Firedrake: the architecture of a compiler that automates the finite element method

Paul Kelly
Group Leader, Software Performance Optimisation
Department of Computing
Imperial College London

Joint work with David Ham (Imperial Maths), Lawrence Mitchell (Imperial Computing)
Fabio Luporini (Imperial Earth Science Engineering), Florian Rathgeber (now with Google), Doru Bercea (now with IBM Research), Michael Lange (now with ECMWF), Andrew McRae (now at University of Oxford), Graham Markall (now at Embecosm Ltd), Tianjiao Sun (now at Cerebras), Thomas Gibson (Imperial Maths)
And many others....

Imperial College London

This talk

- Three different potential audiences:
  - Programming language design and implementation
  - Numerical methods for PDEs
  - High-performance computing

- What *is* Firedrake?
- What is it used for? By whom?
- What does its DSL actually look like?
- What is its domain of applicability?

- How is its compiler designed?
- Does it generate good code?
- Does it automate interesting optimisations that would be hard to do by hand?

- What are the open research challenges?
- What would we do differently?
- What is the opportunity to change the world?

Firedrake

Documentation Download Team Citing Publications Events Funding Contact GitHub Jenkins

Firedrake is an automated system for the solution of partial differential equations using the finite element method (FEM). Firedrake uses sophisticated code generation to provide mathematicians, scientists, and engineers with a very high productivity way to create sophisticated high performance simulations.

Features:

- Expressive specification of any PDE using the Unified Form Language from the FEniCS Project.
- Sophisticated, programmable solvers through seamless coupling with PETSc.
- Triangular, quadrilateral, and tetrahedral unstructured meshes.
- Layered meshes of triangular wedges or hexahedra.
- Vast range of finite element spaces.
- Sophisticated automatic optimisation, including sum factorisation for high order elements, and vectorisation.
- Geometric multigrid.
- Customisable operator preconditioners.
- Support for static condensation, hybridisation, and HDG methods.

Latest commits to the Firedrake master branch on Github

Merge pull request #1520 from firedrakeproject/wence/feature/assemble-diagonal
Lawrence Mitchell authored at 22/10/2019, 09:14:34

