Taking advantage seamlessly of new heterogeneous architectures, with a C++ DSEL to implement lowest-order methods for diffusive problem on general meshes

Jean-Marc Gratien

Department of Applied math and computer science
IFP New Energy, Rueil-Malmaison France
jean-marc.gratien@ifpen.fr

PDESoft 2014, Heidelberg, July 16th 2014

Introduction

- IFP New Energy
  - Technology for energy and environnement
  - Reservoir and Bassin modeling
  - CO2 storage
  - Combustion, engine modeling
  - ...
- A joined work with Christophe Prud'Homme of IRMA laboratory, Universtité de Strasbourg, France
- work funded with the support of the ANR project HAMM :
  Hybrid **A**rchitecture and **M**ulti-scale **M**ethods

Outline

Context : Increasing complexity

Example : CO2 sequestration

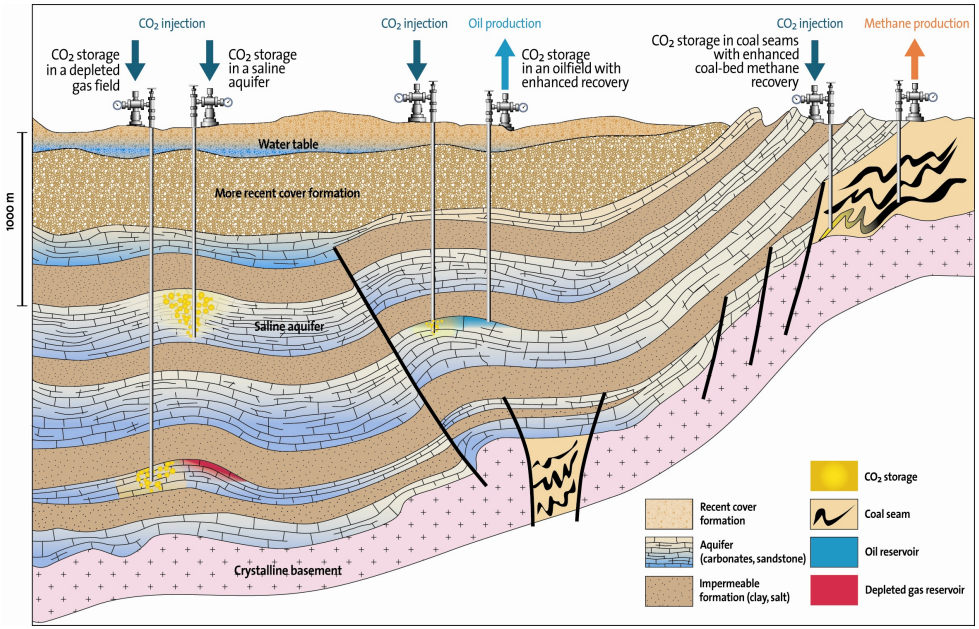


Figure: CO2 storage simulation

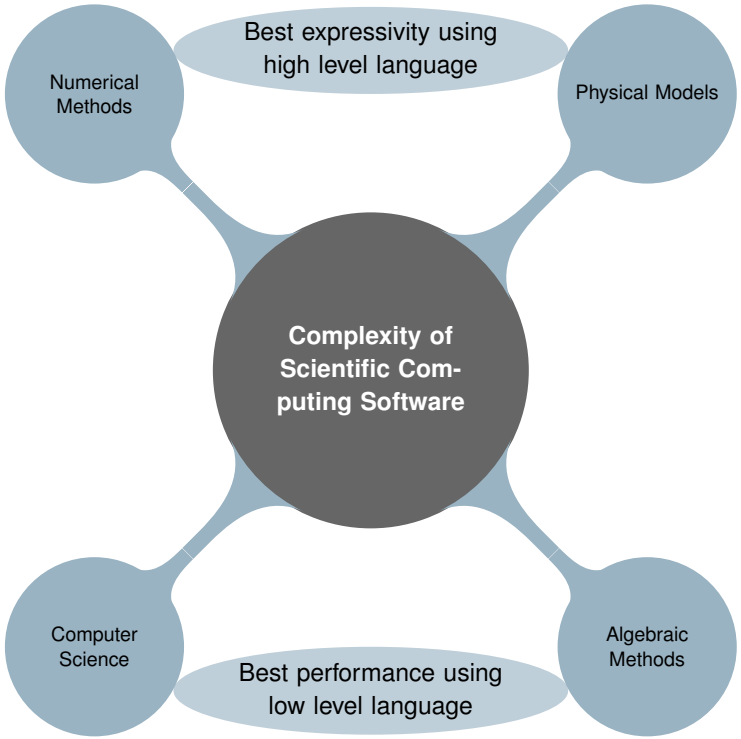
Various physical models :

- Bassin modeling ;
- Reservoir modeling ;
- Well modeling ;
- Reactive transport models ;
- Chemistry, Geo-mecanics

Various numerical methods :

- FV/FE methods ;
- Non linear solvers ;
- Coupling/Splitting methods ;
- Space/Time stepping. . .

