Domain Specific Embedded Languages
A Galerkin framework for finite element and spectral element

C. Prud’homme

Université de Strasbourg
Université de Grenoble

Strasbourg, Jan 25, 2012
Collaborators

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

1. Motivations
2. Mathematical Framework
3. Seamless Interpolation Tool
4. FSI solver
5. Conclusions, Some on-going and future work
6. Benchmarking
Motivations

Generative Programming and DS(E)L

Complexity of Scientific Computing Software

- Best expressivity using high level language
- Best performance using low level language

Complexity Types

- Algebraic
- Numerical
- Models
- Computer science

- Numerical and model complexity are better treated by a high level language
- Algebraic and computer science complexity perform often better with low level languages
Generative Programming and DS(E)L

- **Best expressivity using high level language**
- **Best performance using low level language**
- **Complexity of Scientific Computing Software**
  - Physical Models
  - Algebraic Methods
  - Numerical Methods
  - Computer Science

- **Generative paradigm**
  - **distribute/partition complexity**
  - **developer**: The computer science and algebraic complexity
  - **user(s)**: The numerical and model complexity
Generative Programming and DS(E)L

Motivations

Best expressivity using high level language
Best performance using low level language

Physical Models

Express

Numerical Methods

Domain Specific Embedded Language for Galerkin Methods

Computer Science

Generate

Algebraic Methods

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++)
Example of DS(ES)L $-\Delta u = f$

Applied Mathematicians favorite equation: find $u$ such that

$$-\Delta u = f, \quad u = g \text{ on } \partial \Omega$$

which, using a Galerkin method, reads find $u \in X_h$ such that

$$\forall v \in X_h \int _\Omega \nabla u \cdot \nabla v = \int _\Omega fv$$

which leads to solving $Au = b$

where

$$A = \left( \int _\Omega \nabla \phi _i \cdot \nabla \phi _j \right) _{i,j}, \quad f = \left( \int _\Omega f \phi _i \right) _i$$

```cpp
auto mesh = Mesh<Simplex<2>>::New();
loadGMSHMesh( _mesh=mesh, _filename="mesh.msh" );
// $P_3$ finite element space on triangular elements
auto Xh = FunctionSpace<Mesh<Simplex<2>>, Lagrange<3>>::New(mesh);
auto u = Xh->element();
auto v = Xh->element();
// f a function e.g. $f = 2\pi^2 \sin(\pi x) \cos(\pi y)$
auto f = 2*\pi^2*\sin(\pi Px())*\cos(\pi Py());
form1(_test=Xh, _vector=b) =
    // $\int _\Omega \nabla \phi _i \cdot \nabla \phi _j$
    integrate( elements(mesh),
        gradt(u) * trans(grad(v)) );
// \int _\Omega \nabla u \cdot \nabla v
form1(_test=Xh, _trial=Xh, _matrix=A) =
    // $\int _\Omega \nabla \phi _i \cdot \nabla \phi _j$
    integrate( elements(mesh),
        gradt(u) * trans(grad(v)) );
auto g = sin(\pi Px())*cos(\pi Py());
form1(_test=Xh, _trial=Xh, _matrix=A) +=
    on( boundaryfaces(mesh), u, F, g );
// solve M u = b
backend->solve( _matrix=A, _solution=u, _rhs=b );
std::cout << "\n";
```
Languages for PDEs

- Generate automatically operators, matrices, vectors with variational formulations (FEM, SEM) for an easy integration with linear algebra software (PetSc, Trilinos, ...), it is called “Automatic Computational Mathematical Modeling” (ACMM)
- High level language (DS(E)L) close to mathematical formulation
- See UFL, FFC (FeniCs project), Sundance, Freefem++/3D, FEEL++, and possibly others
Outline

1. Motivations
2. Mathematical Framework
   - Seamless Interpolation Tool
   - FSI solver
3. Conclusions, Some on-going and future work
4. Benchmarking
FEEL++


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

Copyright (C) 2006-2012 Université de Grenoble
Copyright (C) 2006-2012 CNRS
Copyright (C) 2009-2012 U. Coimbra
Copyright (C) 2005-2009 EPFL
This program is free software; you can redistribute it and/or modify it under the terms of the GNU LGPL-3.
Available in Debian/Ubuntu
FEEL++ Principles

