Firedrake: a multilevel domain specific language approach to unstructured mesh stencil computations

Lawrence Mitchell, Gheorghe-Teodor Bercea, David Ham, Paul Kelly, Nicolas Loriant, Fabio Luporini, Florian Rathgeber

Departments of Mathematics and Computing, Imperial College London

21 February 2014

Imperial College London

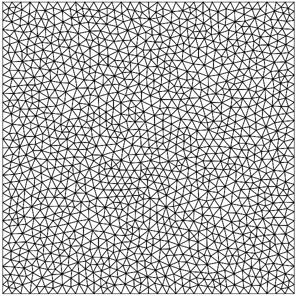
What are we interested in?

- (Predominantly) finite element simulations
  - primary application areas in geophysical fluids (ocean and atmosphere)
  - simulations on unstructured and semi-structured meshes
- Providing high-level interfaces for users, with performance

  - the moon, on a stick

How does FE fit a stencils session?

- Numerics tells us the elementary operation we apply everywhere in the mesh (a "kernel")
- Mesh topology gives us the "stencil" pattern
- Our job: efficiently apply the kernel over the whole mesh



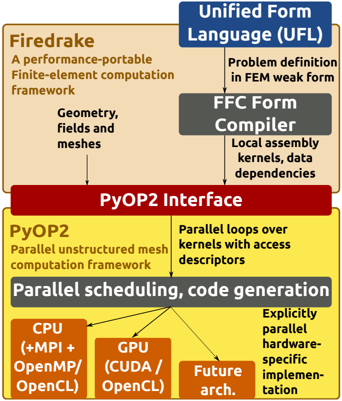
Imperial College London

Express what, not how

- User code should make as few decisions about implementation as possible
- FE discretisations expressed symbolically using the Unified Form Language
    - developed in the FEniCS project (`http://www.fenicsproject.org`)
    - symbolic representation compiled to a C kernel
- Data to feed to kernel (and interface to solvers) provided by Firedrake (`http://www.firedrakeproject.org`)
- Execution of kernel over entire domain expressed as parallel loop with access descriptors
    - uses PyOP2 unstructured mesh library (`http://github.com/OP2/PyOP2`)
    - implementation of loop taken out of user hands

Firedrake

- High level finite element computations
  - `http://www.firedrakeproject.org`

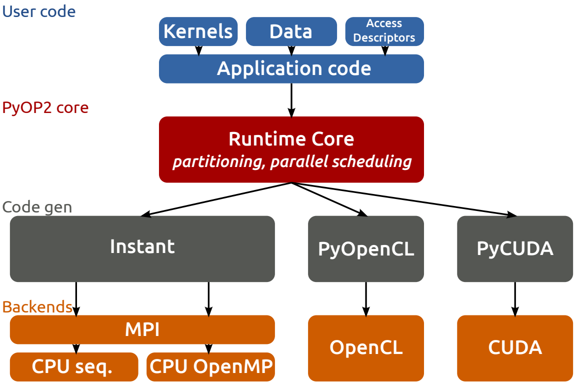


An example

```
from firedrake import *
mesh = UnitSquareMesh(32, 32)
BDM = FunctionSpace(mesh, "BDM", 3)
DG = FunctionSpace(mesh, "DG", 2)
W = BDM * DG
sigma, u = TrialFunctions(W)
tau, v = TestFunctions(W)
f = Function(DG).interpolate(Expression(
    "10*exp(-(pow(x[0] - 0.5, 2) + pow(x[1] - 0.5, 2)) / 0.02)"))
a = (dot(sigma, tau) + div(tau)*u + div(sigma)*v)*dx
L = - f*v*dx
bc0 = DirichletBC(W.sub(0), Expression(("0.0", "-sin(5*x[0])")), 1)
bc1 = DirichletBC(W.sub(0), Expression(("0.0", "sin(5*x[0])")), 2)
w = Function(W)
solve(a == L, w, bcs=[bc0, bc1])
sigma, u = w.split()
```

PyOP2

- A python library for unstructured mesh computations
  - `http://github.com/OP2/PyOP2`



