Mathematical Aspects of Automating Finite Element Computation

Robert C. Kirby

kirby@uchicago.edu

The University of Chicago

Acknowledgements

- Collaborators:
  - Anders Logg, TTI-C
  - Kevin Long, SLNL
  - Ridgway Scott, UC
  - Dmitry Karpeev and Matt Knepley, ANL
- U.S. Dept. of Energy

Introduction

- Survey efforts at automatic PDE simulation
- Meta-theme 1: code is object of *mathematical* investigation
- Meta-theme 2: overarching structure in FEM
- Technical content:
  - Representing discrete multilinear forms
  - Optimizing the evaluation of variational forms
  - Reasoning about form syntax with sieves

Motivation

- Effort is focused on Method X for Problem Y
- Particular experts find development easy (most people do not)
- Difficult to explore wide range of models/methods
- Implicit assumption: Mathematics tells you what to program, not how to program it

Automating PDE simulation

New Languages	Analysa, FreeFEM
Embedded Languages	Sundance, FFC, Lifev
Library support	Albert , Deal

The situation is more complex than ODE:

- General purpose code for $u_t = f(u)$ available since 1970's
- Steady increases in accuracy, adaptive error control, differential algebraic equations, etc
- But no method works for "all" PDE!

Example: Sundance

- Main developer, Kevin Long, SLNL
- C++ (with Python interface) library for specifying weak forms symbolically.
- Differentiation, preprocessing, interface to solvers
- Solve time $>>$ symbolic processing
- Arbitrary order Lagrange elements (other elements pending) from FIAT.
- Example: Pressure-stabilized FEM for Navier-Stokes implemented in 113 lines (I/O, Problem specification, continuation loop, etc)

Variational statement

Steady, incompressible Navier-Stokes equations:
Find $u \in V^g$, $p \in W$ such that

$$
\begin{aligned}
(\nabla u, \nabla v) + Re(u \cdot \nabla u, v) - (p, \nabla \cdot v) &= 0 \\
(\nabla \cdot u, w) &= 0
\end{aligned}
$$

for all $v \in V^0$, $w \in W$.

Crash course in FEM

- Based on weak formulation of problem.
- Approximation is to find solution on finite-dimensional subspace.
- Existence, uniqueness, stability analyzed similar to PDE
- Error estimate $\leftrightarrow$ approximation theory
- But they're hard to program on a computer . . .

Finite element method

We consider the *equal-order stabilized* method

$$(\nabla u_h, \nabla v_h) + Re(u_h \cdot \nabla u_h, v_h) - (p_h, \nabla \cdot v_h) = 0$$
$$(\nabla \cdot u_h, w_h) + \quad \beta h^2 (\nabla p_h, \nabla w_h) \quad = 0$$

Circumvents the “inf-sup” condition (Babuska, Brezzi, Ladyzhenskaya) and allows piecewise linear basis functions for both velocity and pressure.

Finite element method

We consider the *equal-order stabilized* method

$$(\nabla u_h, \nabla v_h) + Re(u_h \cdot \nabla u_h, v_h) - (p_h, \nabla \cdot v_h) = 0$$

$$(\nabla \cdot u_h, w_h) + \beta h^2 (\nabla p_h, \nabla w_h) = 0$$

Circumvents the “inf-sup” condition (Babuska, Brezzi, Ladyzhenskaya) and allows piecewise linear basis functions for both velocity and pressure.

Sample code

Problem definition (declarations happen above)

```
eqn = Integral(interior, (grad*vx)*(grad*ux)  \
              + (grad*vy)*(grad*uy) - p*(dx*vx+dy*vy) \
              + beta*h*h*(grad*q)*(grad*p) + q*(dx*ux+dy*uy) \
              + reynolds*(vx*(u*grad)*ux) \
              + reynolds*(vy*(u*grad)*uy), quad2)

bc = EssentialBC(left, vx*ux + vy*uy, quad2) \
     + EssentialBC(right, vx*ux + vy*uy, quad2) \
     + EssentialBC(top, vx*(ux-1.0) + vy*uy, quad2) \
     + EssentialBC(bottom, vx*ux + vy*uy, quad2)
```

The `NonlinearProblem` class takes derivatives, builds Jacobians, and talks to Newton's method for you.

Current applications

- Source detection
- Geometric/toplogical design of microfluidics devices
- New student projects at Chicago:
  - Studying convergence and conditioning properties of various FEM for Stokes (Andy Terrel, also using FEniCS/FFC/DOLFIN)
  - Incorporation of surface tension in a Rayleigh-Taylor model (Noah Clemons)
  - Comparison of MHD formulations (Peter Brune)
- Many others...

The rest of the talk

- Focus moves beyond *one* code working for *one* problem.
- What is the inherent structure of the pieces of FEM?
- Topics:
  - Form evaluation $\leftrightarrow$ tensor contractions
  - Discrete structures for optimized form evaluation
  - Reasoning about syntax for variational forms.