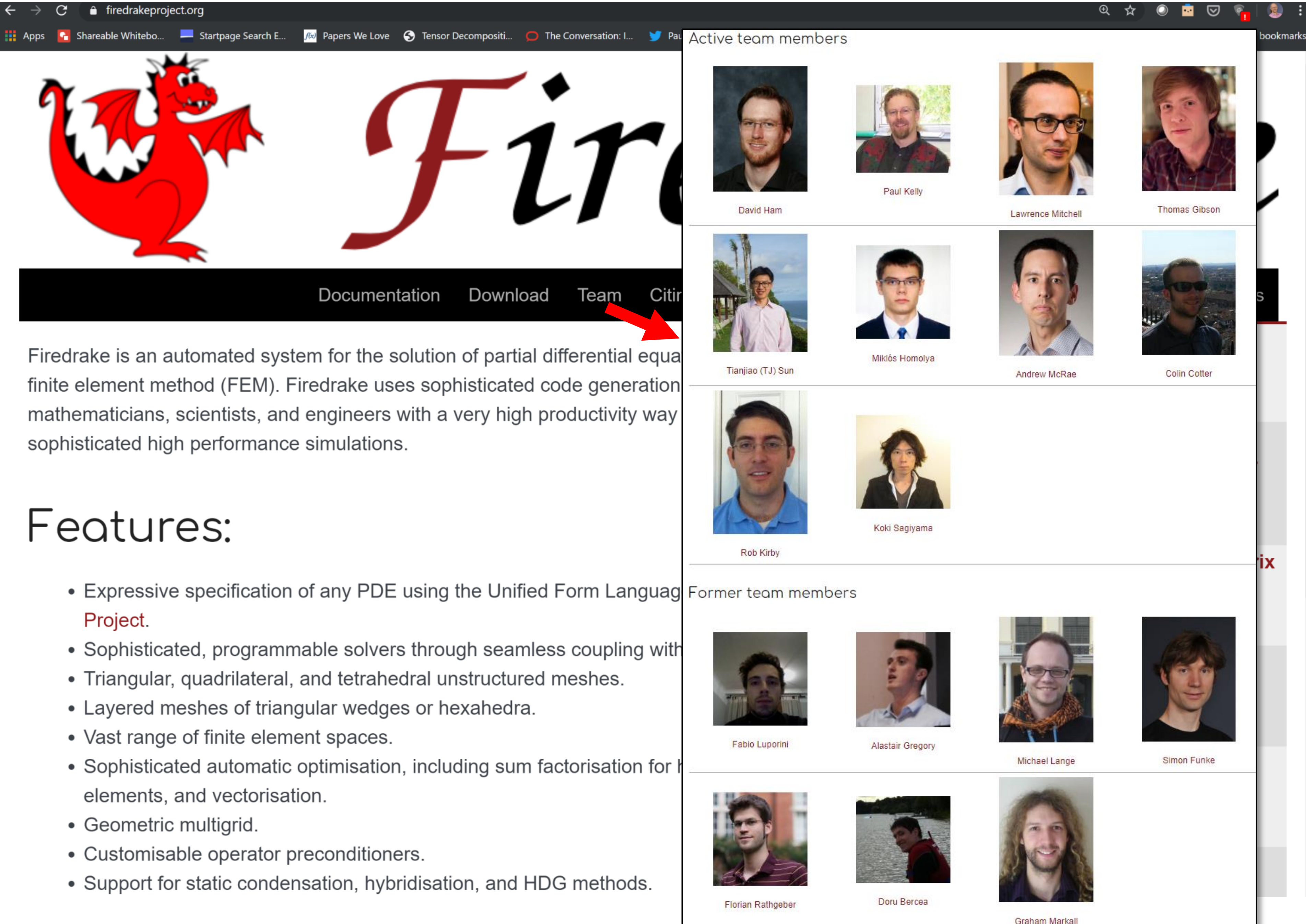
tests: Check that getting diagonal of matrix works
Lawrence Mitchell authored at 21/10/2019, 13:04:04

matfree: Add getDiagonal method to implicit matrices
Lawrence Mitchell authored at 18/10/2019, 10:19:48

assemble: Add option to assemble diagonal of 2-form into Dat
Lawrence Mitchell authored at 18/10/2019, 10:08:37