Context : complexity in computer science

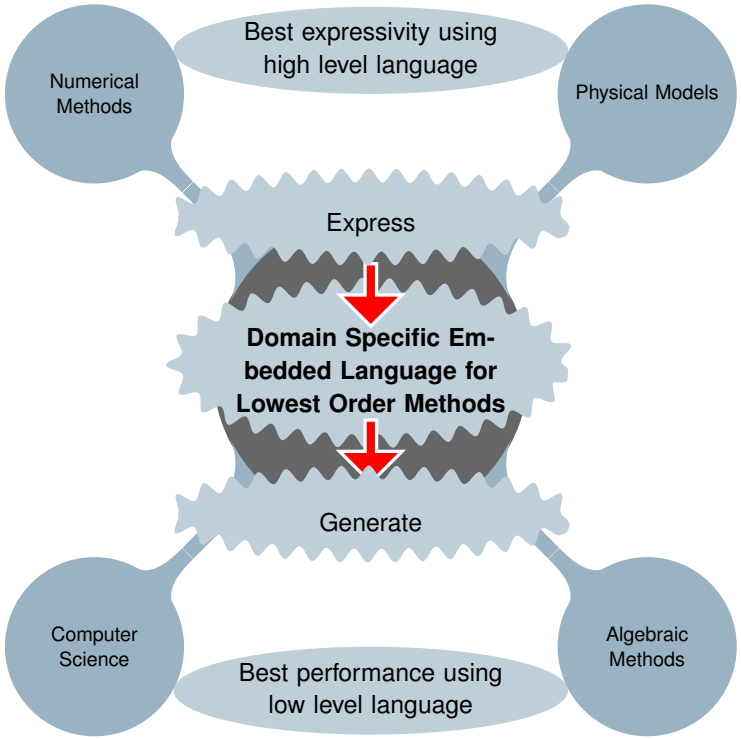


Complexity Types

- Algebraic
- Numerical
- Models
- Computer science

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

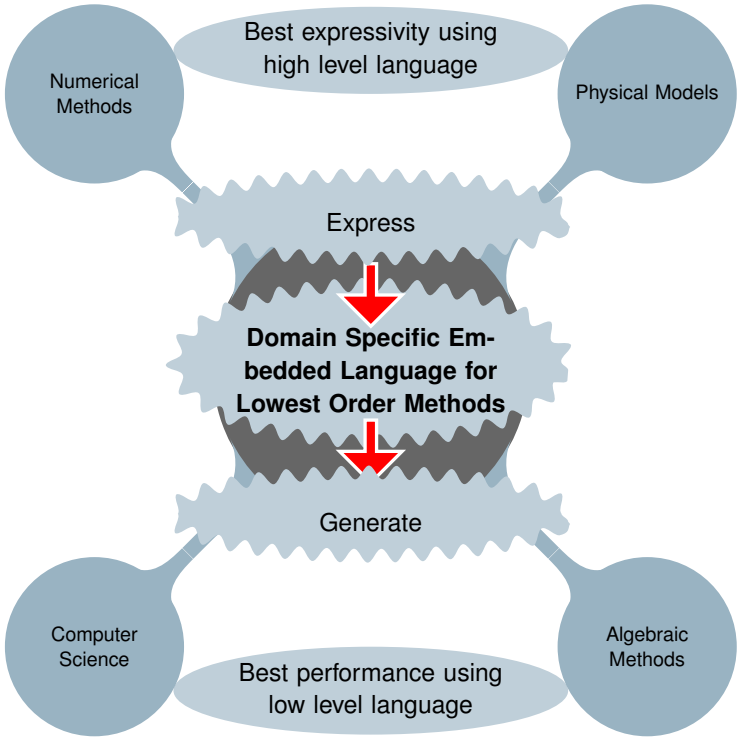
Context : complexity in computer science



Generative paradigm

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

Context : complexity in computer science



Definitions

- A *Domain Specific Language* (DSL) is a programming or specification language dedicated to a particular domain, problem and/or a solution technique
- A *Domain Specific Embedded Language* (DSEL) is a DSL integrated into another programming language (e.g. C++)

State of art

Existing framework solutions

State of art :

- Frameworks to manage parallelism, mesh and linear solver: Arcane, Dune, Trilinos, Petsc. . .
- Frameworks for Finite Element or Galerkin methods :
  - based on an existing unified formalism
  - DSL solution: FreeFem++, GetDP, GetFem++, Fenics
  - DSEL solution: Feel++, Sundance

Motivation of our reseach work :

- No framework for lowest order methods :
  - Finite Volume, Mimetic Finite Difference, Mixed/Hybrid Finite methods ;
- An unified perspective to describe these methods is emerging ;
- What about extending DSEL solutions for FE/DG methods to lowest order methods?

Mathematical framework : Variational formulation

Variational formulation

A unified perspective for FE/DG/FV methods :

- inspired from FreeFEM++, FEnics, Feel++ and Sundance libraries;
- Based on variational formulations of PDEs:
  1. Functional spaces : Trial space U_h, Test space V_h
  2. trial and test functions $(u_h, v_h) \in U_h \times V_h$
  3. Bilinear form $a_h(u_h, v_h)$, linear form $b_h(v_h)$
  4. Find $u_h \in U_h$ so that $\forall v_h \in V_h$: $a_h(u_h, v_h) = b_h(v_h)$

Key ingredients to design Functional Spaces for FV methods :

- **Mesh** ;
- **Space of Degree Of Freedoms** (DOFs) ;
- **Linear combinations** and linear systems (matrices, vectors)
- **Functional spaces, test and trial functions**.

Mathematical framework : Variational formulation

Exemples of linear and bilinear forms

- Diffusive problems: $\nabla\cdot(-\kappa\nabla u) = f$

  $a(u,v) \stackrel{\text{def}}{=} \int_\Omega \kappa\nabla u\cdot\nabla v$

  $b(v) \stackrel{\text{def}}{=} \int_\Omega f\, v$

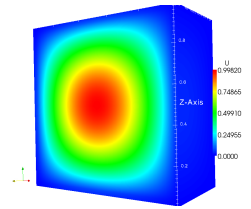


Figure: Diffusion problem

- Linear elasticity: $-\nabla\cdot\sigma(\mathbf{u}) = \mathbf{f}$

  $\varepsilon(\mathbf{v}) \stackrel{\text{def}}{=} \frac{1}{2}(\nabla\mathbf{v} + \nabla\mathbf{v}^T)$

  $\sigma(\mathbf{v}) \stackrel{\text{def}}{=} 2\mu\varepsilon(\mathbf{v}) + \lambda\nabla\cdot\mathbf{v}I_d$