Tensor structure of discrete forms

- Example: Laplacian
- General result
- Local matrix (or its action) expressed as sequence of tensor contractions.
- These are optimized by discrete metrics/geometry

Example: Laplacian

Variational form:

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v$$

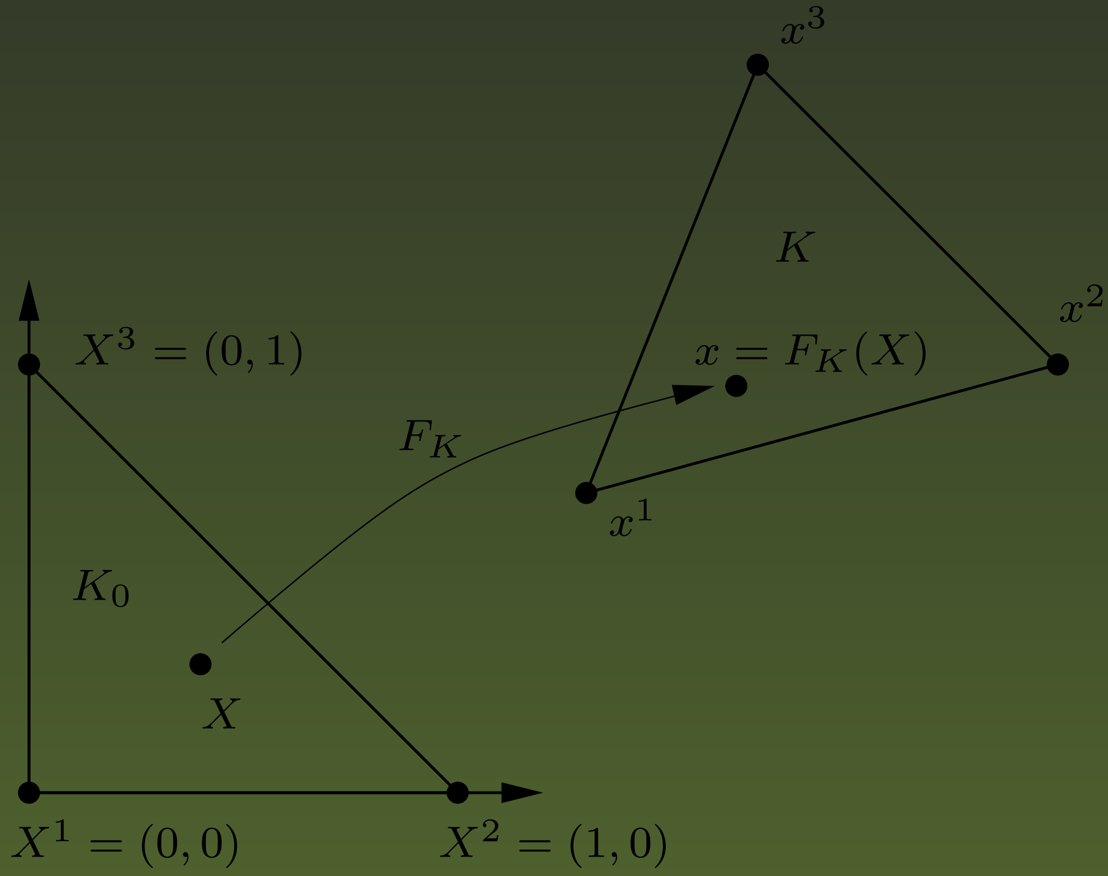
For each $K \in \mathcal{T}_h$, need to build

$$A_i^K = \int_K \nabla \phi_{i_1} \cdot \nabla \phi_{i_2} \, \mathrm{d}x$$
$$= \sum_{d=1}^{D} \int_K D_x^d \phi_{i_1} D_x^d \phi_{i,2} \, \mathrm{d}x$$

This is a sum over *monomial* terms.

Transforming to reference element

Calculation usually happens via a change of variables:



Transforming the Laplacian

$$
\begin{aligned}
A_i^K &= \int_K \nabla \phi_{i_1}^{K,1}(x) \cdot \nabla \phi_{i_2}^{K,2}(x) \, \mathrm{d}x \\
&= \det F_K' \sum_\beta \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta} \int_{K_0} \frac{\partial \Phi_{i_1}^1(X)}{\partial X_{\alpha_1}} \frac{\partial \Phi_{i_2}^2(X)}{\partial X_{\alpha_2}} \, \mathrm{d}X \\
&= \sum_\alpha A_{i\alpha}^0 G_K^\alpha
\end{aligned}
$$

Every A_i^0 is contracted with G_K for each element.

Transforming the Laplacian (2)

Tensors:

$$A^0_{i\alpha} = \int_{K_0} \frac{\partial \Phi^1_{i_1}(X)}{\partial X_{\alpha_1}} \frac{\partial \Phi^2_{i_2}(X)}{\partial X_{\alpha_2}} \, \mathrm{d}X$$

$$G^\alpha_K = \det F'_K \sum_\beta \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta}$$

- Reference element and geometry separated
- One A^0 for the form, one G_K for each element
- Main loop nest for computation of matrix: contract/insert

Quadratics on triangles

3	0	0	-1	1	1	-4	-4	0	4	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
-1	0	0	3	1	1	0	0	4	0	-4	-4
1	0	0	1	3	3	-4	0	0	0	0	-4
1	0	0	1	3	3	-4	0	0	0	0	-4
-4	0	0	0	-4	-4	8	4	0	-4	0	4
-4	0	0	0	0	0	4	8	-4	-8	4	0
0	0	0	4	0	0	0	-4	8	4	-8	-4
4	0	0	0	0	0	-4	-8	4	8	-4	0
0	0	0	-4	0	0	0	4	-8	-4	8	4
0	0	0	-4	-4	-4	4	0	-4	0	4	8

Notice this can be done cheaply, we’ll revisit this idea later.

Symmetry

- Symmetric form $\rightarrow G_K$ symmetric
- Can reduce work for triangles from 4 to 3 (9 to 6 on tets)
- Similar reductions for more complicated symmetric forms.