Merge pull request #1509 from firedrakeproject/wence/patch-c-wrapper

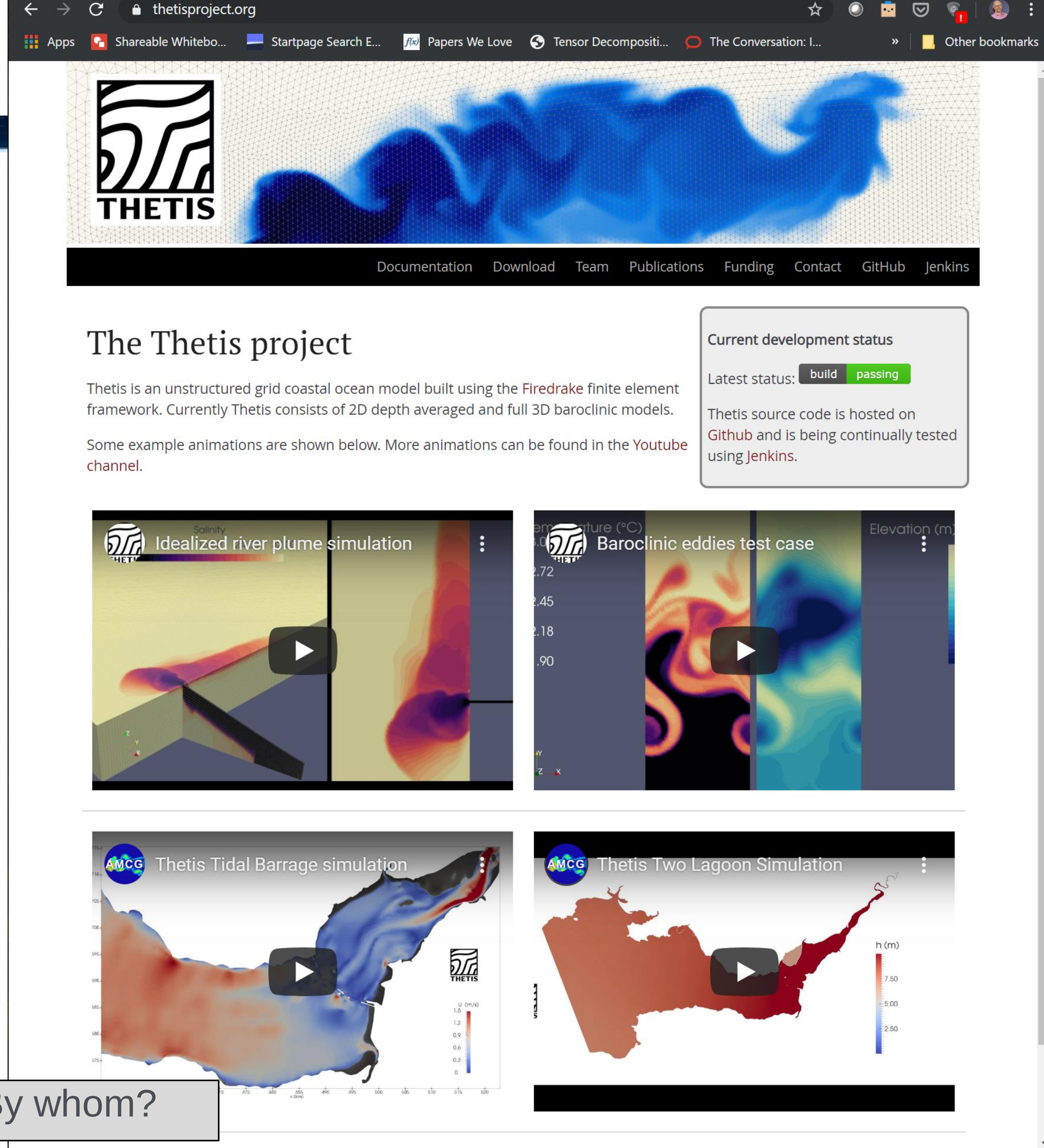
What *is* Firedrake?



What *is* Firedrake?