  $a(\mathbf{u},\mathbf{v}) \stackrel{\text{def}}{=} \int_\Omega \sigma(\mathbf{u}) : \varepsilon(\mathbf{v})$

  $b(\mathbf{v}) \stackrel{\text{def}}{=} \int_\Omega \mathbf{f}\cdot\mathbf{v}$

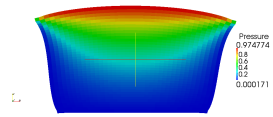


Figure: Elasticity problem

- Stokes problem: $-\triangle\mathbf{u} + \nabla p = \mathbf{f}$ and $\nabla\cdot\mathbf{u} = 0$

  $a((\mathbf{u},p),(\mathbf{v},q)) \stackrel{\text{def}}{=} \int_\Omega \nabla\mathbf{u}:\nabla\mathbf{v} + \int_\Omega p\nabla\cdot v + \int_\Omega \nabla q\cdot\mathbf{u}$

  $b((\mathbf{v},q)) \stackrel{\text{def}}{=} \int_\Omega \mathbf{f}\cdot\mathbf{v}$

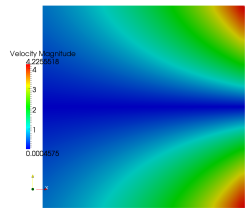


Figure: Stokes problem

Unified mathematical framework for FV methods

Algebraic and functional space

- **Mesh** : Ω domain of $\mathscr{R}^d$, $\mathscr{T}_h = \{\tau\}$ and $\mathscr{F}_h = \{\sigma\}$ mesh representation of Ω
- **Space of DOFs**

$$\mathbb{T}_h \stackrel{\text{def}}{=} \mathbb{R}^{\mathscr{T}_h}, \qquad \mathbb{F}_h \stackrel{\text{def}}{=} \mathbb{R}^{\mathscr{F}_h},$$

  $\mathbb{V}_h$: the space of degree of freedoms. DOFs indexed by elements of $\mathscr{T}_h$ or by elements of $\mathscr{T}_h \cup \mathscr{F}_h$.
- **Linear combinations** :
  - stencils indexed by mesh elements,
  - contribution to matrix rows, cols or local submatrices,
  - link between mesh variables and vectors of DOFs.
- **Functional space** :

  A mapping of every vector of DOFs onto a piecewise affine function
  Recover different families of lowest order methods :
  - Cell centered finite volume (CCFV),
  - Cell centered galerkine (CCG),
  - Mimetic finite difference (MFD),
  - Mixed hybrid finite volume (MHFV).

DSEL design for lowest order methods : Key ingredients

Key ingredients to design a DSEL :

1. Meta-programming :
   - programs that transform types at compile time
2. Generic programming :
   - design generic components composed of abstract programs with generic types
3. Generative programming :
   - generate concrete programs, transforming types to create concrete types to use with abstract programs of generic components
4. Expression template :
   - expression tree representation of a problem
   - tools to describe, parse and evaluate a tree

DSEL design for lowest order methods : Principles

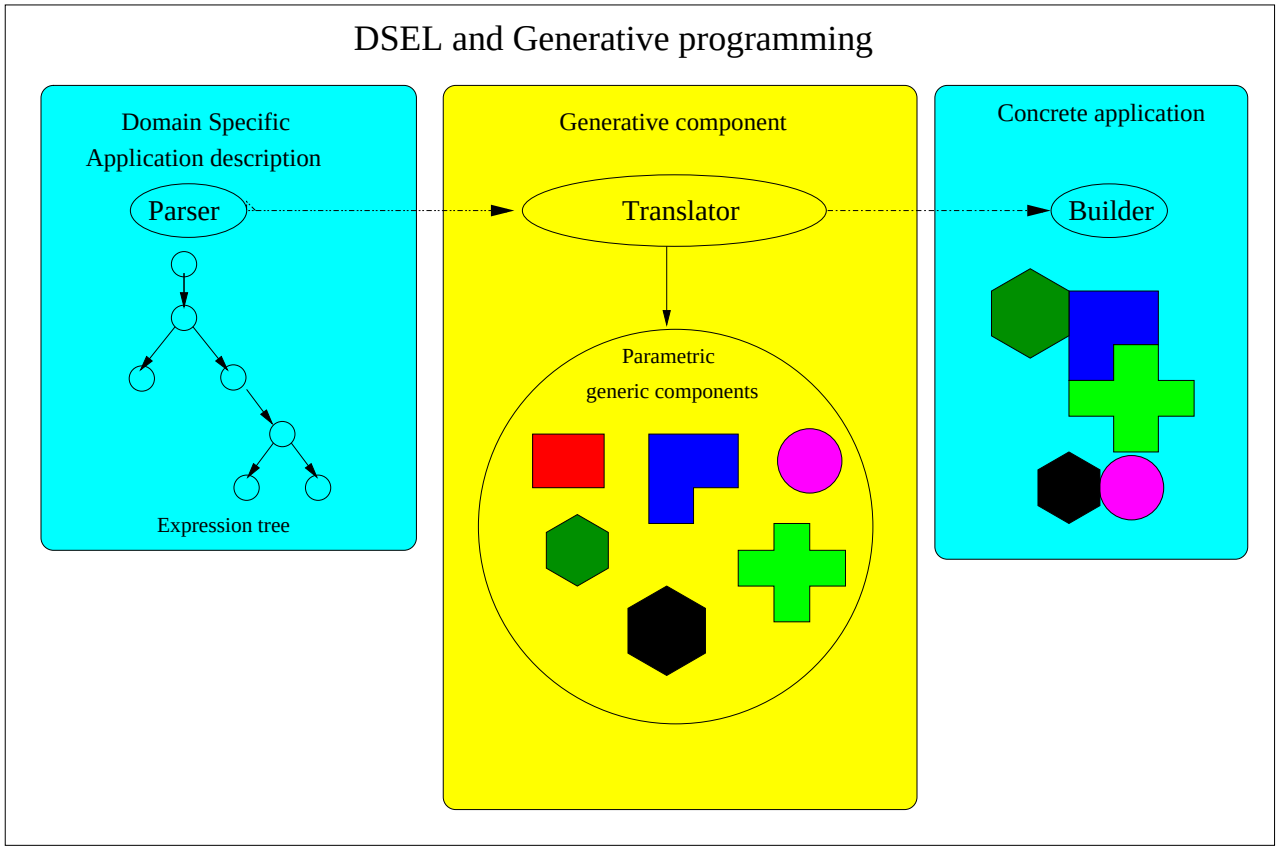


Figure: Generative programming

DSEL design for FV methods : Principles

Language Front ends and Back ends :