Canonical form

General class of multilinear forms:

$$A_i^K = \sum_{\gamma \in \mathcal{C}} \int_K \prod_{j=1}^m c_j(i,\gamma) D_x^{\delta_j(i,\gamma)} \phi_{\iota_j(i,\gamma)}^{K,j}[\kappa_j(i,\gamma)] \, \mathrm{d}x$$

Notation:

$c_j(i,\gamma)$	coefficient
$\iota_j(i,\gamma)$	basis function index
$\kappa_j(i,\gamma)$	vector component index
$\delta_j(i,\gamma)$	multiindex for derivative

Vector Weighted Laplacian

$$a(v,u) = \sum_{\gamma_1=1}^{d} \sum_{\gamma_2=1}^{d} \int_\Omega w \frac{\partial v[\gamma_1]}{\partial x_{\gamma_2}} \frac{\partial u[\gamma_1]}{\partial x_{\gamma_2}} \, \mathrm{d}x.$$

$$A_i^K = \sum_{\gamma_1=1}^{d} \sum_{\gamma_2=1}^{d} \sum_{\gamma_3=1}^{|V_3^K|} \int_\Omega \frac{\partial \phi_{i_1}^{K,1}[\gamma_1]}{\partial x_{\gamma_2}} \frac{\partial \phi_{i_2}^{K,2}[\gamma_1]}{\partial x_{\gamma_2}} w_{\gamma_3} \phi_{\gamma_3}^{K,3} \, \mathrm{d}x,$$

In the notation above, we have $r = 2$, $m = 3$, $\iota(i,\gamma) = (i_1, i_2, \gamma_3)$, $\delta(i,\gamma) = (\gamma_2, \gamma_2, \emptyset)$, $\kappa(i,\gamma) = (\gamma_1, \gamma_1, \emptyset)$ and $c_j(i,\gamma) = (1, 1, w_{\gamma_3})$

Representation result

Theorem. Let $F_K : K_0 \to K$ be affine, $\{V_j^K\}_{j=1}^m$,$\{V_j^0\}_{j=1}^m$ discrete function spaces, $\Phi = \phi \circ F_K$. Then

$$A_i^K = \sum_{\alpha \in \mathcal{A}} A^0_{i\alpha} G_K^\alpha \quad \forall i \in \mathcal{I},$$

$$A^0_{i\alpha} = \sum_{\beta \in \mathcal{B}} \int_{K_0} \prod_{j=1}^m D_X^{\delta'_j(i,\alpha,\beta)} \Phi^{K_0,j}_{\iota_j(i,\alpha,\beta)}[\kappa_j(i,\alpha,\beta)] \, \mathrm{d}X,$$

$$G_K^\alpha = \sum_{\beta \in \mathcal{B}'} \det F'_K \prod_{j=1}^m c_j(i,\alpha,\beta) \prod_{j'=1}^m \prod_{k=1}^{|\delta_{j'}(i,\alpha,\beta)|} \frac{\partial X_{\delta'_{j'k}(i,\alpha,\beta)}}{\partial x_{\delta_{j'k}(i,\alpha,\beta)}}$$

Implications

- Separates *reference element* information from *geometric/coefficient* information.
- Reference tensor and code for geometry tensor can be generated once for all (FFC).
- Can be extended to nonlinearities, curved geometry (Logg & K.)
- One element isomorphic to matrix-vector multiplication
- But instead of BLAS...

Optimizing form evaluation

Abstract problem:

- $V \subset \mathbb{R}^m$ with $|V| < \infty$ be given.
- Find a process for computing $\{v^t g : v \in V\}$ for arbitrary $g \in \mathbb{R}^m$ in minimal flops

Points to remember:

- V is very special – not random.
- Finding true minimum is very hard and not necessary.
- This is not something a general-purpose compiler can do.

What kinds of tricks are there?

Look back at the Laplacian.

- Sparsity
- Equality: $u = v$
- Colinearity: $u = \alpha v$, $v \neq 0$.
- Hamming distance
- Linear combinations $u = \alpha v + \beta w$

If $u^t g$ is known, perhaps $v^t g$ can be computed in less than m multiply-add pairs.

Complexity-reducing relations

Definition. Let $\rho : Y \times Y \to [0, m]$ be symmetric. We say that ρ is *complexity-reducing* if for every $y, z \in Y$ with $\rho(y, z) \le k < m$, $y^t g$ may be computed using the result $z^t g$ in no more than k multiply-add pairs.

Examples

$$d(y,z)=\begin{cases}0, & y=z\\ m, & y\neq z\end{cases}$$ Discrete metric

$$c(y,z)=\begin{cases}1, & y=\alpha z\\ m, & y\neq \alpha z\end{cases}$$ Colinearity

$$d^{\pm}(y,z)=\begin{cases}0, & y=\pm z\\ m, & y\neq \pm z\end{cases}$$ Projective

$$H(y,z)=|\{i : y_i \neq z_i\}|$$ Hamming distance

Hamming:

Let $y=\{1,2,3\}$, $z=\{1,2,5\}$.

$z^t g = y^t g + (z-y)^t g$, but $z-y=\{0,0,2\}$.

CRRs and metrics

- Many CRRs are metrics, but not all are.
- Others are metrics on projective space or equivalence classes
- Minimum over CRRs is a CRR
- Minimum over metric CRRs is not necessarily a metric
- Can define a *meet* operation on metrics.

WLOG, we will assume a single CRR ρ that may or may not be a metric in the following.

