Imperial College London

Firedrake: Burning the Thread at Both Ends

M. Lange[1] G. J. Gorman[1]

[1]AMCG, Imperial College London

April 13 2016

To Thread or Not To Thread

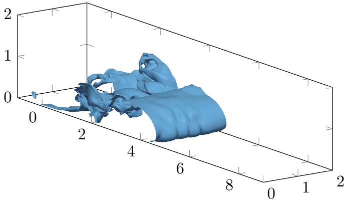
In order to thread the application . . .

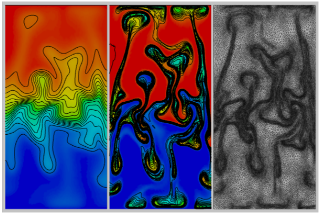
- A while ago, everybody wanted threading:
  - Utilise shared memory parallelism
  - Avoid MPI communication overhead
  - Improved memory footprint

- And it was supposed to be easy:

```
#pragma openmp for
```

- **Fluidity**: A widely used finite element code:
  - CFD, ocean modelling, geophysical flows, renewable energies, reservoir modelling, . . .
  - Adaptive anisotropic mesh refinement





To Thread or Not To Thread

. . . we need to thread the solver

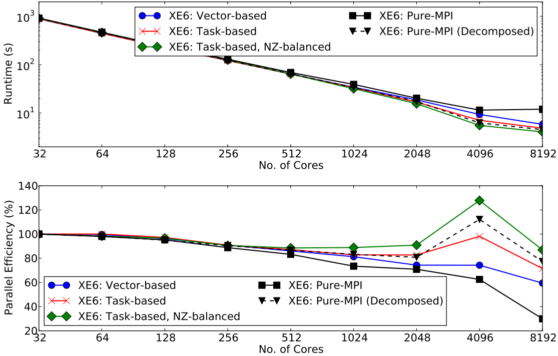
- **PETSc-OMP**:
  - An OpenMP threaded fork of PETSc-3.3
  - Low-level threading on Mat and Vec objects
- **Optimised sparse MatVec**
  - Explicit computation-communication overlap
  - Fined-grained load balance based on non-zero weights

PETSc-OMP IS NOT SUPPORTED ANYMORE!

- Was superseded by PETSc-Threadcomm
- Threadcomm already decommissioned

Imperial College London

Sparse MatVec results on Cray XE6

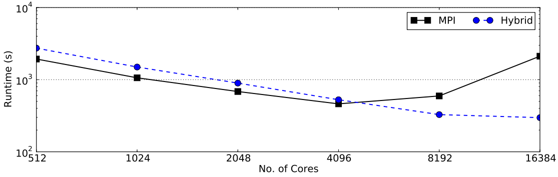


It's extremely hard to beat pure MPI!

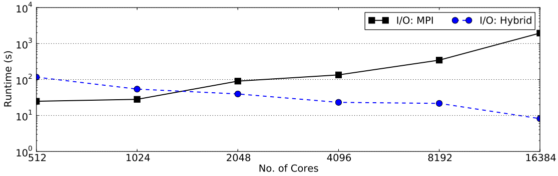
[1] M. Lange, G. Gorman, M. Weiland, L. Mitchell, and J. Southern. *"Supercomputing: 28th ISC 2013. Proceedings"*, chapter "Achieving Efficient Strong Scaling with PETSc Using Hybrid MPI/OpenMP Optimisation", pages 97–108. Springer, 2013

Fluidity performance on Cray XE6

Total run-time for 10 time-steps



I/O for initial mesh read



[1] X. Guo, M. Lange, G. Gorman, L. Mitchell, and M. Weiland. Developing a scalable hybrid MPI/OpenMP unstructured finite element model. *Computers & Fluids*, 110(0):227 – 234, 2015. ParCFD 2013

Fluidity performance on Cray XE6

Hybrid MPI-OpenMP looks faster at scale, **but** . . .