- **Front ends** : function space, test, trial functions, discrete variables ;
- **Back ends** : space of dofs, linear combination, matrix and vectors ;
- **DSEL** : linear and bilinear forms, bilinear operator (+,-,*,/) predefined keywords (**integrate**(.,.), **grad**(.), **flux**(.), **div**(.), **jump**(.), **avg**(.), **dot**(.,.))
- **Purpose** :
  - define linear and bilinear forms representing the discrete formulation of the PDE problem,
  - solve the problem evaluating the expressions of the forms.

Standard tools :

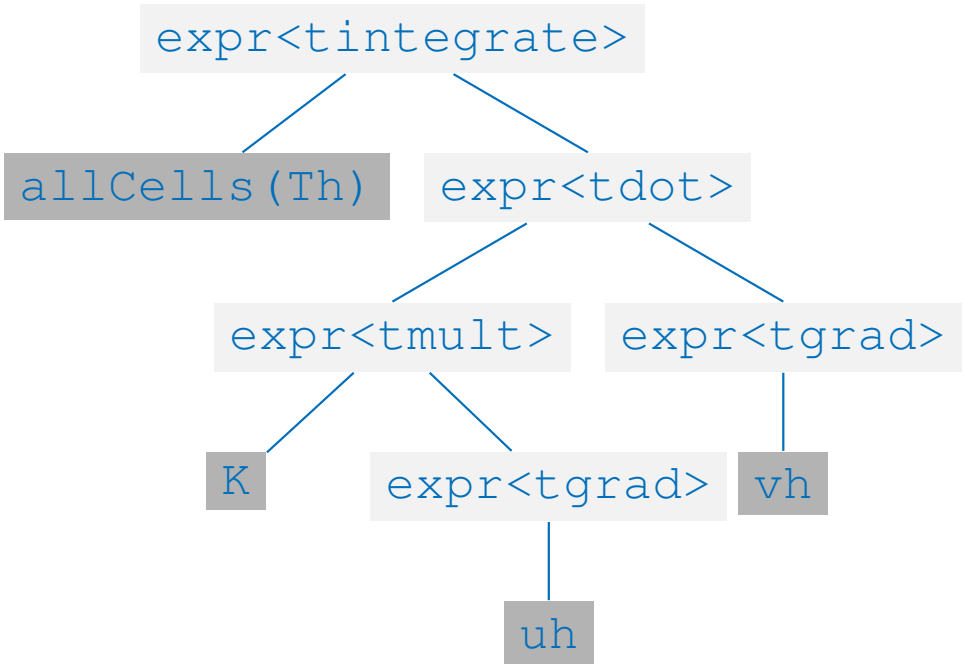
- Boost::Proto library to design the DSEL
- Boost::MPL, Fusion,. . . : MetaProgramming
- standard C++ : Generic Programming
- Arcane : C++ parallel framework providing mesh structures, network, IO services, post traitment tools,. . .
- External C++ libraries (Mesh, linear solvers,. . .)

DSEL design for lowest order methods : Principles

$$a(u,v) \stackrel{\text{def}}{=} \int_\Omega \kappa \nabla u \cdot \nabla v$$

Example of Bilinear Expression :

```
integrate(allCells(Th), dot(K*grad(u),grad(v)) ;
```



Tree expression representation

```
auto Th = mesh(itemGroupExpr) ;
auto Uh = trialSpace(trialExpr);
auto Vh = testSpace(testExpr);
LinearAlgebra::Matrix matrix(Uh,Vh);
std::for_each(itemGroupExpr.begin(),
              itemGroupExpr.end(),
              [& Th,& matrix](Cell&
{
  auto meas = measure(Th,cell);
  auto KGuGv =
   LCAlg::dot(eval(trialExpr,cell),
              eval(testExpr,cell));
  matrix.assemble(meas,KGuGv);
}
```

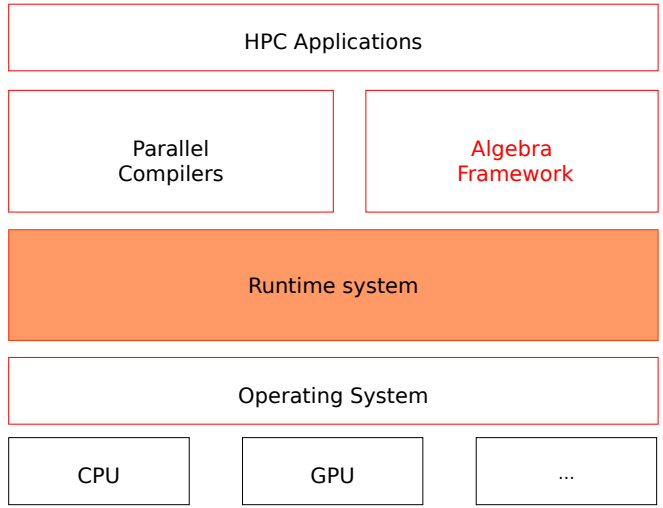
Generic algorithm for linear evaluation

HARTS layer

Architecture

HARTS Hybrid **A**rchitecture **R**un**T**ime **S**ystem is :
An Abstract Object Oriented Runtime System based :

- a **Hardware architecture model** based on **HWLOC**;
- a **task management abstraction** ;
- a **data management abstraction** ;
- a **scheduler abstraction**.