A sketch of an “optimal” algorithm

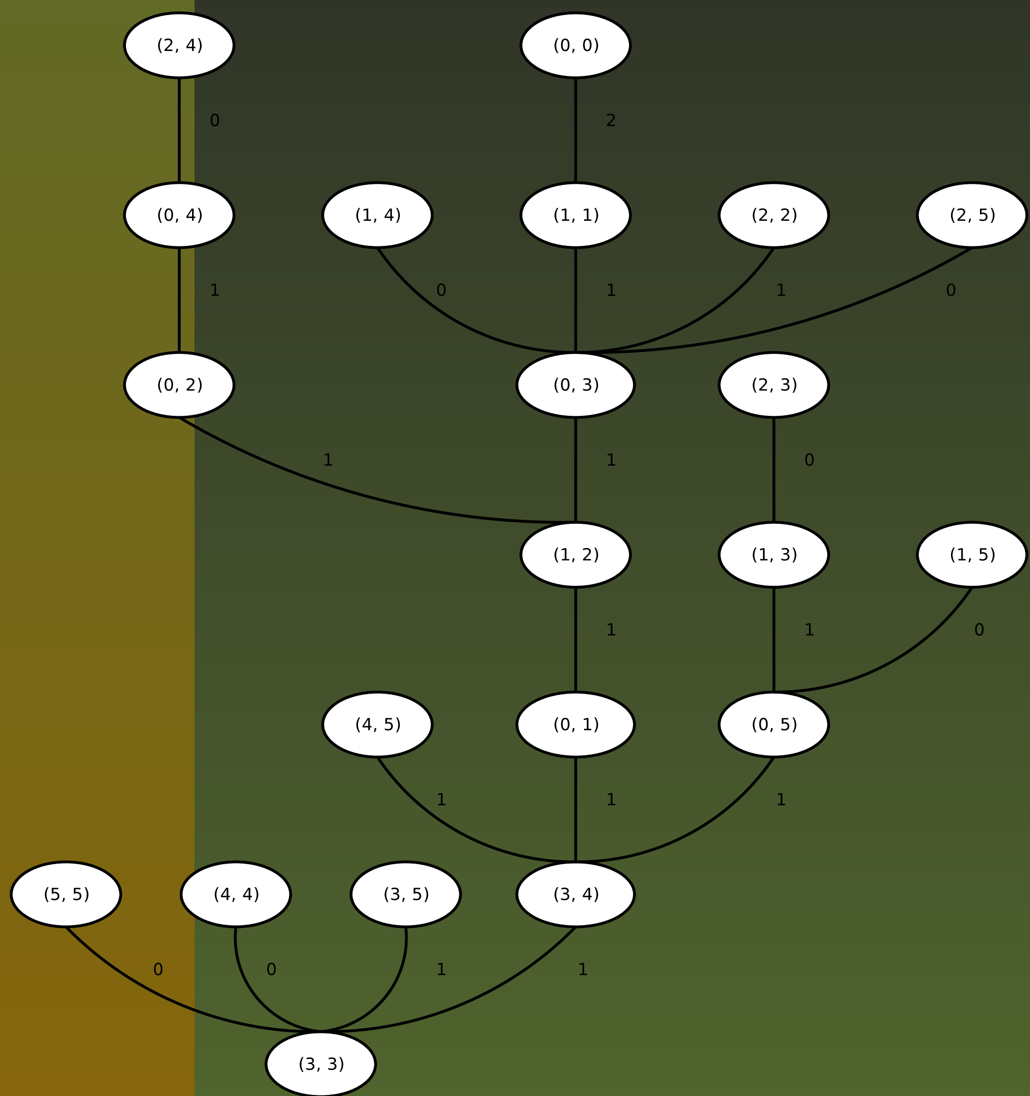
$g \in \mathbb{R}^m$ given
compute $v_1^t g$
$I = \{1\}$
while $I \neq [1, n]$ **do**
    Pick $i \notin I$ to minimize over $\{\rho(v_i, v_j) : j \in I\}$.
    Compute $v_i^t g$
    $I \leftarrow I \cup \{i\}$
end while

Relation to graph structure

(V,ρ) defines a complete weighted graph with elements of V as nodes and $\rho(v_i,v_j)$ the weight of the edge between v_i and v_j.
The above algorithm needs a *minimum spanning tree* of the graph associated with (V,ρ).

Theorem. Let ρ be a complexity-reducing relation on V and $g \in \mathbb{R}^m$ be arbitrary. Then, computing $\{v^t g : v \in V\}$ by traversing of a minimum spanning tree of (V,ρ) gives a minimal-flop computation.

MST example



Total cost: 17 MAPs

Practical details

- V generated by FFC (need to pipe code back)
- Can generate straight-line code (MST/graph is only used for generation, *not* at run-time)
- Computing MST is (worst-case) $n^2 \log n$, $n = |V|$ by Prim or Kruskal.