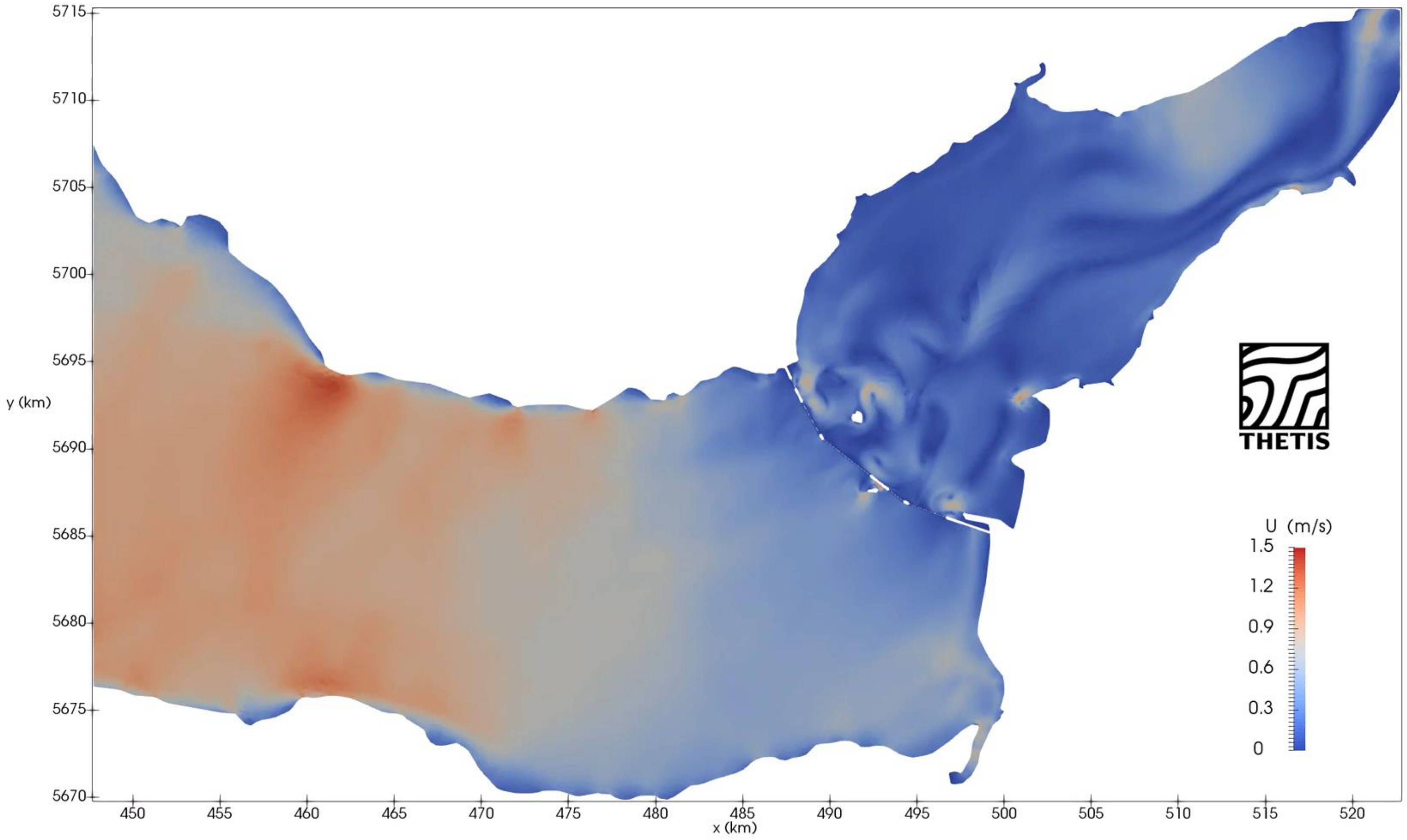
Imperial College London

- Firedrake is used in:
  - **Thetis**: unstructured