Implementation:

- Abstractions are modelized by C++ concepts
- They are highly inspired from StarPU, Xkaapi, SarSS abstractions
- Enables various implementations with various technologies (TBB, Boost.Thread, Posix thread, StarPU, MPC,. . .)

Applications

Hybrid method

Darcy problem with Hybrid method

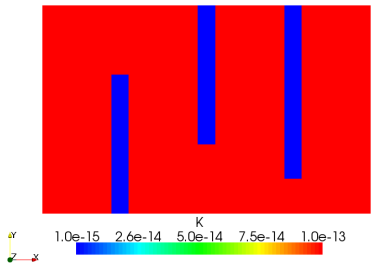


Figure: SHPCO2 : permeability field

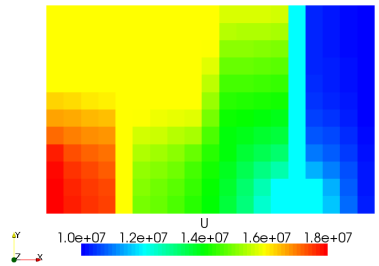


Figure: SHPCO2 : pressure field

C++ PDE definition

```
MeshType Th ;
auto Uh = newHybridSpace(Th) ;
auto u = Uh->trial() ;
auto v = Uh->test() ;
BilinearForm ah_hyb =
   integrate( allCells(Th),
              dot(nu*grad(u),grad(v)) ) ;
```

CCG method

Stokes problem with CCG method

C++ PDE definition

```
MeshType Th(/*\ldots*/) ;
auto Uh = newCCGSpace(Th) ;
auto Ph = newP0Space(Th) ;
auto u = Uh->trialArray(Th::dim) ;
auto v = Uh->testArray(Th::dim) ;
auto p = Ph->trial() ;
auto q = Ph->test() ;
FVDomain::algo::Range<1> _i(Th::dim) ;
BilinearForm ah = integrate( allCells(Th),
    sum(_i)[   dot(grad(u(_i)),grad(v(_i)))] ))
               + integrate( InternalFaces(Th),
    sum(_i)[  -dot(N(Th),avg(grad(u(_i))))*jump(v(_i))
              -jump(u(_i))*dot(N(),avg(grad(v(_i))))
               + eta/H(Th)*jump(u(_i))*jump(v(_i)) ] ) ;
BilinearForm bh = integrate( allCells(Th), -p*div(v) )
                + integrate( allFaces(Th),  avg(p)*dot(N(Th),jump(v))) ;
BilinearForm bth = integrate( allCells(Th), div(u)*q )
                 + integrate( allFaces(Th), -dot(N(Th),jump(u))*avg(q) ) ;
BilinearForm sh = integrate(internalFaces(Th),H(Th)*jump(p)*jump(q));
LinearForm fh = integrate( allCells(Th),sum(_i)[ f(_i)*v(_i) ]) ;
```

Applications : Elasticity problem

Hybrid method

Continuous settings:

$-\nabla\cdot\sigma(\mathbf{u}) = \mathbf{f}$ on Ω

$u = g$ on $\partial\Omega_d$,

$\sigma\cdot\mathrm{n}$ on $\partial\Omega_n$

$\varepsilon(\mathbf{v}) \stackrel{\text{def}}{=} \frac{1}{2}(\nabla\mathbf{v} + \nabla\mathbf{v}^T)$

$\sigma(\mathbf{v}) \stackrel{\text{def}}{=} 2\mu\varepsilon(\mathbf{v}) + \lambda\nabla\cdot\mathbf{v}I_d$

Weak formulation:

$U_h(\mathscr{T}_h)$ a Hybrid space,

$(u,v) \in U_h^d(\mathscr{T}_h) \times U_h^d(\mathscr{T}_h)$

Find $\mathbf{u}$ such that $\forall\mathbf{v}\ a_h(\mathbf{u},\mathbf{v}) = b_h(\mathbf{v})$

$a_h(\mathbf{u},\mathbf{v}) \stackrel{\text{def}}{=} \int_\Omega \sigma(\mathbf{u}) : \varepsilon(\mathbf{v})$

$b_h(\mathbf{v}) \stackrel{\text{def}}{=} \int_\Omega \mathbf{f}\cdot\mathbf{v}$

C++ PDE definition

```
MeshType Th(/*\ldots*/) ;
auto Uh = newHybridSpace(Th) ;
auto u = Uh->trialArray("U",Th::dim) ;
auto v = Uh->testArray("V",Th::dim) ;
//BILINEAR AND LINEAR FORMS
BilinearForm ah  = integrate( allCells(Th),m_2mu*ddot(eps(u),eps(v)))
                 + integrate( allCells(Th),m_lambda*(div(u)*div(v)));
LinearForm bh = integrate( allCells(Th),dot(m_f,v)) ;

// Boundary conditions DIRICHLET + NEUNMANN
ah += on(boundary(Th,"Dirichlet", trace(u) = g ) ;
ah += integrate(boundary(Th,"Neunman"), alpha*dot(SigmaN(u),v)) ;
bh += integrate(boundary(Th,"Neunman"),alpha*dot(t,v) ) ;) ;
```

