If you've scheduled loops, you've gone too far

Lawrence Mitchell[1,*]

24th October 2017

[1]Departments of Computing and Mathematics, Imperial College London
*lawrence.mitchell@imperial.ac.uk

Engineering and Physical Sciences Research Council

Write data parallel code

as any fule kno

Write data/task parallel code

as any fule kno

I’ve got 99 problems

Lemma

You can’t trust computational scientists to write good code.

Corollary

Make it “impossible” to not write good code.

DSLs for finite elements

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

$$
\begin{aligned}
\int \nabla u \cdot \nabla v + (u \cdot \nabla u) \cdot v \\
-p \nabla \cdot v + \frac{\mathrm{Ra}}{\mathrm{Pr}} T g \hat{z} \cdot v \, \mathrm{d}x = 0 \\
\int \nabla \cdot u q \, \mathrm{d}x = 0 \\
\int (u \cdot \nabla T) S + \mathrm{Pr}^{-1} \nabla T \cdot \nabla S \, \mathrm{d}x = 0
\end{aligned}
$$

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

DSLs for finite elements

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

$$\int \nabla u \cdot \nabla v + (u \cdot \nabla u) \cdot v$$

$$-p\nabla \cdot v + \frac{\mathrm{Ra}}{\mathrm{Pr}} T g\hat{z} \cdot v \,\mathrm{d}x = 0$$

$$\int \nabla \cdot u q \,\mathrm{d}x = 0$$

$$\int (u \cdot \nabla T) S + \mathrm{Pr}^{-1} \nabla T \cdot \nabla S \,\mathrm{d}x = 0$$

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

```python
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)
```

+ a DSL for solver configuration

```
-snes_type newtonls
-snes_rtol 1e-8
-snes_linesearch_type basic
-ksp_type fgmres
-ksp_gmres_modifiedgramschmidt
-mat_type matfree
-pc_type fieldsplit
-pc_fieldsplit_type multiplicative
-pc_fieldsplit_0_fields 0,1
-pc_fieldsplit_1_fields 2
-prefix_push fieldsplit_1_
  -ksp_type gmres
  -ksp_rtol 1e-4,
  -pc_type python
  -pc_python_type firedrake.AssembledPC
  -assembled_mat_type aij
  -assembled_pc_type telescope
  -assembled_pc_telescope_reduction_factor 6
  -assembled_telescope_pc_type hypre
  -assembled_telescope_pc_hypre_boomeramg_P_max 4
  -assembled_telescope_pc_hypre_boomeramg_agg_nl 1
  -assembled_telescope_pc_hypre_boomeramg_agg_num_paths 2
  -assembled_telescope_pc_hypre_boomeramg_coarsen_type HMIS
  -assembled_telescope_pc_hypre_boomeramg_interp_type ext+i
  -assembled_telescope_pc_hypre_boomeramg_no_CF True
-prefix_pop
-prefix_push fieldsplit_0_
  -ksp_type gmres
  -ksp_gmres_modifiedgramschmidt
  -ksp_rtol 1e-2
  -pc_type fieldsplit
  -pc_fieldsplit_type schur
  -pc_fieldsplit_schur_fact_type lower
  -prefix_push fieldsplit_0_
    -ksp_type preonly
    -pc_type python
    -pc_python_type firedrake.AssembledPC
    -assembled_mat_type aij
    -assembled_pc_type hypre
    -assembled_pc_hypre_boomeramg_P_max 4
    -assembled_pc_hypre_boomeramg_agg_nl 1
    -assembled_pc_hypre_boomeramg_agg_num_paths 2
    -assembled_pc_hypre_boomeramg_coarsen_type HMIS
    -assembled_pc_hypre_boomeramg_interp_type ext+i
    -assembled_pc_hypre_boomeramg_no_CF
  -prefix_pop
  -prefix_push fieldsplit_1_
    -ksp_type preonly
    -pc_type python
    -pc_python_type firedrake.PCDPC
    -pcd_Fp_mat_type matfree
    -pcd_Kp_ksp_type preonly
    -pcd_Kp_mat_type aij
    -pcd_Kp_pc_type telescope
    -pcd_Kp_pc_telescope_reduction_factor 6
    -pcd_Kp_telescope_pc_type ksp
    -pcd_Kp_telescope_ksp_ksp_max_it 3
    -pcd_Kp_telescope_ksp_ksp_type richardson
    -pcd_Kp_telescope_ksp_pc_type hypre
    -pcd_Kp_telescope_ksp_pc_hypre_boomeramg_P_max 4
    -pcd_Kp_telescope_ksp_pc_hypre_boomeramg_agg_nl 1
    -pcd_Kp_telescope_ksp_pc_hypre_boomeramg_agg_num_paths 2
    -pcd_Kp_telescope_ksp_pc_hypre_boomeramg_coarsen_type HMIS
    -pcd_Kp_telescope_ksp_pc_hypre_boomeramg_interp_type ext+i
    -pcd_Kp_telescope_ksp_pc_hypre_boomeramg_no_CF
    -pcd_Mp_mat_type aij
    -pcd_Mp_ksp_type richardson
    -pcd_Mp_pc_type sor
    -pcd_Mp_ksp_max_it 2
  -prefix_pop
-prefix_pop
```