- The mathematical language is considered the language between the applications and the computing resources to break complexity.
- A computational framework that maps closely the mathematical one.
- Delay as much as possible the manipulation of the algebraic representations, work as much as possible with the functional framework.
- Use state of the art C++ techniques and develop new ones.
- Use Boost as much as possible.
- Use C++11 as much as possible:
  - reduce considerably code complexity
  - introduce very powerful tools
  - allows to write C++ code (strongly typed) that resembles scripts (weakly typed) using the keyword `auto` (type inference).
Architecture

Order Reduction Methods
Reduced Basis Methods

Mono/Multiphysics // Mono/Multiscale solvers

Language for variational formulations, projection and integration embedded in C++

Mesh
Localisation tools
Parallel

Function Spaces
associated elements

Forms
operators

Geometric
transformations

Finite/Spectral/hp Element

Integration methods
numerical / exact

Polynomials, polynomial sets

slepc
petsc
trilinos
gmm

ublas
suitesparse
superlu
arpack++

Linear Algebra

Geometry
d-simplices
simplex products
Basic geometry ops

Programming environments
Runtime environments
MPI Boost,MPI
PETSC / SLEPC
TRILINOS

Core

Independend

Numerical types
standard types
dd_real
qd_real
std::complex<>
autodiff<>
interval<>

Mathematical Kernel
Mesh

- Convexes and associated geometric transformation \( (\mathbb{P}_N, N = 1, 2, 3, 4, 5...) \)
- Support for high order ALE maps [Pena et al., 2010]
- Geometric entities are stored using Boost.MultiIndex
- Element-wise partitioning using Scotch/Metis, sorting over process id key

Example

```cpp
elements(mesh [, processid]);
markedfaces(mesh, marker [, processid]);
```
Function Spaces

- Product of $N$-function spaces (a mix of scalar, vectorial, matricial and different basis types)
- Get each function space and associated “component” spaces
- Associated elements/functions of $N$ products and associated components, can use different backend (gmm, petsc/slepc, trilinos)

Example

```cpp
typedef FunctionSpace<Mesh, bases<Lagrange<2, Vectorial>, Lagrange<1, Scalar>> > space_t;

space_t Xh(mesh);
auto Uh = Xh.functionSpace<0>();
auto x = Xh.element();
auto p = x.element<1>(); // view
```
Natural convection

Coupling heat transfer and fluid flow in a closed square domain

Example

```cpp
// velocity space
typedef Lagrange<Order_s, Vectorial> basis_u_type;

// pressure space
typedef Lagrange<Order_p, Scalar> basis_p_type;

// temperature space
typedef Lagrange<Order_t, Scalar> basis_t_type;

// multipliers for pressure space
typedef Lagrange<0, Scalar> basis_l_type;

typedef bases< basis_u_type, basis_p_type, basis_t_type, basis_l_type> basis_type;

typedef FunctionSpace<mesh_type, basis_type> space_type;
```
Operators and Forms

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

Example

```c++
X1 Xh; X2 Vh;
auto u = Xh.element(); auto v = Vh.element();
// operator \( T: X_1 \rightarrow X_2 \)
auto T = LO( Xh, Vh [, backend] );
T = integrate(elements(mesh), id(u) * idt(v) );
// linear functional \( f: X_2 \rightarrow \mathbb{R} \)
auto f = LF( Vh [, backend] );
T.apply( u, f ); f.apply( v );
```
A Language for variational formulations
Enablers and Features

- Meta/Functional - programming (Boost.MPL...): high order functions, recursion, ...
- Crossing Compile-time to Run-time (Boost.fusion...)
- Lazy evaluations (multiple evaluation engines) use Expr<...> (expression) and Context<...> (evaluation) (e.g. Boost.Proto)

Features: Use the C++ compiler/language optimizations

- Optimize away redundant calculations (C++)
- Optimize away expressions known at compile time(C++)

Example

```cpp
// a : X1 × X2 → ℝ   a = ∫Ω ∇u · ∇v
form (_test=X1,_trial=X2,_matrix=M)=
    integrate( elements(mesh), gradt(u)*trans(grad(v)));
```
Outline

1. Motivations
2. Mathematical Framework
3. Seamless Interpolation Tool
4. FSI solver
5. Conclusions, Some on-going and future work
6. Benchmarking
Context

Motivations

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

Some Applications

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

Global problem

\[
\begin{cases}
-\Delta u &= f \text{ in } \Omega \\
u &= 0 \text{ in } \partial\Omega
\end{cases}
\]