grid coastal modelling framework



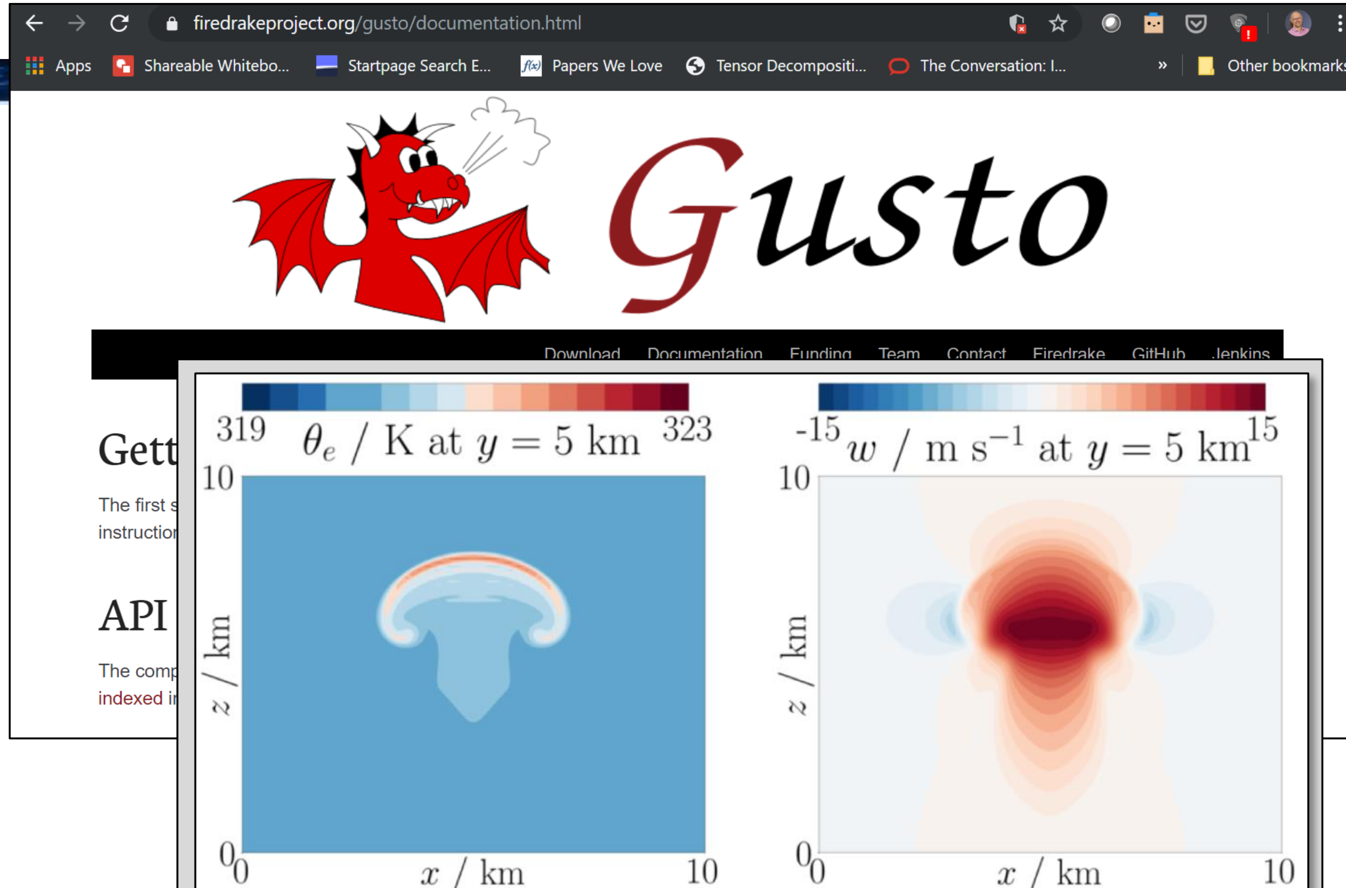
- What is it used for? By whom?

