Firedrake: composable abstractions for high performance finite element computations

Lawrence Mitchell$^1$ ∈ Firedrake team
12th September 2017

$^1$Department of Computing and Department of Mathematics, Imperial College London
[...] an automated system for the solution of partial differential equations using the finite element method.

- Written in Python.
- Finite element problems specified with embedded domain specific language, UFL (Alnæs, Logg, Ølgaard, Rognes, and Wells 2014) from the FEniCS project.
- Runtime compilation to low-level (C) code.
- Explicitly data parallel API.

A DSL for finite element computations

Find \((u, p, T) \in V \times W \times Q\) s.t.

\[
\int \nabla u \cdot \nabla v + (u \cdot \nabla u) \cdot v - p \nabla \cdot v + \frac{Ra}{Pr} T g \hat{z} \cdot v \, dx = 0
\]

\[
\int \nabla \cdot u q \, dx = 0
\]

\[
\int (u \cdot \nabla T) S + Pr^{-1} \nabla T \cdot \nabla S \, dx = 0
\]

\forall (v, q, T) \in V \times W \times Q

\[
\text{from firedrake import *}
\]
\[
\text{mesh} = \text{Mesh}(\ldots)
\]
\[
V = \text{VectorFunctionSpace(mesh, "CG", 2)}
\]
\[
W = \text{FunctionSpace(mesh, "CG", 1)}
\]
\[
Q = \text{FunctionSpace(mesh, "CG", 1)}
\]
\[
Z = V \ast W \ast Q
\]
\[
Ra = \text{Constant}(200)
\]
\[
Pr = \text{Constant}(6.18)
\]
\[
upT = \text{Function}(Z)
\]
\[
\text{u, p, T = split}(upT)
\]
\[
v, q, S = \text{TestFunctions}(Z)
\]
\[
\text{bcs} = [\ldots] \ # \ no-flow + temp \ gradient
\]
\[
\text{nullspace} = \text{MixedVectorSpaceBasis(}
\]
\[
\quad Z, [Z\text{.sub(0)}, \text{VectorSpaceBasis(constant=True)},
\]
\[
\quad \quad Z\text{.sub(2)})]
\]
\[
F = (\text{inner(grad(u), grad(v))} + \text{inner(dot(grad(u), u), v)} - \text{inner(p, div(v))} + \text{(Ra/Pr)*inner(T*g, v)} + \text{inner(div(u), q)} + \text{inner(dot(grad(T), u), S)} + \text{(1/Pr) * inner(grad(T), grad(S)))}*dx
\]
\[
\text{solve(F = 0, upT, bcs=bcs, nullspace=nullspace)}
\]
Lemma
Most research groups do not have the expertise to produce high performance simulations.

Corollary
If we want high performance expertise to be available to all model developers, we need a way of scaling the expertise.
Lemma
Most research groups do not have the expertise to produce high performance simulations.

Corollary
If we want high performance expertise to be available to all model developers, we need a way of scaling the expertise.

In Firedrake, we do this by synthesising efficient code with domain-specific compilers.
Lemma
Most research groups do not have the expertise to produce high performance simulations.

Corollary
If we want high performance expertise to be available to all model developers, we need a way of scaling the expertise.

In Firedrake, we do this by synthesising hopefully efficient code with domain-specific compilers.
Two stages of compilation

Local kernels: TSFC
Synthesise *element local* kernel from weak form.

Global iteration: PyOP2
Weave together *local kernel* with global iteration over some set of mesh entities (e.g. cells, exterior facets).
Global iteration

Local computation
Kernels do not know about global data layout.

- Kernel defines contract on local, packed, ordering.
- Global-to-local reordering/packing applied by runtime library.

Data parallel API
Application code does not specify explicit iteration order.

- Define data structures, then just “iterate”
- Lazy evaluation, permits loop tiling and fusion without changing application code.
Maintainability

With good abstractions, you write little code.

Library usability

- High-level language enables rapid model development
- Ease of experimentation
- Small model code base

Library development

- Automation of complex optimisations
- Exploit expertise across disciplines
- Small library code base
Maintainability

With good abstractions, you write little code.

<table>
<thead>
<tr>
<th>Core Firedrake</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firedrake</td>
<td>12000</td>
</tr>
<tr>
<td>PyOP2</td>
<td>5200</td>
</tr>
<tr>
<td>TSFC</td>
<td>4000</td>
</tr>
<tr>
<td>finat</td>
<td>1300</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>22500</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shared with FEniCS</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIAT</td>
<td>4000</td>
</tr>
<tr>
<td>UFL</td>
<td>13000</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>17000</strong></td>
</tr>
</tbody>
</table>
Maintainability

With good abstractions, you write little code.

**Thetis**
/github.com/thetisproject/thetis/
- (2+1)D unstructured coastal ocean model, equal order DG
- 7000 LOC
- 4-8x faster than previous code in group (same numerics)

**Gusto**
/www.firedrakeproject.org/gusto/
- (2+1)D atmospheric dynamical core using compatible FE
- Implements Met Office “Gung Ho” numerics
- 2000 LOC
We first transform to reference space

\[
\int_e uv \, dx \rightarrow \int_e \tilde{u}\tilde{v} \det \frac{\partial X}{\partial x} \, dX
\]

and then evaluate integrals with quadrature rule \{(w_q, x_q)\}.

\[
A_{i,j} = \sum_q w_q E_i(x_q) E_j(x_q) \det \begin{bmatrix}
\sum_r C_1^r(x_q) c_r & \sum_r C_1^r(x_q) c_r \\
\sum_r C_2^r(x_q) c_r & \sum_r C_2^r(x_q) c_r
\end{bmatrix}
\]