Applications : Elasticity problem

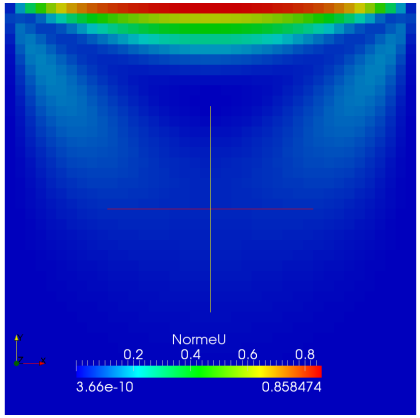


Figure: Elasticity Cavity test

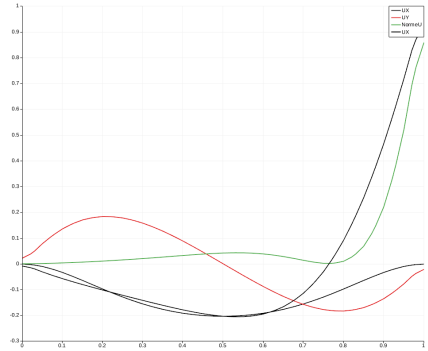


Figure: Elasticity Cavity test

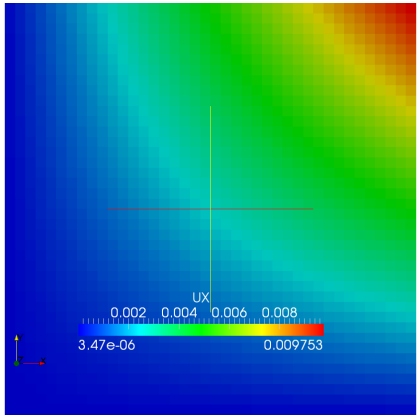


Figure: 2D Synthetic test

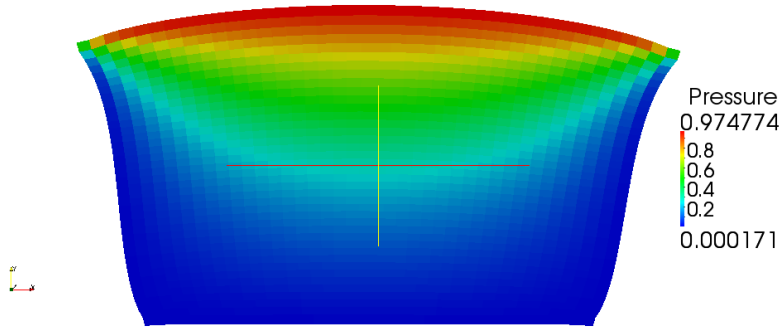


Figure: Darcy + Elasticity test

Extension of the DSEL to multiscale methods

Multiscale pressure solver : continuous settings

Two level mesh :
- $\mathscr{T}_h^f$ and $\mathscr{T}_h^c$

Fine problem on $\mathscr{T}_h^f$:

$$\begin{cases} v = -\nu \nabla u & \text{in } \Omega, \\ \nabla \cdot (-\nu \nabla u) = f & \text{in } \Omega, \\ u = g & \text{on } \partial\Omega, \\ \partial_n u = f & \text{on } \partial\Omega, \end{cases}$$

Basis function definition Φ_{F_c} :

$$\begin{cases} \nabla \cdot (-\nu \nabla \Phi_{F_c}) = w & \text{in } \Omega_{F_c}, \\ w = \dfrac{trace(v)}{\int_{\Omega_{F_c}} v} & \text{on } \Omega_{F_c}^{front} \\ w = -\dfrac{trace(v)}{\int_{\Omega_{F_c}} v} & \text{on } \Omega_{F_c}^{back} \\ \partial_n u = 0 & \text{on } \partial\Omega_{F_c}, \end{cases}$$

Coarse problem definition on $\mathscr{T}_h^c$:
$U^{hms} = \mathbb{P}_d^0(\mathscr{T}_h) + span < \Phi_{F_c} >_{F_c \in \mathscr{F}_{h\Omega c}}$
Find $\mathbf{u} \in U^{hms}$,
$\mathbf{u} = \sum_{K_c} \mathbf{u}_{K_c} \chi_{K_c} + \sum_{F_c} \mathbf{v}_{F_c} \Phi_{F_c}$

Multiscale method : basis function support

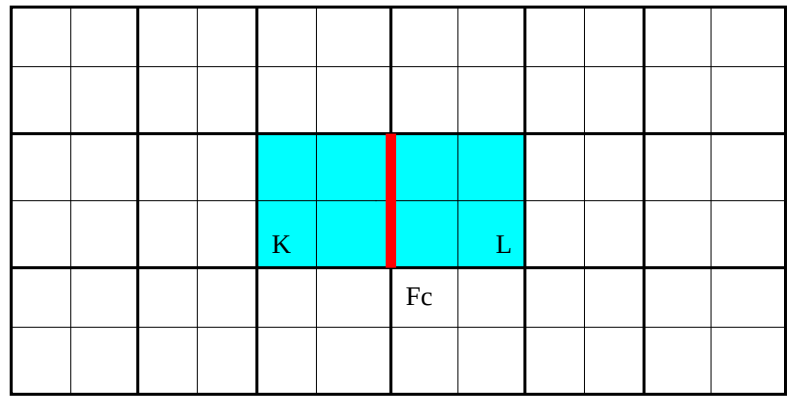


Figure: Basis function

Extension of the DSEL to multiscale methods

Multiscale pressure solver : variational formulation

Basis function problem Φ_{F_c} :
$(u_h, v_h) \in U_h^{hyb} \times U_h^{hyb}$,

$$\begin{cases} a_h^{hyb}(u_h, v_h) & \stackrel{\text{def}}{=} \int_{\Omega_b} \nu \nabla_h u_h \cdot \nabla_h v_h \\ b_h(v_h) & \stackrel{\text{def}}{=} \int_{\Omega_b} w * v_h \end{cases}$$