- Huge gains due to initial mesh I/O
  - Fluidity does off-line mesh decomposition
  - Partitioning **and halo** read from file
  - Using threads we need less partitions (x8)
- Sparse MatVec beats pure MPI
  - Only in strong scaling limit with little local work
  - Need threading to enforce asynchronous communication
  - Improvement due to better load balance, **not** MPI overheads!

No actual gain from threading!

- We just ameliorated some other underlying problem

Threading: Should we even care?

Threading is never the whole story . . .

- What is my application really limited by?
  - Different tasks can have different limitations (flops vs. bandwidth)
  - Profiling (roofline plots, analysis tools) must guide optimisation!
- Can we do better algorithmically?
  - Am I using the right numerical scheme?
  - Can I use better solvers?
- What about data-intensive tasks?
  - Is my communication model appropriate?
  - Am I doing I/O right? Are there better file formats?

. . . but threading looks so much easier!

- Changing any of the above is invasive
- Fundamental changes are impractical in monolithic codes

Firedrake - A finite element framework

Automated symbolic computation[1]

- Re-envisioned FEniCS/DOLFIN[2]

$$\phi^{n+1/2} = \phi^n - \frac{\Delta t}{2} p^n$$
$$p^{n+1} = p^n + \frac{\int_\Omega \nabla \phi^{n+1/2} \cdot \nabla v \, \mathrm{d}x}{\int_\Omega v \, \mathrm{d}x} \quad \forall v \in V$$
$$\phi^{n+1} = \phi^{n+1/2} - \frac{\Delta t}{2} p^{n+1}$$

where

$$\nabla \phi \cdot n = 0 \text{ on } \Gamma_N$$
$$p = \sin(10\pi t) \text{ on } \Gamma_D$$

```
from firedrake import *
mesh = Mesh("wave_tank.msh")
V = FunctionSpace(mesh, 'Lagrange', 1)
p = Function(V, name="p")
phi = Function(V, name="phi")
u = TrialFunction(V)
v = TestFunction(V)
p_in = Constant(0.0)
bc = DirichletBC(V, p_in, 1)
T = 10.
dt = 0.001
t = 0
while t <= T:
    p_in.assign(sin(2*pi*5*t))
    phi -= dt / 2 * p
    p += assemble(dt * inner(grad(v), grad(phi))*dx) \
         / assemble(v*dx)
    bc.apply(p)
    phi -= dt / 2 * p
    t += dt
```

[1] F. Rathgeber, D. Ham, L. Mitchell, M. Lange, F. Luporini, A. McRae, G. Bercea, G. Markall, and P. Kelly. Firedrake: Automating the finite element method by composing abstractions. *Submitted to ACM TOMS*, 2015

[2] A. Logg, K.-A. Mardal, and G. Wells. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012

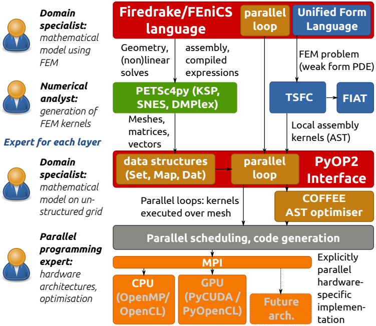
Firedrake - A finite element framework

Automated symbolic computation[1]

- Implements UFL[2], a finite element DSL embedded in Python
- Run-time C code generation
- PyOP2: Assembly kernel execution framework

Separation of concerns

- Expert for each layer
- Use third-party packages
    - "Write as little code as possible"



[1] F. Rathgeber, D. Ham, L. Mitchell, M. Lange, F. Luporini, A. McRae, G. Bercea, G. Markall, and P. Kelly. Firedrake: Automating the finite element method by composing abstractions. *Submitted to ACM TOMS*, 2015

[2] M. Alnæs, A. Logg, K. Ølgaard, M. Rognes, and G. Wells. Unified Form Language: A domain-specific language for weak formulations of partial differential equations. *ACM Transactions on Mathematical Software (TOMS)*, 40(2):9, 2014

