Firedrake: composable abstractions for high performance finite element computations

Lawrence Mitchell[1] $\in$ 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.

F. Rathgeber, D.A. Ham, **LM**, M. Lange, F. Luporini, A.T.T. McRae, G.-T. Bercea, G.R. Markall, P.H.J. Kelly. TOMS, 2016. `arXiv:1501.01809 [cs.MS]`

A DSL for finite element computations

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

$$\int \nabla u \cdot \nabla v + (u \cdot \nabla u) \cdot v$$
$$-p \nabla \cdot v + \frac{\mathrm{Ra}}{\mathrm{Pr}} T g \hat{z} \cdot v \,\mathrm{d}x = 0$$
$$\int \nabla \cdot u q \,\mathrm{d}x = 0$$
$$\int (u \cdot \nabla T) S + \mathrm{Pr}^{-1} \nabla T \cdot \nabla S \,\mathrm{d}x = 0$$
$$\forall (v, q, T) \in V \times W \times Q$$

```
from firedrake import *
mesh = Mesh(...)
V = VectorFunctionSpace(mesh, "CG", 2)
W = FunctionSpace(mesh, "CG", 1)
Q = FunctionSpace(mesh, "CG", 1)
Z = V * W * Q
Ra = Constant(200)
Pr = Constant(6.18)
upT = Function(Z)
u, p, T = split(upT)
v, q, S = TestFunctions(Z)
bcs = [...] # no-flow + temp gradient
nullspace = MixedVectorSpaceBasis(
   Z, [Z.sub(0), VectorSpaceBasis(constant=True),
       Z.sub(2)])
F = (inner(grad(u), grad(v))
     + inner(dot(grad(u), u), v)
     - inner(p, div(v))
     + (Ra/Pr)*inner(T*g, v)
     + inner(div(u), q)
     + inner(dot(grad(T), u), S)
     + (1/Pr) * inner(grad(T), grad(S)))*dx

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.

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).

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

With good abstractions, you write little code.

Core Firedrake

Component	LOC
Firedrake	12000
PyOP2	5200
TSFC	4000
finat	1300
Total	22500

Shared with FEniCS

Component	LOC
FIAT	4000
UFL	13000
Total	17000

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

$$\int_e uv \, \mathrm{d}x \to \int_e \tilde{u}\tilde{v} \det \frac{\partial x}{\partial X} \, \mathrm{d}X$$

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

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

- $E_i(x_q)$ tabulation of ith basis function at x_q.
- c the vector of basis function coefficients of the coordinate field.
- $C_r^a(x_q)$ the tabulation of the first derivative of the ath component of the rth basis function of coordinate element at x_q.

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

- 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, $\{q\}$.

Generic

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

Luporini, Ham, and Kelly. TOMS 2017. `arXiv:1604.05872 [cs.MS]`

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

Luporini, Varbanescu, et al. TACO 2015. `doi:10.1145/2687415`

Structured basis

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

Homolya, **LM**, Luporini, Ham. `arXiv:1705.03667 [cs.MS]`

- Sum factorisation and spectral underintegration.

Homolya, Kirby, Ham. In preparation.

- 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 $\phi_j(x_q) \to \Phi_{q,j}$

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

- Now a compiler can transform the sums

$$\begin{aligned}\mathcal{F}_{(j_1,j_2)} &= \sum_{(q_1,q_2)} w_{q_1} w_{q_2} \Phi^1_{j_1,q_1} \Phi^2_{j_2,q_2} f_{(j_1,j_2)} \\ &= \sum_{q_1} w_{q_1} \Phi_{j_1,q_1} \sum_{q_2} \Phi_{j_2,q_2} f_{(j_1,j_2)}\end{aligned}$$

Sum factorisation II

- Improves complexity $\mathcal{O}((p+1)^{d-1})$-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

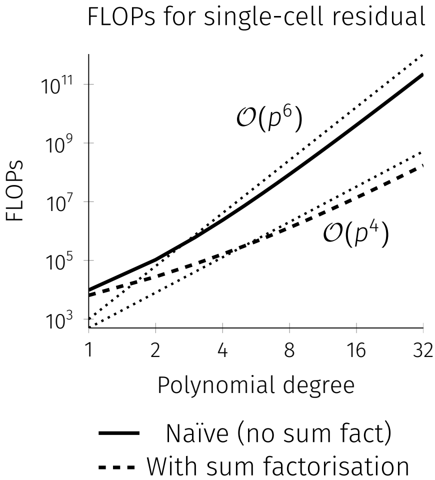
Method	Build operator (FLOPs)	MatVec (FLOPs)	Mem refs (bytes)
Naïve assembled	$\mathcal{O}(p^{3d})$	$\mathcal{O}(p^{2d})$	$\mathcal{O}(p^{2d})$
SF assembled	$\mathcal{O}(p^{2d+1})$	$\mathcal{O}(p^{2d})$	$\mathcal{O}(p^{2d})$
Naïve matrix free	0	$\mathcal{O}(p^{2d})$	$\mathcal{O}(p^{d})$
SF matrix free	0	$\mathcal{O}(p^{d+1})$	$\mathcal{O}(p^{d})$

Not just for Q and dQ

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

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