Coarse problem :
$\mathbf{P_c} = \sum_{K_c} \mathbf{p}_{K_c} \chi_{K_c} + \sum_{F_c} \mathbf{v}_{F_c} \Phi_{F_c}$
$(u_h, v_h) \in U_h^{hms} \times U_h^{hms}$,

$$\begin{cases} a_h^{hms}(u_h, v_h) & \stackrel{\text{def}}{=} \int_{\Omega} \nabla_h \nu u_h \cdot \nabla_h v_h \\ & - \sum_{\sigma \in \Omega_h} \int_{\sigma} (\llbracket u_h \rrbracket (\{\nu \nabla_h u_h\} \cdot \mathrm{n}_\sigma) + (\{\nu \nabla_h u_h\} \cdot \mathrm{n}_\sigma) \llbracket v_h \rrbracket) \\ & + \sum_{\sigma \in \Omega_h} \int_{\sigma} \frac{\eta}{h} \llbracket u_h \rrbracket \llbracket u_h \rrbracket \end{cases}$$

Fine solution :
$\mathbf{v_f} = \mathbf{flux}(\mathbf{P_c}) = \sum_{F_c} \mathbf{v}_{F_c} \mathbf{flux}(\Phi_{F_c})$

Extension of the DSEL to multiscale methods

Multiscale pressure solver : DSEL implementation

C++ PDE definition

```
MultiscaleMeshType Th(/*...*/) ;
//COARSE PROBLEM DEFINITION
auto Uh = HMSSpaceType::create(Th) ;
/*...*/
auto u = Uh->trial() ;
auto v = Uh->test() ;
BilinearForm ah =
    integrate( allCells(Th),dot(grad(u),grad(v)) ) +
    integrate( allFaces(Th),-jump(u)*dot(N(Th),avg(grad(v)))
                            -dot(N(Th),avg(grad(u)))*jump(v)
                            +eta/H(Th)*jump(u)*jump(v));
ah += on(boundaryFaces(Th), u=ud ) ; //! dirichlet condition
LinearComputeContext lctx(solver) ;
fvdsel::eval(ah,lctx) ;
solver.solve() ;

//FINE PROBLEM SOLUTION
FaceRealVariable& fine_velocity = /*...*/ ;
DownScaleEvalContext dctx(fine_velocity) ;
fvdsel::eval(downscale(allCells(Th),flux(u)),dctx) ;
```

Extension of the DSEL to multiscale methods

Interest of multiscale methods

Many computations are independant :

- Basis computations;
- assembling computations;
- downscaling computations.

New hybrid architectures provide different ways of optimization :

- GP-GPU;
- multi-core parallelism;
- multi-node parallelism.

The DSEL separates the numerical level from the back end level. Optimisations should be easily handled at a low level.

Applications : the SPE10 benchmarck

Study case from the 10^{th} comparatif SPE benchmarck

SPE10 model:

- Darcy problem (Pressure)
- Transport problem (Saturation)
- Mesh : 65x220x85
- Boundary conditions :
  - $P_{xmin} = 1000 bars$, $S_{xmin} = 1.$
  - $P_{xmax} = 500 bars$, $S_{xmx} = 0.$

Methods implemented:

- Hybrid method on fine mesh
- Multiscale method on 85th layer
  - Fine mesh : 65x220x1
  - Coarse mesh : 10x10x1