Firedrake `www.firedrakeproject.org`

[...] *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]`

Code transformation

- Represent fields as expansion in some basis $\{\phi_i\}$ for the discrete space.
- Integrals are computed by numerical quadrature on mesh elements.

$$\int_\Omega F(\phi_i)\phi_j \,\mathrm{d}x \to \sum_{e\in\mathcal{T}} \sum_q w_q F(\phi_i(q))\phi_j(q)$$

- Need to evaluate ϕ_j and $F(\phi_i)$, defined by basis coefficients $\{f_i\}_{i=1}^N$, at quadrature points $\{q_j\}_{j=1}^Q$.

$$\mathcal{F}_q = \left[\Phi f\right]_q = \sum_i \phi_{i,q} f_i$$

- Φ is a $Q \times N$ matrix of basis functions evaluated at quadrature points.

How fast can you do dgemv?

- For degree p elements in d dimensions. $N, Q = \mathcal{O}(p^d)$.

How fast can you do dgemv?

- For degree p elements in d dimensions. $N, Q = \mathcal{O}(p^d)$.
- So I need $\mathcal{O}(p^{2d})$ operations. Right?

How fast can you do dgemv?

- For degree p elements in d dimensions. $N, Q = \mathcal{O}(p^d)$.
- So I need $\mathcal{O}(p^{2d})$ operations. Right?
- Well not always. . . .

Exploiting structure

Often, ϕ might have a tensor decomposition.

$$\phi_{i,q}(x, y, \dots) := \varphi_{j,p}(x)\varphi_{k,r}(y)\dots$$

and so

$$\begin{aligned}\mathcal{F}_{(p,r)} &= \sum_{j,k} \phi_{(j,k),(p,r)} f_{j,k} \\ &= \sum_{j,k} \varphi_{j,p}\varphi_{k,r} f_{j,k} \\ &= \sum_{j} \varphi_{j,p} \sum_{k} \varphi_{k,r} f_{j,k}\end{aligned}$$

at the cost of some temporary storage, this requires only $\mathcal{O}(dp^{d+1})$ operations.

Tensions

- You want the granularity of the data parallel operation to be small
- That way the programmer has less chance to get it wrong
- But, can you then get the “good” algorithm?

Tensions

- You want the granularity of the data parallel operation to be small
- That way the programmer has less chance to get it wrong
- But, can you then get the "good" algorithm?

Will your compiler hoist into array temporaries?

```
for (p = 0; p < L; p++)
 for (r = 0; r < L; r++)
  for (j = 0; j < M; j++)
   for (k = 0; k < M; k++)
    F[L*p+r][j*M+k] += f(p,j)*g(r,k)
```

Will your blas library notice if `PHI` has Kronecker-product structure?

```
double PHI[L][M] = {{ ... }};
dgemv(PHI, Fi, Fq)
```

Scheduling

- Front-end DSL matches finite elements
- Compiler frontend removes finite element specific constructs $\to$ DAG representation of tensor-algebra.
- These operations can have structure (e.g. tensor-product decomposition)
- Transformations on the DAG to minimise op-count, perhaps promote vectorisation.
- Scheduling $\leftrightarrow$ topological sort of DAG
- Opportunity to introduce hardware- and problem-guided heuristics, and optimisation passes

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

Homolya, Kirby, Ham. In preparation

DSLs are handcuffs

- ✓ A DSL should elegantly capture mathematical structure
- ✓ things expressible in the mathematics can be compiled to efficient code *and algorithms!*
- ✗ All else cannot be compiled, need graceful degradation.
- ✗/✓ Greatest advantages come when you incorporate them *at the top level*.

Thanks!

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].

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].