Imperial College
London

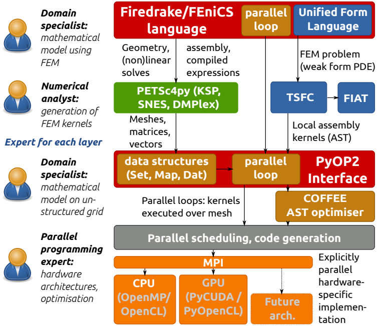
Firedrake - A finite element framework

End-to-end optimisation

- Exploration of numerical schemes
- Automated parallelisation
- Data layout optimisations
- Automated kernel optimisation

Parallelisation model

- Mostly MPI on CPUs
  - We have threads, but no gains
- Extendable to MPI+X, or just X
  - for some unknown X
- **Model definition doesn't change!**
  - Can even adjust numerics if needed



Imperial College London

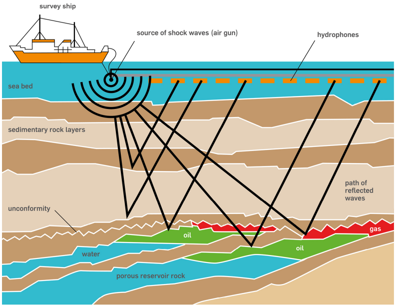
Case study: Seigen

Seismology through code generation[1]

- Seismic model using elastic wave equation
- Implemented purely on top of Firedrake (UFL)
- Explore end-to-end optimisation through symbolic computation

As used in energy exploration

- Full Waveform Inversion (FWI)
- Traditionally finite difference (FD)
- Explore use of unstructured meshes

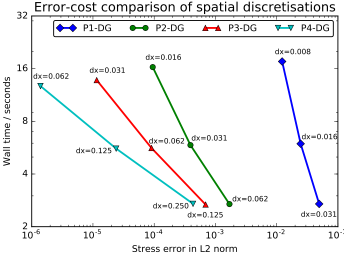


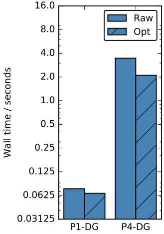
[1] C. T. Jacobs, M. Lange, F. Luporini, and G. J. Gorman. Application of code generation to high-order seismic modelling with the discontinuous galerkin finite element method. Under Preparation

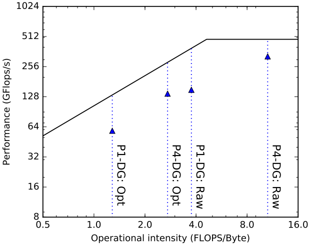
Case study: Seigen

Seismology through code generation[1]

- Discontinuous finite element (DG-FEM) with implicit and explicit solves
- 4th order time-stepping and up to 4th order spatial discretisation







[1] C. T. Jacobs, M. Lange, F. Luporini, and G. J. Gorman. Application of code generation to high-order seismic modelling with the discontinuous galerkin finite element method. Under Preparation

Conclusion

Threading: Yes, no, maybe . . .

- Performance optimsiation is usually more complicated than `#pragma openmp for`

What matters is end-to-end optimisation

- Consider model, numerics, data optimisation and compiler tricks
- Optimisation needs to fit parallelisation, needs to fit hardware!

Separation of concerns through abstraction layering

- Enables end-to-end optimisation
- Allows expertise from all relevant fields
- Requires run-time decisions[1]

[1] J. Brown, M. Knepley, and B. Smith. Run-time extensibility and librarization of simulation software. *IEEE Computing in Science and Engineering*, 2015

Thank You

Don't miss:

- Poster session - Seigen: Seismic modelling through code generation
- Friday, 4.50pm - F. Luporini: Generating High Performance Finite Element Kernels Using Optimality Criteria

www.firedrakeproject.org

http://www.opesci.org