Imperial College London



- Tidal barrage simulation using Thetis (https://thetisproject.org/)
  - What is it used for? By whom?

- Firedrake is used in:
  - **Gusto**: atmospheric modelling framework being used to prototype the next generation of weather and climate simulations for the UK Met Office

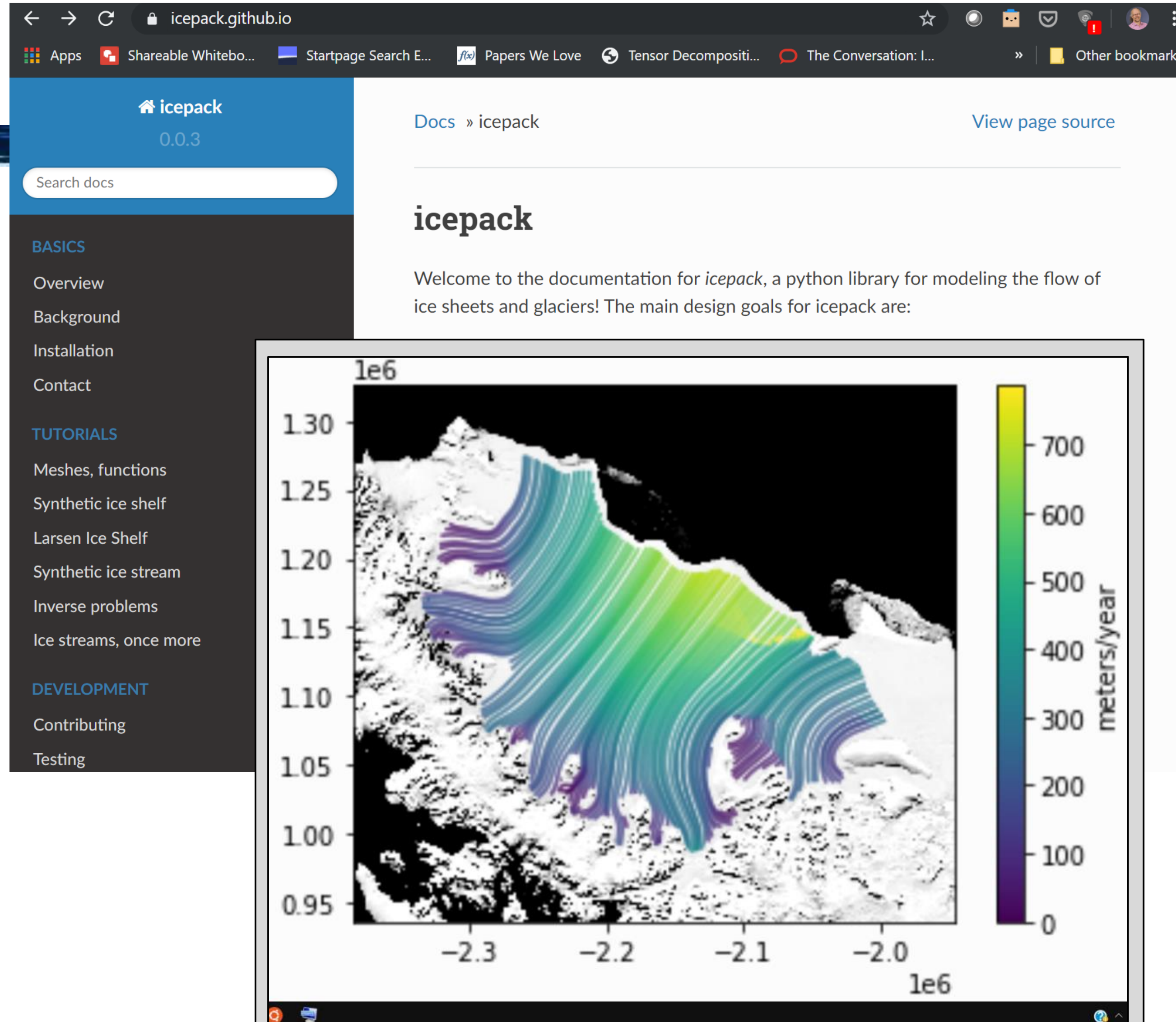


Three-dimensional simulation of a thermal rising through a saturated atmosphere. From *A Compatible Finite Element Discretisation for the Moist Compressible Euler Equations (Bendall et al, https://arxiv.org/pdf/1910.01857.pdf)*

- What is it used for? By whom?