Experimental results

- Observe flop reduction for a few forms
- Run-time impact for Laplacian

Flop reduction, Laplacian

Total flop count for computing one element matrix. Note that the main cost on triangles is writing down the answer.

triangles

degree	n	m	nm	MAPs
1	6	3	18	9
2	21	3	63	17
3	55	3	165	46

tetrahedra

degree	n	m	nm	MAPs
1	10	6	60	27
2	55	6	330	101
3	210	6	1260	370

Scalar weighted Laplacian

$$a_w(u,v) = \int_\Omega w \nabla u \cdot \nabla v \, \mathrm{d}x$$

$$A^0_{i\alpha} = \int_E \Phi_{\alpha_1}(X) \frac{\partial \Phi_{i_1}(X)}{\partial X_{\alpha_2}} \frac{\partial \Phi_{i_2}(X)}{\partial X_{\alpha_3}} \, \mathrm{d}X$$

$$\begin{aligned} G_e^\alpha &= w_{\alpha_1} \det F_e' \frac{\partial X_{\alpha_2}}{\partial x_\beta} \frac{\partial X_{\alpha_3}}{\partial x_\beta}. \\ &= w_{\alpha_1} (G^L)_e^{(\alpha_2,\alpha_3)} \end{aligned}$$

Note the outer product structure of G_e

Options

Do contraction all at once (must build G_K)

$$A_i^K = \sum_\alpha A^0_{i\alpha} G_K^\alpha$$

Contract with $(G^L)^K$ first (optimize), then contract with w.

$$\tilde{A}^K_{i,\alpha_1} = \sum_{\alpha_2,\alpha_3} A^0_{i\alpha} (G^L)_K^{(\alpha_2,\alpha_3)}$$

$$A_i^K = \sum_{\alpha_1} \tilde{A}_{i,\alpha_1} w_{\alpha_1}$$

Options (2)

Contract with w first (optimize), then $(G^L)_K$.

$$\hat{A}^K_{i,(\alpha_2,\alpha_3)} = \sum_{\alpha_1} A^0_{i\alpha} w_{\alpha_1}$$

$$A^K_i = \sum_{\alpha_2,\alpha_3} \hat{A}^K_{\alpha_2,\alpha_3} (G^L)_K (\alpha_2, \alpha_3$$

Must account for

- Computing outer product for G_K or not.
- Cost of optimized first phase.
- Cost of second, nonoptimized phase.

Results

In most cases, third approach gives the best reduction

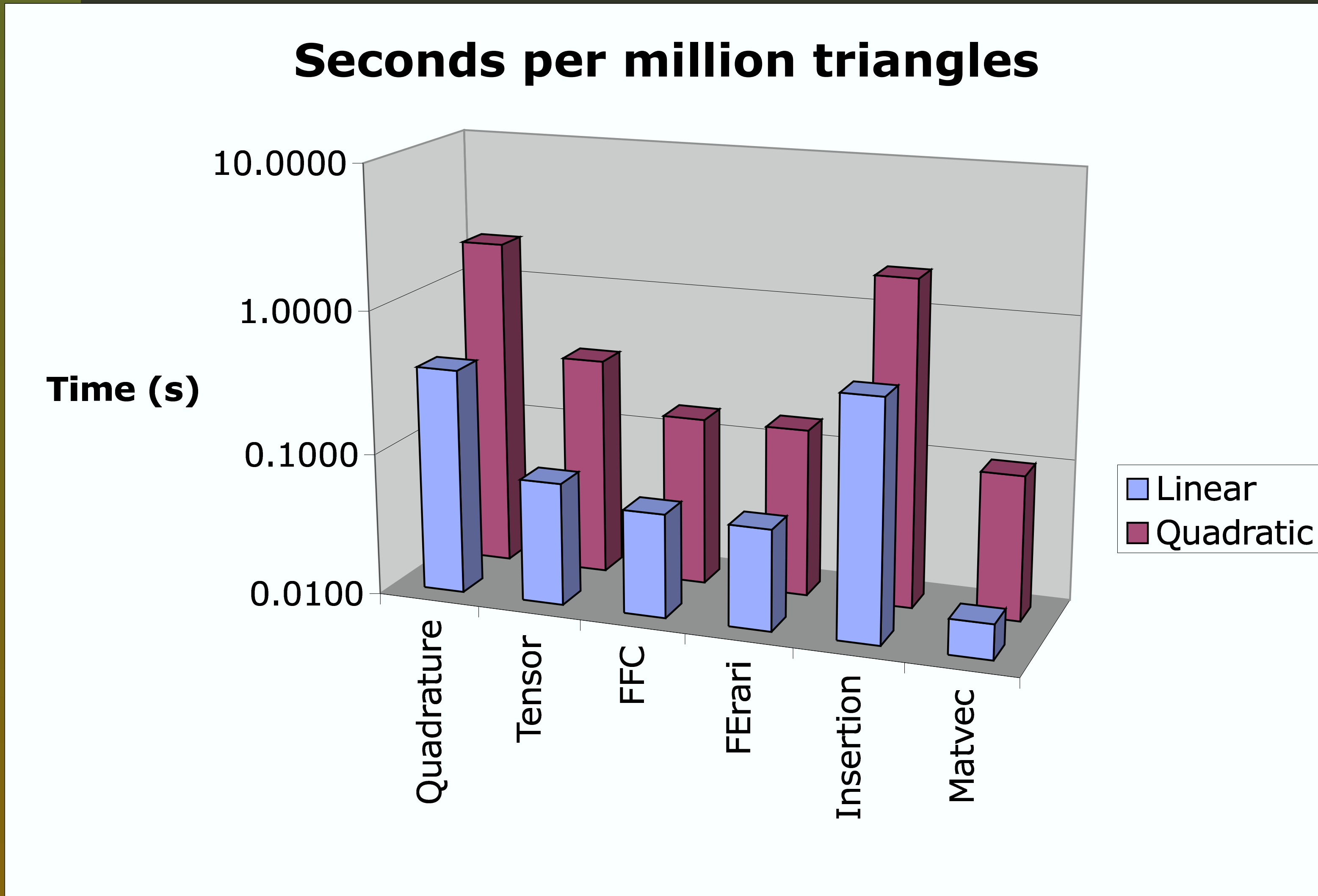
triangles

degree	n	m	nm	MAPs
1	6	9	54	25 (3)
2	21	18	378	201 (3)
3	55	30	1650	1064 (3)

tetrahedra

degree	n	m	nm	MAPs
1	10	24	240	67 (2)
2	55	60	3300	795 (3)
3	210	120	25200	8988 (3)