- \(E_i(x_q)\) tabulation of \(i\)th basis function at \(x_q\).
- \(c\) the vector of basis function coefficients of the coordinate field.
- \(C_a^r(x_q)\) the tabulation of the first derivative of the \(a\)th component of the \(r\)th basis function of coordinate element at \(x_q\).
Compiling finite element kernels

\[ A_{i,j} = \sum_q w_q E_i(x_q) E_j(x_q) \det \begin{bmatrix} \sum_r C^1_r(x_q)c_r & \sum_r C^1_r(x_q)c_r \\ \sum_r C^2_r(x_q)c_r & \sum_r C^2_r(x_q)c_r \end{bmatrix} \]

- Naïve code generation transforms this tensor algebra expression into low-level C code.
- But there are likely opportunities for optimisation.
- For example, \( \det J \) constant for affine geometries.
- Others available, depending on structure in \( E, C, \{q\} \).
Optimisations in TSFC

Generic

• Flop reduction via factorisation, code motion, and CSE.

• Alignment and padding for vectorisation (either intrinsics or rely on C compiler).

Structured basis

• Structure (e.g. tensor products) preserved in intermediate representation in TSFC, enables new optimisation passes.

• Sum factorisation and spectral underintegration.
Sum factorisation

- Consider evaluating residual

\[ \mathcal{F}_j = \sum_{q} w_q \phi_j(x_q) f_j \]

- Form compiler obtains \( \phi_j(x_q) \) from element library.

**old** FIAT can only provide the array substitution \( \phi_j(x_q) \rightarrow \Phi_{q,j} \)

**new** FIAT, provides *symbolic expression* \( \phi_j(x_q) \rightarrow \Phi^1_{j_1,q_1} \Phi^2_{j_2,q_2} \)

- Now a compiler can transform the sums

\[
\mathcal{F}_{(j_1,j_2)} = \sum_{(q_1,q_2)} w_{q_1} w_{q_2} \Phi^1_{j_1,q_1} \Phi^2_{j_2,q_2} f_{(j_1,j_2)} \\
= \sum_{q_1} w_{q_1} \Phi_{j_1,q_1} \sum_{q_2} \Phi_{j_2,q_2} f_{(j_1,j_2)}
\]
Sum factorisation II

- Improves complexity $O((p + 1)^{d-1})$-fold.
- Gives *optimal complexity* evaluation for matrix assembly, matrix-vector products, and residual evaluation.
- For a degree $p$ approximation on a $d$-dimensional tensor product cell we have

<table>
<thead>
<tr>
<th>Method</th>
<th>Build operator (FLOPs)</th>
<th>MatVec (FLOPs)</th>
<th>Mem refs (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve assembled</td>
<td>$O(p^{3d})$</td>
<td>$O(p^{2d})$</td>
<td>$O(p^{2d})$</td>
</tr>
<tr>
<td>SF assembled</td>
<td>$O(p^{2d+1})$</td>
<td>$O(p^{2d})$</td>
<td>$O(p^{2d})$</td>
</tr>
<tr>
<td>Naïve matrix free</td>
<td>0</td>
<td>$O(p^{2d})$</td>
<td>$O(p^d)$</td>
</tr>
<tr>
<td>SF matrix free</td>
<td>0</td>
<td>$O(p^{d+1})$</td>
<td>$O(p^d)$</td>
</tr>
</tbody>
</table>
Not just for $Q$ and $dQ$

Find $u \in V \subset H(\text{curl})$ s.t.

$$\int \text{curl} u \cdot \text{curl} v \, dx = \int B \cdot v \, dx \quad \forall v \in V.$$ 

NCE = FiniteElement("NCE", hexahedron, degree)
Q = VectorElement("Q", hexahedron, degree)
u = Coefficient(NCE) # Solution variable
B = Coefficient(Q) # Coefficient in $H^1$
v = TestFunction(NCE)
F = (dot(curl(u), curl(v)) - dot(B, v)) * dx

FLOPs for single-cell residual

- $O(p^6)$
- $O(p^4)$

Polynomial degree

- Naïve (no sum fact)
- With sum factorisation
Not just for $Q$ and $dQ$

Find $u \in V \subset H(\text{curl})$ s.t.

$$\int \text{curl} \ u \cdot \text{curl} \ v \, dx = \int B \cdot v \, dx \quad \forall v \in V.$$ 

NCE = FiniteElement("NCE", hexahedron, degree)
Q = VectorElement("Q", hexahedron, degree)
u = Coefficient(NCE) # Solution variable
B = Coefficient(Q) # Coefficient in $H^1$
v = TestFunction(NCE)
F = (dot(curl(u), curl(v)) - dot(B, v))*dx

- TSFC obtains optimal complexity evaluation
- In progress: is the constant factor good?
- Much still to be done in terms of vectorisation.

FLOPs for single-cell residual

![Graph showing FLOPs vs Polynomial degree]

$\mathcal{O}(p^6)$

$\mathcal{O}(p^4)$

12
Conclusions

• Firedrake provides a layered set of abstractions for finite element computations.
• By capturing mathematical structure in code, we can automate many transformations that people do by hand.
• Enables automated provision of “HPC expertise” to model developers.
• Good for experimentation from laptop to supercomputer.

Future developments

• Better support for subdomains and domain-decomposition PCs
  Extending ideas from Kirby and LM. arXiv:1706.01346 [cs.MS]
• Code generation for wide vector lanes
• ...

13
References