Schwarz non Overlap

- Problem \(i\)
  \[
  \begin{cases}
  -\Delta u_i &= f \quad \text{in } \Omega_i \\
u_i &= 0 \quad \text{in } \partial\Omega_i \setminus \Gamma \\
u_i &= \pm \lambda \quad \text{in } \Gamma
  \end{cases}
  \]

- While not convergence
  
  solveProblem\(i\)_1
  solveProblem\(i\)_2
  \[
  \lambda = \lambda - \frac{1}{2} (u_1 - u_2)
  \]
// interpolation operator
auto opI = IO( Xh2, Xh1 );
while (( err1+err2 ) > err_tol ) {
    u1old=u1; u2old=u2;
    localProblem( u1, - idv ( lambda ) );
    localProblem( u2,  idv ( lambda ) );
    err1=computeConv( u1, u1old );
    err2=computeConv( u2, u2old );

    u2bis = opI( u2 );
    lambda = 0.5*(u1-u2bis);
}

// template interpolation operator

template< typename RhsExprType>
localProblem( element_type& u , RhsExprType RhsExp )
{
    auto Xh=u.functionSpace();
    auto mesh=Xh->mesh();
    auto v = Xh->element("v");
    auto f=cst(1.);

    auto A = backend->newMatrix( Xh, Xh );
    form2( Xh, Xh, A, _init=true ) =
        integrate( elements(mesh),
            gradt( u )*trans( grad( v ) ) );

    auto B = backend->newVector( Xh );
    form1( Xh, B, _init=true ) =
        integrate( elements(mesh), f*id( v ) );
    form1( Xh, B ) +=
        integrate( markedfaces(mesh,"Interface"),
            RhsExp*id( v ) );
    form2( Xh, Xh, A ) +=
        on( markedfaces(mesh,"Boundary"),
            u1, B, cst(0.) );

    backend->solve( _matrix=A, _solution=u, _rhs=B );
}
Seamless Interpolation Tool

Trace of Trace and Lift of lift

(l) volume and (m) trace mesh
the function $g$
and trace of $g$

(n) wirebasket (o) warp with re-
and trace trace spect the func-
tion of $g$

```cpp
auto Xh = space_type::New(mesh);
    // trace function space associated to trace(mesh)
auto TXh = trace_space_type::New(
    mesh->trace(markedfaces(mesh,marker)));
    // trace function space associated to trace(trace(mesh))
auto TTXh = trace_trace_space_type::New(
    TXh->mesh()->trace(boundaryfaces(TXh->mesh())));
    // Let $g$ be an function given on 3D mesh
auto g = sin(pi*(2*Px()+Py()+1./4))*cos(pi*(Py()-1./4));
    // trace of $g$ on the 2D trace_mesh
auto trace_g = vf::project(TXh,
    elements(TXh->mesh()), g);
    // trace of $g$ on the 1D trace_trace_mesh
auto trace_trace_g = vf::project(TTXh,
    elements(TTXh->mesh()), g);
    // extension of trace_trace_g by zero on 2D trace_mesh
auto zero_extension = vf::project(TXh,
    boundaryfaces(TXh->mesh()), idv(trace_trace_g));
    // extension of trace_trace_g by the mean of
// trace_trace_g on trace_mesh
auto const_extension = vf::project(TXh,
    boundaryfaces(TTXh->mesh()), idv(trace_trace_g)-mean);
    const_extension += vf::project(TXh,
    elements(TXh->mesh()), cst(mean));
    // harmonic extension of const_extension on 3D mesh
auto op_lift = operatorLift(Xh);
auto glift = op_lift->lift(_range=markedfaces(mesh,marker),
    _expr=idv(const_extension));
```

C. Prud'homme et al. (UDS/UJF)
Fat Boundary Method

Flow of particles

Strategy

Navier-Stokes Equations

\[ \alpha \mathbf{v} - \Delta \mathbf{v} = f_1 \quad \text{in} \quad \Omega_f \]
\[ \mathbf{v} = g \quad \text{on} \quad \partial \Omega_f \]

\[ -\Delta p = f_2 \quad \text{in} \quad \Omega_f \]
\[ \frac{\partial p}{\partial n} = 0 \quad \text{on} \quad \partial \Omega_f \]

Requires the solution of two types of problem

CIRA/CHPID

C. Prud’homme et al. (UDS/UJF)
Fat Boundary Method

How to use rapid solvers without accuracy loss in the neighborhood of particles?

