)
    + on(2,u1=0,u2=0);
matrix AA = vPrimitive(Vh,Vh,tgv=ttgv) ;
matrix A = AA*Ih;  A = Ih'*A;          //    A = Ih'*A*Ih .
```

Solve , FreeFem++

```
real[int] bb= vPrimitive(0,Vh,tgv=ttgv) ;
real[int] b=Ih'*bb;
real[int]  uu= A^-1*b;
Vh [u1,u2,p3] ;
u1[] = Ih*uu;

varf vU3(u3,v3)=int3d(Th)( dz(u3)*dz(v3) )
        + on(3,2,u3=0);
varf vU3U([u1,u2],[v3])=
     int3d(Th)(-1.*div2(u1,u2)*dz(v3));

matrix A3= vU3(Uh,Uh), B3= vU3U(Vh,Uh);

real[int] b3 = B3*u1[];
Uh u3;  u3[] = A3^-1*b3;
```

Run:primitive-eq.edp

Some examples 2D

- Execute BlackScholes2D.edp
- Execute cavityNewtow.edp
- Execute Poisson-mesh-adap.edp
- Execute Micro-wave.edp
- Execute wafer-heating-laser-axi.edp
- Execute nl-elast-neo-Hookean.edp
- Execute Stokes-eigen.edp
- Execute fluid-Struct-with-Adapt.edp
- Execute optim-control.edp
- Execute VI-2-menbrane-adap.edp

Conclusion/Future

Freefem++ v3 is

- very good tool to solve non standard PDE in 2D/3D
- to try new domain decomposition domain algorithm

The future we try to do :

- Build more graphic with VTK, paraview , ... (in progress)
- Add Finite volume facility for hyperbolic PDE (just begin C.F. FreeVol Projet)
- 3d anisotrope mesh adaptation
- automate the parallel tool

Thank for you attention.