Imperial College London

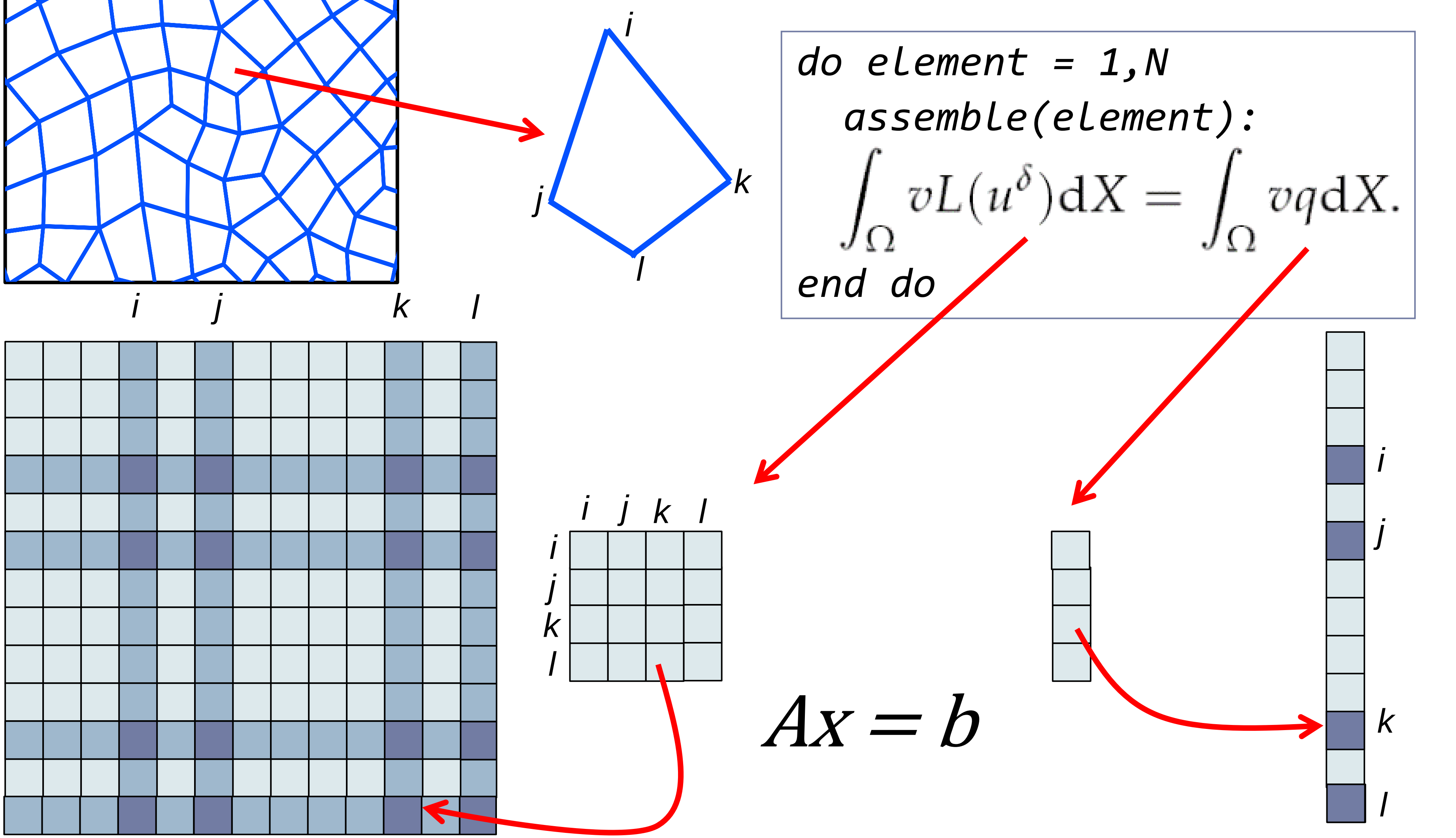
- Firedrake is used in:
  - **Icepack**: a framework for modeling the flow of glaciers and ice sheets, developed at the Polar Science Center at the University of Washington



Larsen ice shelf model, from the Icepack tutorial by Daniel Shapero (https://icepack.github.io/icepack.demo.02-larsen-ice-shelf.html)

- What is it used for? By whom?

The finite element method in outline



- Key data structures: Mesh, dense local assembly matrices, sparse global system matrix, and RHS vector

Imperial College London

Multilayered abstractions for FE

- Local assembly:
  - Computes local assembly matrix
  - Using:
    - The (weak form of the) PDE
    - The discretisation
  - Key operation is evaluation of expressions over basis function representation of the element
- Mesh traversal:
  - *PyOP2*
  - *Loops over the mesh*
  - *Key is orchestration of data movement*
- Solver:
  - Interfaces to standard solvers through PetSc

Example: Burgers equation

- We start with the PDE: (see https://www.firedrakeproject.org/demos/burgers.py.html)

The Burgers equation is a non-linear equation for the advection and diffusion of momentum. Here we choose to write the Burgers equation in two dimensions to demonstrate the use of vector function spaces:

$$\frac{\partial u}{\partial t} + (u \cdot \nabla) u - \nu \nabla^2 u = 0$$
$$(n \cdot \nabla) u = 0 \text{ on } \Gamma$$

where Γ is the domain boundary and ν is a constant scalar viscosity. The solution u is sought in some suitable vector-valued function space V. We take the inner product with an arbitrary test function $v \in V$ and integrate the viscosity term by parts:

$$\int_\Omega \frac{\partial u}{\partial t} \cdot v + ((u \cdot \nabla) u) \cdot v + \nu \nabla u \cdot \nabla v \, \mathrm{d}x = 0.$$