PyOP2 data model

- Data types

Set	e.g. cells, degrees of freedom (dofs)
Dat	data defined on a `Set` (one entry per set element)
Map	a mapping between two sets (e.g. cells to dofs)
Global	global data (one entry)
Kernel	a piece of code to execute over the mesh

- access descriptors
  - READ, RW, WRITE, INC
- iteration construct

`par_loop`	execute a Kernel over every element in a `Set`

Example

```
par_loop(kernel, elements,
         element_data(READ),  # direct access
         node_data(WRITE, elem_node), # indirect
         count(INC))
```

- executes kernel for each ele in elements

  first argument data corresponding to ele (read-only)

  second argument the nodes of this ele (written)

  third argument a global counter (incremented)
- runtime knows it has to care about data dependencies for
  - write to node_data
  - increment into elem_count

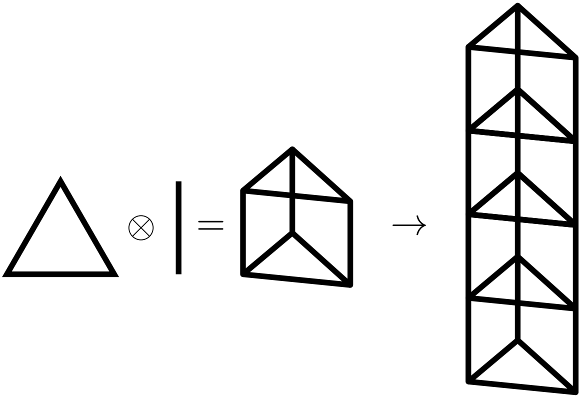
Synthesis, not analysis

- Access descriptors on parallel loops mean:
  - code generation requires synthesis, not analysis
  - determination of when halo exchanges need to occur is automatic
  - colouring for shared memory parallelisation can be computed automatically

Semi-structured meshes

- Many application areas have a "short" direction
  - ocean and atmosphere
  - thin shells
- Numerics dictate we should do something different in short direction
- Use semi-structured meshes
  - unstructured in "long" directions, structured in short
  - can we exploit this structure?

A picture of triangles



Admits a fast implementation

- Exploit structure in mesh to amortize indirect lookups
  - arrange for iteration over short direction to be innermost loop
  - pay one indirect lookup per mesh column
  - walk up column directly

Imperial College London

A bandwidth bound test case

- Walk over mesh, read from vertices and cells, sum into global

```c
void kernel(double *a, double *x[], double *y[]) {
    const double area = fabs(x[0][0]*(x[2][1]-x[4][1])
                             + x[2][0]*(x[4][1]-x[0][1])
                             + x[4][0]*(x[0][1]-x[2][1]));
    *a += area * 0.5 * y[0][0];
}
```

- Can we sustain an appreciable fraction of memory bandwidth?

Measuring throughput

- "Effective" data volume
  - assume every piece of data is touched exactly once (in perfect order)
  - don't count data movement for indirection maps
  - effectively, just count the volume of degrees of freedom touched
- "Valuable" bandwidth
  - effective data volume per second
- Actual memory bandwidth will be higher (reading indirection maps)
  - but this is not "useful"

Benchmark setup

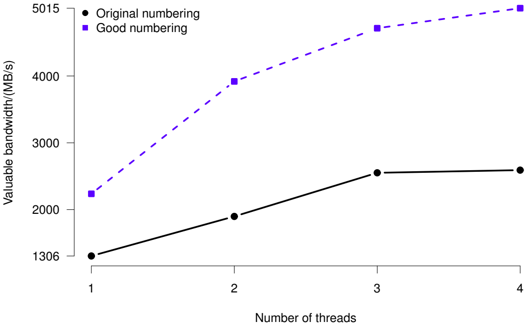
- 2D unstructured mesh: 806110 cells, 403811 vertices.
  - 2D coordinate field located at vertices (implicit 3rd coordinate)
  - scalar field stored at cell centres
- Run with increasing number of extruded cell layers (n_{layer})
  - data volume ($806110 * n_{layer}$) + $403811 * 2$ ($n_{layer} + 1$) doubles
  - 1 layer: 18.4MB
  - 200 layers: 2468MB
- Execute kernel over mesh 100 times