Performance



Geometric optimization

- Relations between three or more (e.g. linear dependence) tensors don't fit in graph-theoretic structure.
- What's the right model?
- Integrate geometric dependencies with CRRs

Partial geometry

- Let $|V| < \infty$ be a set and $L \subset \mathcal{P}(V)$ be a set of *lines*. Then (V, L) is a *partial geometry* if there is at most one line passing through each pair of points and each line contains at least three points.
- Note: typical geometries have every pair of points contained in *exactly* one line.
- Partial geometries are encoded by ternary relations on distinct unordered triples that satisfy $R\{u,v,w\} \wedge R\{v,w,x\} \rightarrow R\{u,v,x\} \wedge R\{u,w,x\}$
- Coplanarity is such a ternary relation.
- Can generalize to relations of higher arity.

Closure and generators

We define the *closure* of $S \subset V$, denoted by $\bar{S}$, recursively by

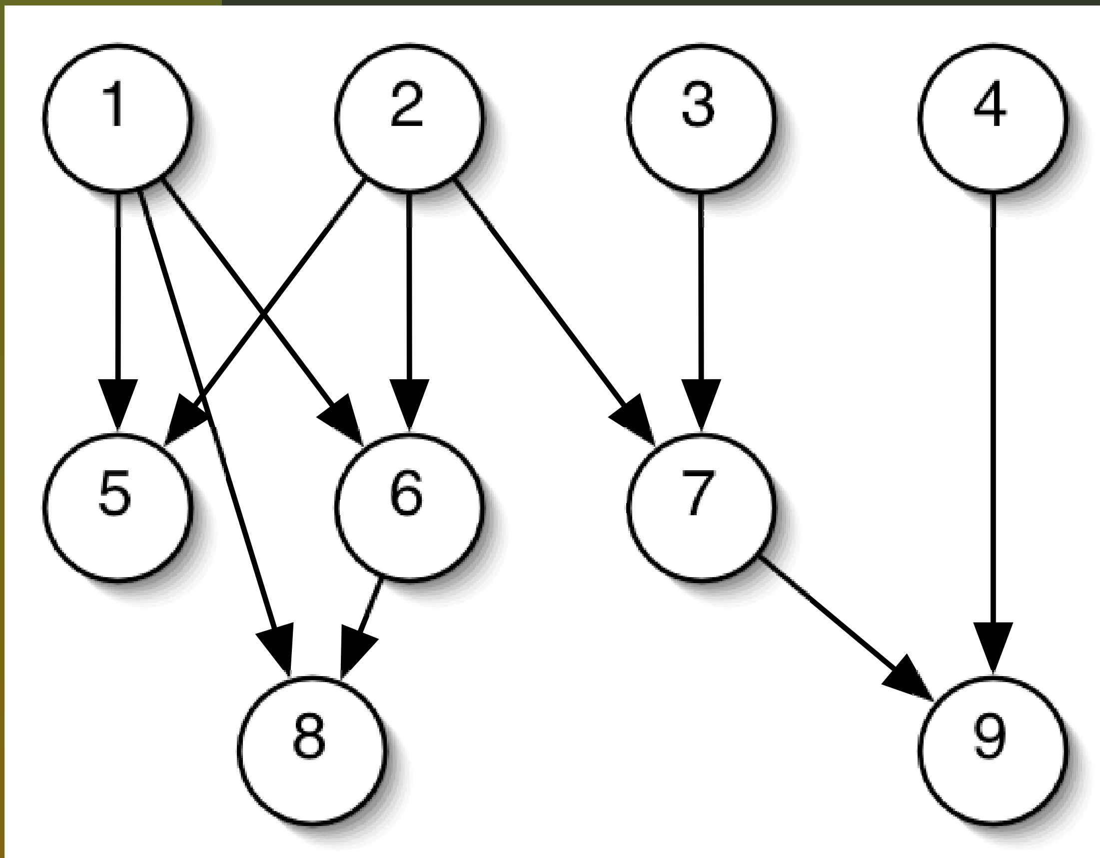
- $v \in S \rightarrow v \in \bar{S}$
- $z \in V \wedge \exists x, y \in \bar{S} \ni R\{x, y, z\} \rightarrow z \in \bar{S}$

We can also define *generators* for a set:

- If $\bar{S} = T \subset V$, we say S *generates* T.
- If S generates T and no subset of S generates T, then S is a *minimal generator* for T.