```
NCE = FiniteElement("NCE", hexahedron, degree)
Q = VectorElement("Q", hexahedron, degree)
u = Coefficient(NCE) # Solution variable
B = Coefficient(Q)   # Coefficient in H1
v = TestFunction(NCE)
F = (dot(curl(u), curl(v)) - dot(B, v))*dx
```



Not just for Q and dQ

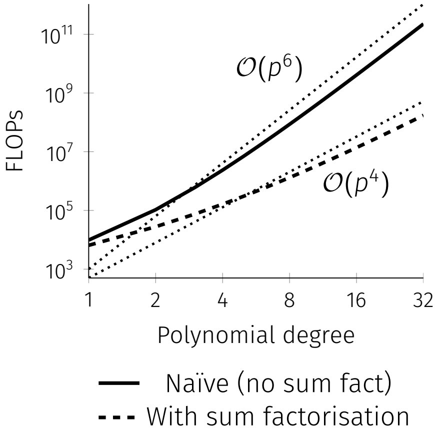
Find $u \in V \subset H(curl)$ s.t.

$$\int \mathrm{curl}\, u \cdot \mathrm{curl}\, v \, \mathrm{d}x = \int B \cdot v \, \mathrm{d}x \quad \forall v \in V.$$

```
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



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
- ...

EPSRC
Engineering and Physical Sciences Research Council

References

Alnæs, M. S., A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells (2014). "Unified Form Language: A Domain-specific Language for Weak Formulations of Partial Differential Equations". *ACM Trans. Math. Softw.* 40. doi:10.1145/2566630. arXiv: 1211.4047 [cs.MS].

Homolya, M., L. Mitchell, F. Luporini, and D. A. Ham (2017). *TSFC: a structure-preserving form compiler*. arXiv: 1705.03667 [cs.MS].

Kirby, R. C. and L. Mitchell (2017). *Solver composition across the PDE/linear algebra barrier*. arXiv: 1706.01346 [cs.MS].

Luporini, F., D. A. Ham, and P. H. J. Kelly (2017). "An algorithm for the optimization of finite element integration loops". *ACM Transactions on Mathematical Software* 44. doi:10.1145/3054944. arXiv: 1604.05872 [cs.MS].

Luporini, F., A. L. Varbanescu, F. Rathgeber, G.-T. Bercea, J. Ramanujam, D. A. Ham, and P. H. J. Kelly (2015). "Cross-Loop Optimization of Arithmetic Intensity for Finite Element Local Assembly". *ACM Trans. Archit. Code Optim.* 11. doi:10.1145/2687415.

Rathgeber, F., D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. McRae, G.-T. Bercea, G. R. Markall, and P. H. J. Kelly (2016). "Firedrake: automating the finite element method by composing abstractions". *ACM Transactions on Mathematical Software* 43. doi:10.1145/2998441. arXiv: 1501.01809 [cs.MS].