Single node

- Intel Sandybridge 4 cores (2 way hyperthreading)
  - 32kB L1 cache (per core)
  - 256 kB L2 cache (per core)
  - 8 MB L3 cache (shared)
- Measured stream bandwidth (8 threads)
  - 11341 MB/s

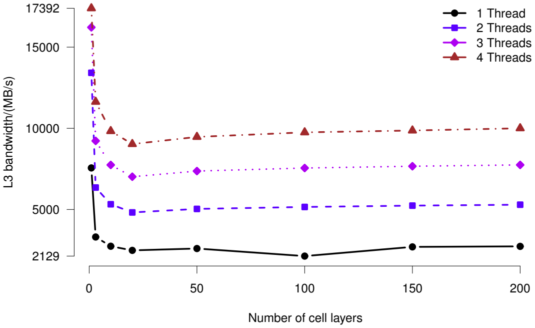
Effect of good base numbering

- Being completely unstructured hurts a lot
- Compare default (mesh generator) numbering with renumbered mesh using 2D space filling curve



Adding layers amortizes indirection cost

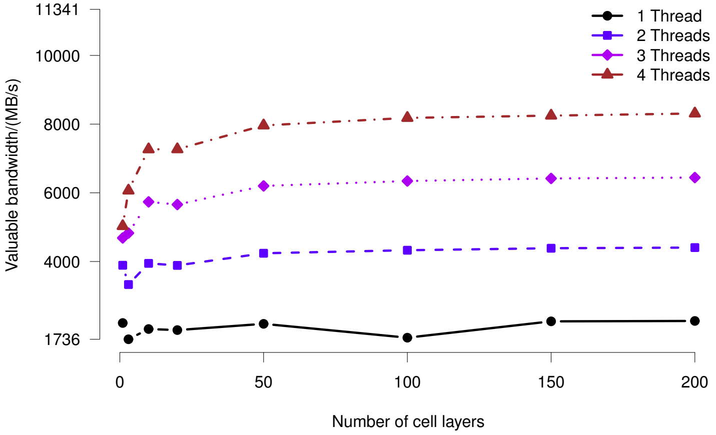
- L3 cache bandwidth
  - low layer numbers hit the L3 more often (indirection lookups)



- What about actual throughput though?

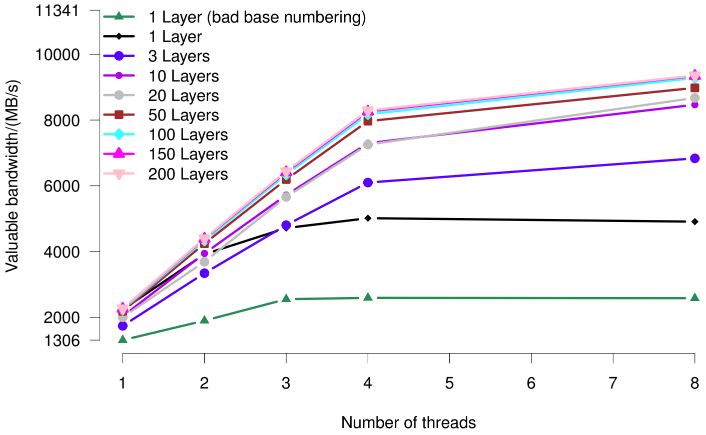
Valuable bandwidth

- Above ~20 layers, indirection cost "hidden"



More threads

- Hyperthreading gives some further gains (82% stream bandwidth)



Imperial College London

Possible to be unstructured and fast

- A good numbering gets you a reasonable way there
- If there is structure in your problem, use it!
- High level abstractions need not kill performance

Thanks

- Institutions
  - Imperial College London
  - Grantham Institute for climate change
- Funding
  - NERC (NE/K008951/1, NE/K006789/1, NE/G523512/1)
  - EPSRC (EP/L000407/1, EP/K008730/1, EP/I00677X/1)

Imperial College London