Original Problem:

\[-\Delta u = f \text{ in } \Omega \setminus \overline{B} \]
\[u = 0 \text{ on } \Gamma \cup \gamma\]

FBM: [Bertoluzza et al., 2010]

- one Global:
  \[-\Delta \hat{u} = \bar{f} + \partial_n \nu \delta_\gamma\]

- one Local:
  \[-\Delta v = f \quad v|_{\gamma'} = \hat{u}|_{\gamma'}\]
Fat Boundary Method

Fix point alg. for FBM

// global problem
form2( Xh, Xh, Mg ) =
   integrate( elements(globalmesh), gradt(u)*trans(grad(w)));
form2( Vh, Vh, Ml ) =
   integrate( elements(localmesh), gradt(ul)*trans(grad(wl)));

// fix point iteration with relaxation, \( u^0, \ldots, u^n \)
do {
   // iteration \( i \)
   \( u_{\text{relax}}^i = \theta u_{\text{relax}}^{i-1} + (1 - \theta)u^{i-1} \)
   form1( Vh, Fl, _init=true ) =
      integrate( elements(localmesh), f*id(wl) );
   form1( Vh, Fl ) +=
      on( markedfaces(localmesh, \( \gamma' \)), ul, u_{\text{relax}}^i );

   // solve for local problem \( ul \)
   // \( \chi_p \) : characteristic function of the particle
   form1( Xh, Fg, _init=true ) =
      integrate( elements(globalmesh), f*(1-\( \chi_p \))*id(w) );
   form1( Xh, Fg ) +=
      // interpolation tool: integrate on local mesh
      // against global test functions
      integrate( markedfaces(localmesh, \( \gamma \)),
                  (gradv(ul)*N())*id(w) );

   // solve for global problem on \( u^i \)
}
   // until convergence
Outline

1. Motivations
2. Mathematical Framework
3. Seamless Interpolation Tool
4. FSI solver
5. Conclusions, Some on-going and future work
6. Benchmarking
Implementation of an FSI solver using Feel++ — Navier-Stokes Solver

// automatic type (need to specify -std=c++0x)
// test strain tensor
auto def = 0.5*(grad(v) + trans(grad(v)));
// trial strain tensor
auto deft = 0.5*(gradt(u) + trans(gradt(u)));
// oseen
form2( _test=Xh, _trial=Xh, _matrix=M) =
  // automatic quadature
integrate( elements(Xh->mesh()),
  alpha*trans(idt(u))*id(v)
  + 2.0*nu*trace(trans(deft)*def)
  + trans(gradt(u)*idv(beta))*id(v)
  - div(v)*idt(p) + divt(u)*id(q) );

\[
\frac{\partial \mathbf{u}}{\partial t} \bigg|_Y + [(\mathbf{u} - \mathbf{w}) \cdot \nabla] \mathbf{u} + \nabla p - 2\nu \mathbf{D}(\mathbf{u}) = \mathbf{f}, \text{ in } \Omega_t
\]

\[
\nabla \cdot \mathbf{u} = 0, \text{ in } \Omega_t
\]
Implementation of an FSI solver using Feel++ – Structure Solver

\[ \begin{align*}
\rho_s \frac{\partial^2 d}{\partial t^2} + \text{div} \hat{T} &= f \\
\hat{T} \cdot \hat{n}_s &= \det \left( \frac{\partial A_t}{\partial \hat{x}} \right) \sigma_f \frac{\partial A_t}{\partial \hat{x}} \hat{n}_s \\
d &= 0 \text{ or } \hat{T} \cdot \hat{n}_s = 0
\end{align*} \]

in \( \hat{\Omega}_s \)
on \( \hat{\Sigma} \)
on \( \partial \hat{\Omega}_s / \hat{\Sigma} \)
Implementation of an FSI solver using Feel++

