The Unified Form Language – UFL

A domain specific language for finite element methods

```
J = (u-d)**2*dx + alpha*v**2*dx
a = dot(grad(u), grad(w))*dx
L = J + a
Lw = derivative(J, w)
Lwu = derivative(Lw, u)
```

Martin Sandve Alnæs

Center for Biomedical Computing,
Simula Research Laboratory,
Oslo, Norway

June 5th, 2012
FEniCS'12

Overview of talk

- Equation complexity (what kind of scalability)
- Expression architecture
- Automatic simplifications
- Profiling (techniques and optimizations applied recently)
- Scalable compilation (of complex tensor algebra expressions)
- Outlook

Topics

Equation complexity

Expression architecture

Automatic simplifications

Profiling

Scalable compilation

Outlook

UFL is a DSL, a symbolic framework, and a compiler frontend for FEniCS (and other libs)

Example UFL expressions:

```
a = inner(dot(grad(u), M), grad(v))
b = M[i,j] * u[k].dx(i) * v[k].dx(j)
```

- By expression complexity I mean roughly the number of values and operators in an expression, denoted n.
- Goal: Want both time and memory usage in UFL and overall form compilation process to be O(n), i.e. linear in the expression complexity.

Example: Hyperelasticity equations(1/2), taken from DOLFIN demo directory

```
cell = tetrahedron
V = VectorElement("Lagrange", cell, 1)

du = TrialFunction(V)     # Incremental displacement
v  = TestFunction(V)      # Test function

u = Coefficient(V)        # Displacement from previous iteration
B = Coefficient(V)        # Body force per unit volume
T = Coefficient(V)        # Traction force on the boundary
# Elasticity parameters
mu    = Constant(cell)
lmbda = Constant(cell)
```

Example: Hyperelasticity equations(2/2), taken from DOLFIN demo directory

```
# Kinematics
I = Identity(cell.d)                # Identity tensor
F = I + grad(u)                     # Deformation gradient
C = F.T*F                           # Right Cauchy-Green tensor
# Invariants of deformation tensors
Ic = tr(C); J = det(F)
# Stored strain energy density (compressible neo-Hookean model)
psi = (mu/2)*(Ic - 3) - mu*ln(J) + (lmbda/2)*(ln(J))**2
# Total potential energy
Pi = psi*dx - inner(B, u)*dx - inner(T, u)*ds

# First variation of Pi (directional derivative
# about u in the direction of v)
F = derivative(Pi, u, v)
J = derivative(F, u, du)
```

Topics

Equation complexity

Expression architecture

Automatic simplifications

Profiling

Scalable compilation

Outlook

Expression architecture overview

- Class hierarchy basics, typical symbolic expression tree.
- Immutable objects are important!
- Arbitrary nesting of tensor algebra and index notation.
- Running id count of indices and functions.
- Form signatures to avoid regenerating code.
- Fast O(1) hash and eq operators for use as keys in dict and set.

Topics

Equation complexity

Expression architecture

Automatic simplifications

Profiling

Scalable compilation

Outlook

Automatic simplifications overview

- Basic simplifications at expression node construction time reduce expression growth during algorithms: any scalar operator(scalar literals) -> scalar literal 1*a -> a 0*b -> 0 a + 0 -> a as_tensor(A[i,j], (i,j)) -> A
- Simplifications central to keep differentiation algorithm output from growing: d/dx (x * g(y)) = 1 * g + x * 0 -> g
- Basic canonical term ordering at expression node construction time a*b -> a*b, b*a -> a*b, a+b -> a+b, b+a -> a+b,
- Not doing simplifications requiring deeper inspection of operands, e.g. 2*a + -2*a -> 0 2*a + 3*a -> 5*a as_tensor(A[i,j], (j,i))[k,l] -> A[l,k]
- Definitely not doing unsafe polynomial rewriting (a + b) + (c) != a + (b + c) in floating point (u-v)**2 -> (u**2 - 2*u*v + v**2) numerically unstable (u-v)**2 - (u**2

Topics

- Equation complexity
- Expression architecture
- Automatic simplifications
- Profiling
- Scalable compilation
- Outlook

Profiling overview

- Time profiling with simple Identify heavy algorithms. Get rid of renumber_indices. See that expand_indices is the main bottleneck.
- Memory usage of python objects. Use slots! Found a bug.
- Memory profiling with heapy. Identify which classes to remove member variables from.
- UFL expr node counting. Identify need for caching literal value objects. Identify which classes to optimize Identify explosion of certain types of expression objects in certain algorithms (expand_indices). Identify that v.dx(i) != v.dx(j), but grad(v) == grad(v). Fix in AD.

Profiling time usage from ipython to identify heavy algorithms

%run -p myscript.py

- See that expand_indices is the main bottleneck in current quadrature loop based form compilers. This algorithm rewrites outer(u,v) -> [u[0]*v[0], u[1]*v[0], ...] etc.

Memory usage of python objects

- sys.getsizeof(obj) gives bytesize of obj
- Use __slots__ feature to drop __dict__, which is 280 b alone when empty!
- Still, a single object is 48 b + 8 per member!
- Rough profiling of heap usage: from guppy import hpy; hp=hpy(); print hp.heap()
- Counting objects of each Expr subtype in Expr con-/destructor
- Introduced e.g. reuse of literal value objects.

For one complex case, reduced memory usage in UFL with 10x!

Topics

Equation complexity

Expression architecture

Automatic simplifications

Profiling

Scalable compilation

Outlook

Scalable compilation (1/3)

- Value numbering crossing boundaries of tensor and indexing operations. Sees right through any number of levels of indirection. Backside is that this makes the expression less regular.
- Reconstructing scalar subexpressions, triggering UFL simplifications at the scalar level and thus constant propagation and dead code elimination at the symbolic level.

Scalable compilation (2/3)

- Identifying dependency counts on resulting flat scalar DAG representation.
- Identifying candidates for subexpressions to store in intermediate variables in generated code. Tunable heuristics, consider dependency count and operation cost.

Scalable compilation (3/3)

- Partitioning DAG by loop level required for each subexpression (inside quadrature or test/trial function loops)
- For each scalar subexpression, generate C++ expression, emit assignment statement if candidate for intermediate storing.

Topics

Equation complexity

Expression architecture

Automatic simplifications

Profiling

Scalable compilation

Outlook

Questions?

- Compilation algorithms available in the experimental UFLACS project: *http://www.launchpad.net/uflacs*
- First proof of concept: *soon* possible to use with DOLFIN through SFC.
- Should probably integrate into FFC. Need some help with that.
- Can use to generate element tensor kernels for other FEM libraries!
- *martinal@simula.no*