Minimal generators/optimization

Computing the closure of a set S gives a digraph:



- Topological sort resolves dependencies, sequences computation
- Want minimal generator that gets all of V and has

Minimal minimal generators

- Hardness unknown (so far)
- Greedy algorithm:
  - Add "most connected" point to the set of generators
  - Compute closure
  - Repeat until all items are generated or generators
- Don't know if this gets the minimal, but seems effective
- Geometric optimization not as effective as CRR

Combining approaches

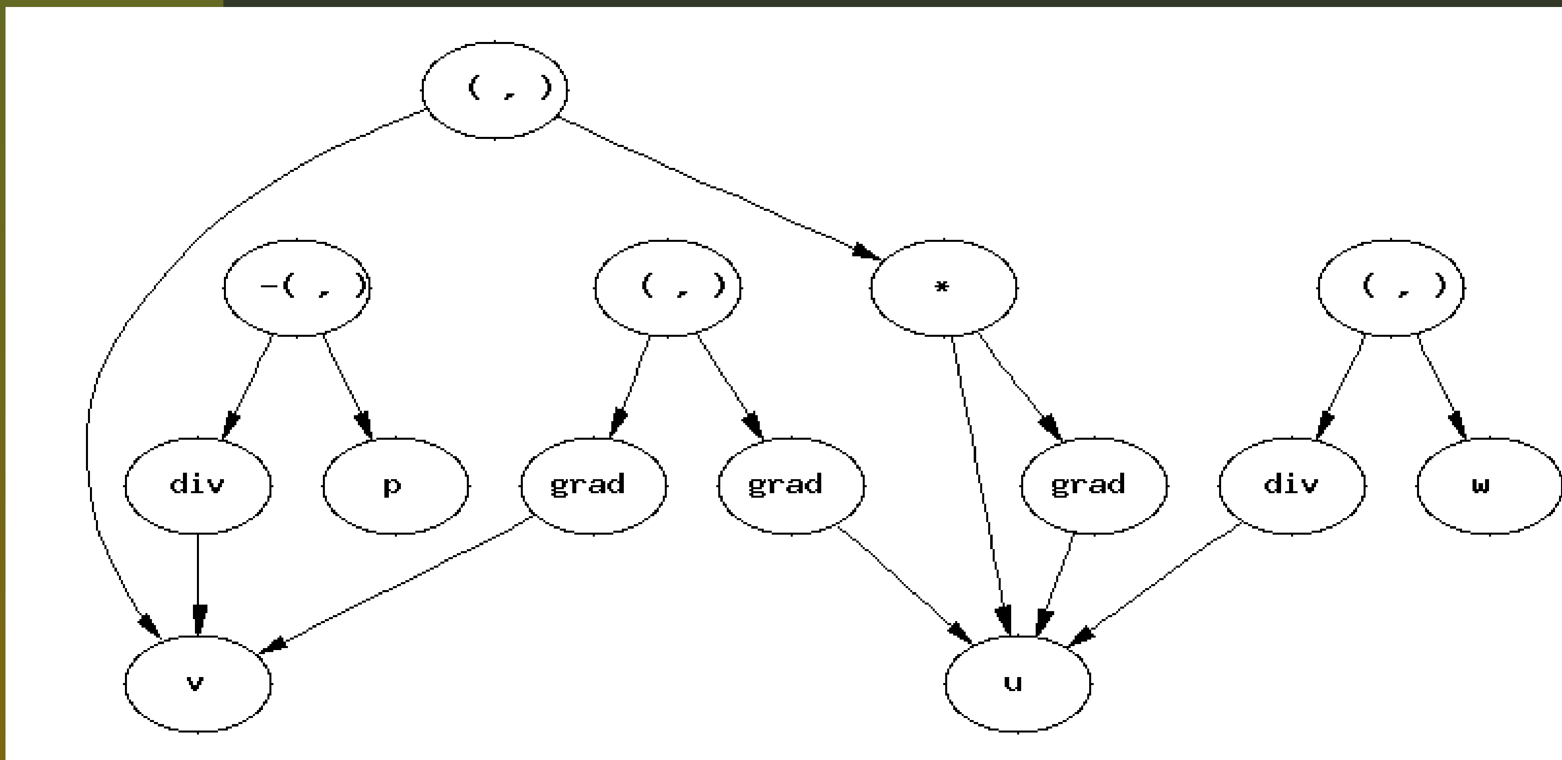
- Want a “minimum spanning hypertree”
- Each item
  - Is root (costs m) OR
  - Has one (binary) or two (geometric) parents
- This is probably NP-hard (optimization over all permutations of V)
- Simple modification of Prim’s algorithm is a first approximation algorithm
- Typically get about 25% additional reduction in flop count

Moving up a level

- Reasoned about low-level algorithms
- Can reason about “form syntax”
  - Represented as a DAG
  - DAG $\longrightarrow$ Sieve
  - Nonlinear coupling
  - Extracting logical blocks
  - Implementation still in progress

Abstract syntax graph

Incompressible Navier-Stokes equations:



Sieve

- Introduced by Knepley and Karpeev (TOMS, 2005) as a combinatorial/topological model for finite element meshes
- Based on covering relation
- Expresses dimensional/shape-independent meshes (and many interesting operations)
- Operations defined on *chains* (sets) of nodes in the graph.
- Allows construction of a *lattice* on the power set of a graph
- Also allows us to reason about abstract syntax.

Sieve operations

- `cone(u)`: in-neighbors of `u`
- `support(u)`: out-neighbors of `u`
- Define these on chains by union of nodewise results
- `closure(u)`: apply `cone` recursively, all points from which `u` is reachable
- `star(u)`: apply `support` recursively, all points reachable from `u`.
- These are extended to chains as well.