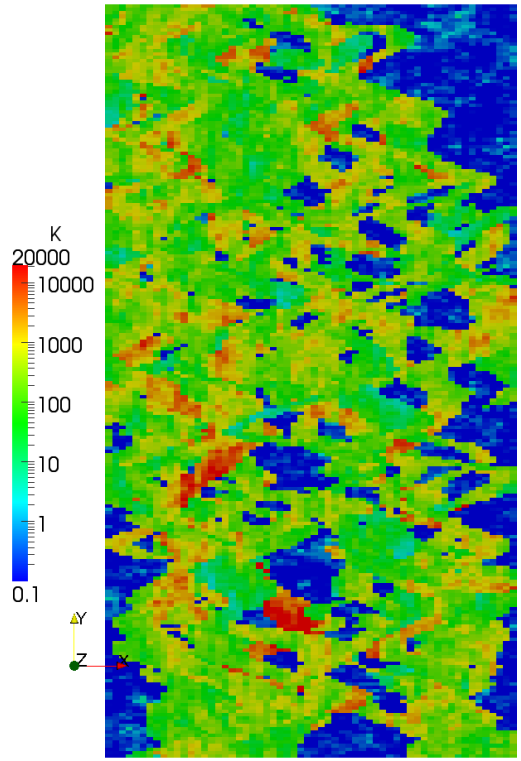


Figure: Fine permeability

Applications : the SPE10 benchmarck

The Hybrid method results

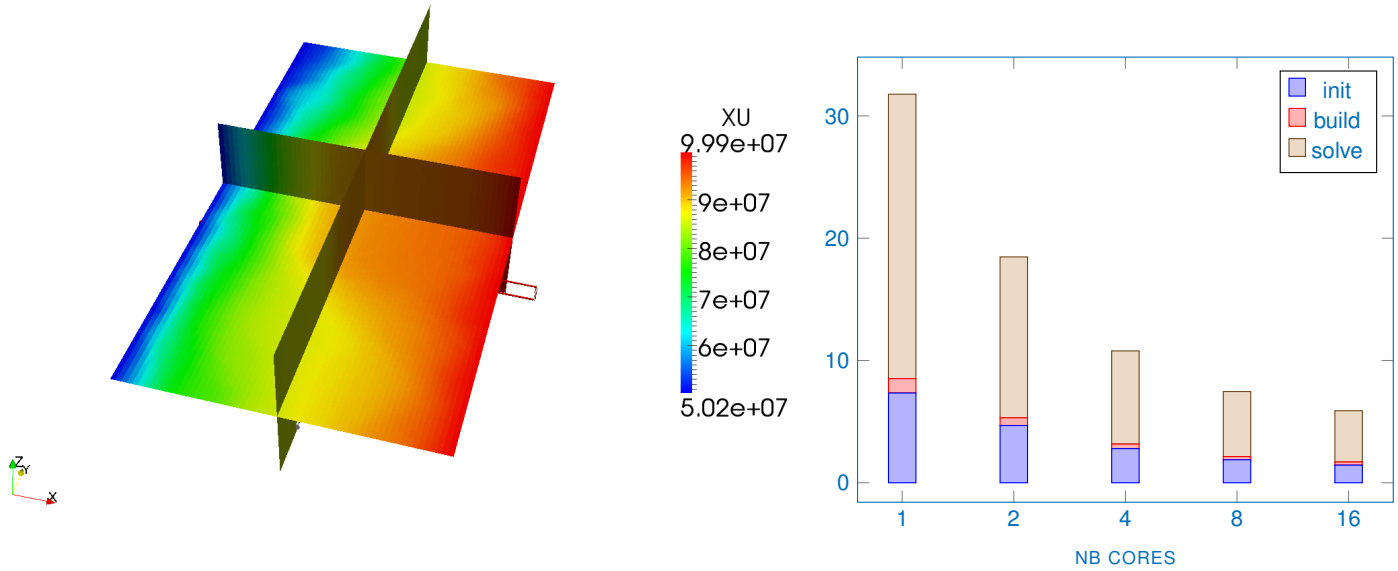


Figure: SPE10 3D : Performance results of the Hybrid method

Applications : the SPE10 benchmarck

Basis function computation : Multi-core and GPU performance results

ForkJoin : TBB, Boost.Thead, PThread

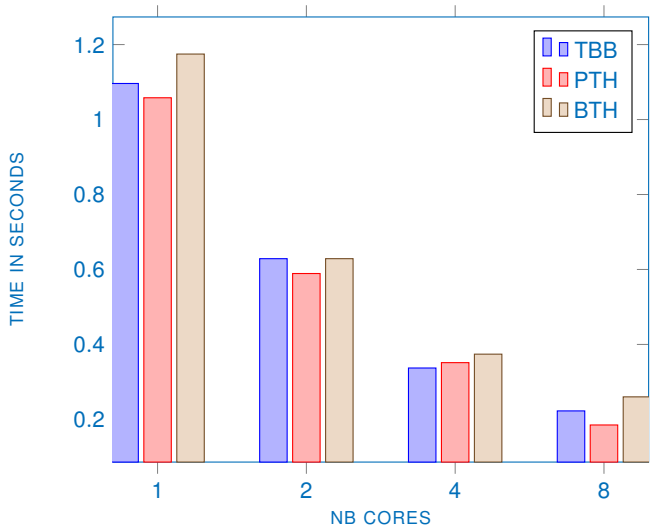


Figure: Multi-core configuration

Pipeline : GPUAlgebraFramework

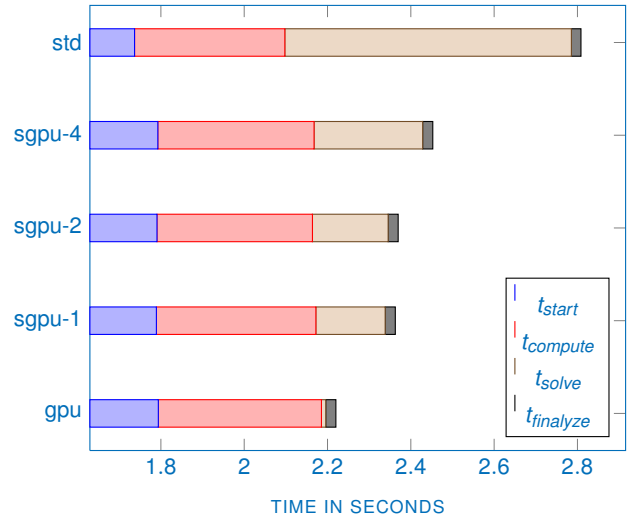


Figure: GPU configuration

Conclusions

Conclusion

- A new DSEL for lowest order methods ;
- Recover various methods (L-scheme, ccG, Hybrid method) ;
- Implementation of non trivial academic test cases ;
- Performance issues (language overhead, benchmarcks with hand written codes).
- HAMM ANR projects : Multi-scale models and hybrid architecture
  - extension of the DSEL for multi scale methods ;
  - use GPU back ends for linear solvers ;
  - take into account hardware specifications :
    - multi nodes ;
    - multi cores ;
    - general purpose accelerators ;
    - link with HARTS an abstract object oriented RunTime System for Hybrid Architecture ;

Conclusions

Benefits of Boost.Proto framework :

- Productivity for the developper :
  - DSEL to design DSEL ;
  - a lot of of useful generic tools ;
  - enable to design easily complex DSEL ;
  - DSEL can be easily extended ;
  - DSEL Factory to easily extend Proto standard tools.
- Productivity for the end user :
  - Language to design complexe numerical methods ;
  - Language that seperates concerns :
    - mathematics, numerics ;
    - computer science, high performance computing;

Perspectives

- Extend the DSL for :
  - non linearity with Frechets derivatives;
  - advection and transport problems;
- add DG methods to handle higher order methods
- New business applications :
  - Linear elasticity, poro-mecanic, Biot equation ;
  - dual medium model ;
  - advection and combustion.
- HARTS extensions :
  - extension with StarSS, StarPU and XKaapi implementation;
  - handle Xeon Phi and MIC architectures;
  - better handle fine grain size tasks parallelism;

Links

Some links :

- Boost : www.boost.org
- Boost.Proto : www.boost.org/libs/proto
- HAMM projects : www.hamm-project.org
- Arcane framework : POOSC '09 Proceedings of the 8th workshop on Parallel/High-Performance Object-Oriented Scientific Computing
- Dune framework : www.dune-project.org
- Fenics : fenicsproject.org
- Feel++ : www.feelpp.org