- Solve structure model
- Compute and apply ALE map on fluid domain
- Solve fluid model
- Implicit and semi implicit FSI schemes using High order methods in space, time and geometry

```cpp
// solve structure
elasticity.solve();
auto Ustruct = elasticity.displacement();
// generate the ale map A_t using structure displacement
ale->generateMap(Ustruct);
// get mesh displacement and generate a new mesh using A_t applied on the reference domain
meshmove.apply(mesh,
    ale->displacement());
// compute mesh velocity
// current position after displacement
auto x = vf::project(Xh, elements(mesh), P());
// w = (x^{n+1} - x^n)/dt
// with x^{n+1} = A_t(Y, t_{n+1})
// and x^n = A_t(Y, t_n)
w = (x - bdf_ale->last()).scale(1/dt);
// update and solve fluid
navierstokes.update(w);
```

(a) Fluid pressure (displacement magnified 10 times)
(b) Structure displacement (displacement magnified 10 times)
Outline

1. Motivations
2. Mathematical Framework
3. Seamless Interpolation Tool
4. FSI solver
5. Conclusions, Some on-going and future work
6. Benchmarking
Conclusion

- Generative programming for PDE works thanks to C++ (GCC and C++11) and compilation time improves (not there yet but better)
- Feedback: fast prototyping (at least for methodology), domain specific language, devil lurks in the details (interpolation, ...), used by physicist in micro-fluidic
- Wide range of applications

Some on-going/future work

- Mathematical framework (EF, FV, MDF...), more abstractions
- Full parallelisation, domain decomposition framework
- Exploit hybrid architectures (CPU-GPGPU)

Dissemination

- http://www.feelpp.org,
  http://forge.imag.fr/projects/feelpp
References I

Analysis of the fully discrete fat boundary method.
Numerische Mathematik.
accepted.

Continuous interior penalty finite element method for the time-dependent navier-stokes equations: space discretization and convergence.

High order methods for the approximation of the incompressible navier-stokes equations in a moving domain.
Accepted.

Construction of a high order fluid-structure interaction solver.
To appear.
Outline

1. Motivations
2. Mathematical Framework
3. Seamless Interpolation Tool
4. FSI solver
5. Conclusions, Some on-going and future work
6. Benchmarking
Benchmark — Equations

\[
\begin{align*}
\mu \Delta u &= 0 \quad \text{D,} \\
\mu \Delta u + \sigma u &= 0 \quad \text{DR,} \\
\mu \Delta u + \beta \cdot \nabla u + \sigma u &= 0 \quad \text{DAR}
\end{align*}
\]

with \( \mu, \sigma \) and \( \beta \) constant or space-depandant:

<table>
<thead>
<tr>
<th>( \mu(x, y, z) = )</th>
<th>nD</th>
<th>2D</th>
<th>3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1 )</td>
<td>( x^3 + y^2 )</td>
<td>( x^3 + y^2 z )</td>
<td></td>
</tr>
<tr>
<td>( (1)_{i=1,n} )</td>
<td>( (x^3 + y^2, x^3 + y^2) )</td>
<td>( (x^3 + y^2 z, x^3 + y^2, x^3) )</td>
<td></td>
</tr>
<tr>
<td>( 1 )</td>
<td>( x^3 + y^2 )</td>
<td>( x^3 + y^2 z )</td>
<td></td>
</tr>
</tbody>
</table>
Performances I

(c) 2D

(d) 3D
Performances II

(e) 2D

(f) 3D
Performances III

Relative CPU Time for $R$ with respect to polynomial order $N$ per matrix entry and per quadrature point.
Testing '0' Terms

Influence of terms that should be 0

// 2D -> dz(.)-> 0
vf_D = integrate( elements(*mesh),
IM, dxt(u)*dx(v)+dyt(u)*dy(v)+dzt(u)*dz(v));
vh_D = integrate( elements(*mesh),
IM, dxt(u)*dx(v)+dyt(u)*dy(v));

<table>
<thead>
<tr>
<th>Order</th>
<th>Elt</th>
<th>Dof</th>
<th>Without '0' Terms</th>
<th>With '0' Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/4</td>
<td>20786</td>
<td>10592</td>
<td>0.11</td>
<td>0.11</td>
</tr>
<tr>
<td>2/9</td>
<td>20786</td>
<td>41969</td>
<td>0.48</td>
<td>0.5</td>
</tr>
<tr>
<td>1/4</td>
<td>20786</td>
<td>41618</td>
<td>0.47</td>
<td>0.45</td>
</tr>
<tr>
<td>2/9</td>
<td>20786</td>
<td>165673</td>
<td>1.92</td>
<td>1.94</td>
</tr>
</tbody>
</table>