Sieve operations (2)

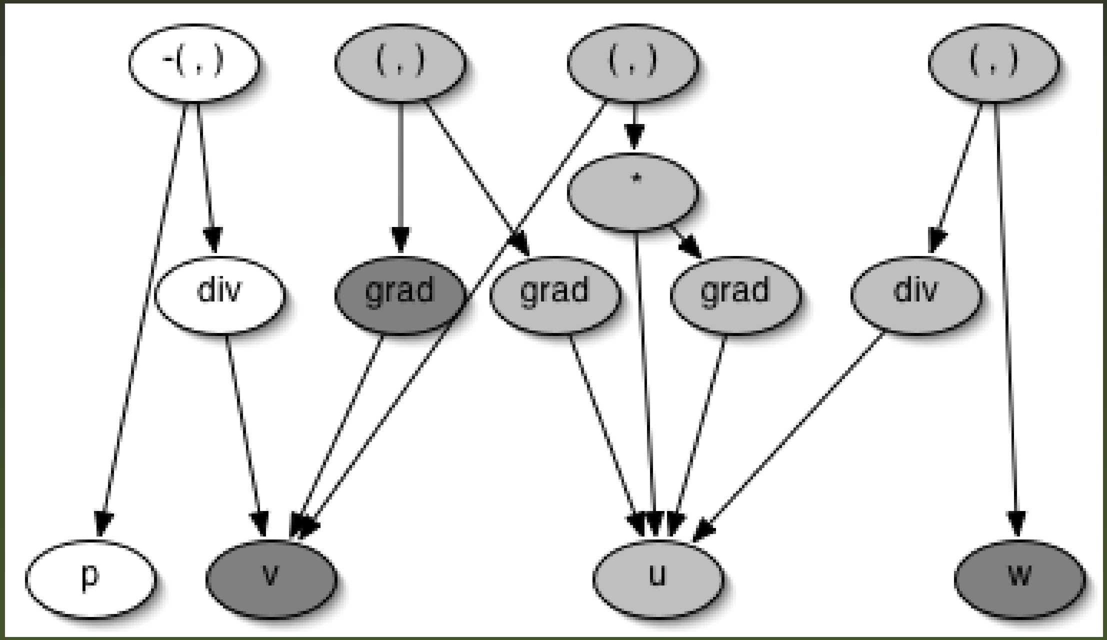
We can introduce lattice operations as follows

- `meet(u,v)` is the minimal separator – minimal set of points which, if removed, ensure that `u,v` are not both reachable from any point.
- `meet` defined on chains, `join` is `meet` on the dual graph.
- `meet`,`join` defined on chains by union

These operations are critical to reasoning about abstract syntax.

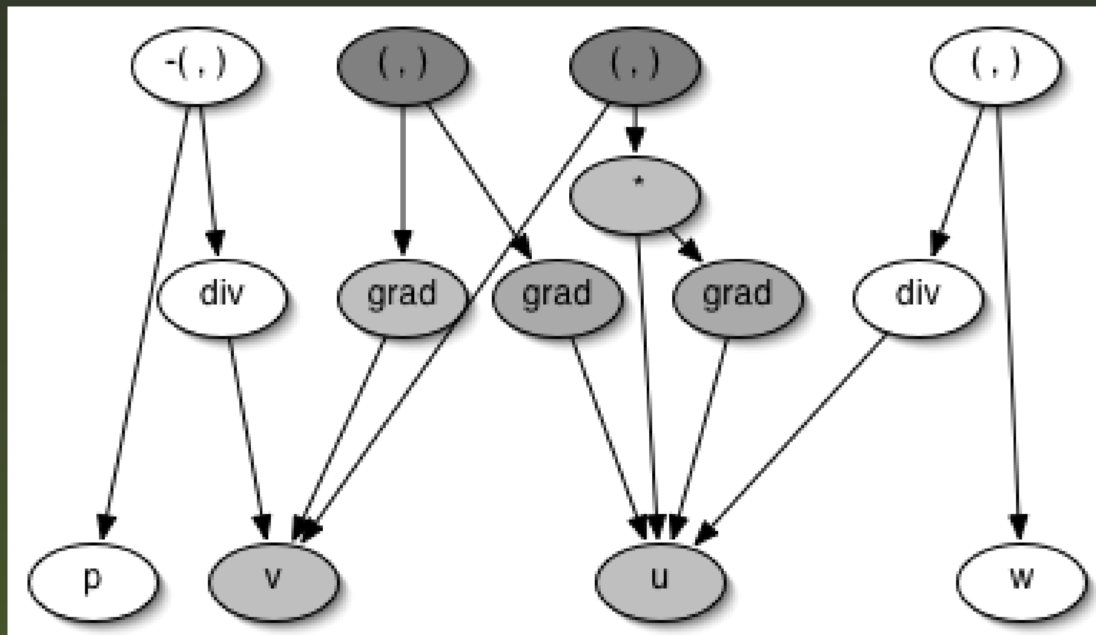
What equations contain u?

```
star(closure(u))
```



What equations couple u? and v

```
star(meet(u,v))
```



Other analyses

- `u`,`p` would couple nonlinearly if `meet(u,p)` are nonempty.
- Example: MHD, Lorenz force couples velocity, magnetic field
- polynomial nonlinearity in `u`. Need a slightly different operator `selfmeet(u)`.

Applications

- PDE language allows extraction of logical blocks
  - Schur-type solver/preconditioner
  - Pattern match against existing code
- Automatic/adaptive implicitness? (cf adaptive ODE)

Conclusions

- Numerical PDE rich in structure at *many levels*
- Potential for automation/optimization enriched if we let mathematics inform our software engineering (Mathematical software should be *mathematical*)
- Improve reliability and efficiency of scientists, new opportunities for numerical analysts (and other mathematicians, too)