The boundary condition has been used to discard the surface integral. Next, we need to discretise in time. For simplicity and stability we elect to use a backward Euler discretisation:

$$\int_\Omega \frac{u^{n+1} - u^n}{dt} \cdot v + ((u^{n+1} \cdot \nabla) u^{n+1}) \cdot v + \nu \nabla u^{n+1} \cdot \nabla v \, \mathrm{d}x = 0.$$

- From the weak form of the PDE, we derive an equation to solve, that determines the state at each timestep in terms of the previous timestep

Example: Burgers equation

$$\int_\Omega \frac{u^{n+1} - u^n}{dt} \cdot v + ((u^{n+1} \cdot \nabla)u^{n+1}) \cdot v + \nu \nabla u^{n+1} \cdot \nabla v \, \mathrm{d}x = 0.$$

- From the weak form of the PDE, we derive an equation to solve, that determines the state at each timestep in terms of the previous timestep

Example: Burgers equation

$$\int_\Omega \frac{u^{n+1} - u^n}{dt} \cdot v + ((u^{n+1} \cdot \nabla) u^{n+1}) \cdot v + \nu \nabla u^{n+1} \cdot \nabla v \, \mathrm{d}x = 0.$$

- From the weak form of the PDE, we derive an equation to solve, that determines the state at each timestep in terms of the previous timestep
- Transcribe into Python – u is u^{n+1}, u_ is u^n :

```
F = (inner((u - u_)/timestep, v)
    + inner(dot(u,nabla_grad(u)), v) + nu*inner(grad(u), grad(v)))*dx
```

Burgers equation

```
from firedrake import *
n = 50
mesh = UnitSquareMesh(n, n)

# We choose degree 2 continuous Lagrange polynomials. We also need a
# piecewise linear space for output purposes::

V = VectorFunctionSpace(mesh, "CG", 2)
V_out = VectorFunctionSpace(mesh, "CG", 1)

# We also need solution functions for the current and the next timestep::

u_ = Function(V, name="Velocity")
u = Function(V, name="VelocityNext")

v = TestFunction(V)

# We supply an initial condition::

x = SpatialCoordinate(mesh)
ic = project(as_vector([sin(pi*x[0]), 0]), V)

# Start with current value of u set to the initial condition, and use the
# initial condition as our starting guess for the next value of u::

u_.assign(ic)
u.assign(ic)

# :math:`\nu` is set to a (fairly arbitrary) small constant value::

nu = 0.0001

timestep = 1.0/n

# Define the residual of the equation::

F = (inner((u - u_)/timestep, v)
     + inner(dot(u,nabla_grad(u)), v) + nu*inner(grad(u), grad(v)))*dx

outfile = File("burgers.pvd")

outfile.write(project(u, V_out, name="Velocity"))

# Finally, we loop over the timesteps solving the equation each time::

t = 0.0
end = 0.5
while (t <= end):
    solve(F == 0, u)
    u_.assign(u)
    t += timestep
    outfile.write(project(u, V_out, name="Velocity"))
```

- Firedrake implements the Unified Form Language (UFL)
- Embedded in Python

$$\int_\Omega \frac{u^{n+1}-u^n}{dt}\cdot v + ((u^{n+1}\cdot\nabla)u^{n+1})\cdot v + \nu\nabla u^{n+1}\cdot\nabla v \,\mathrm{d}x = 0.$$

- From the weak form of the PDE, we derive an equation to solve, that determines the state at each timestep in terms of the previous timestep
- Transcribe into Python – u is u^{n+1}, u_ is u^n :

```
F = (inner((u - u_)/timestep, v)
  + inner(dot(u,nabla_grad(u)), v) + nu*inner(grad(u), grad(v)))*dx
```

- UFL is also the DSL of the FEniCS project
- What does its DSL actually look like?

```
from firedrake import *
n = 50
mesh = UnitSquareMesh(n, n)

# We choose degree 2 continuous Lagrange polynomials.
# piecewise linear space for output purposes::

V = VectorFunctionSpace(mesh, "CG", 2)
V_out = VectorFunctionSpace(mesh, "CG", 1)

# We also need solution functions for the current and

u_ = Function(V, name="Velocity")
u = Function(V, name="VelocityNext")

v = TestFunction(V)

# We supply an initial condition::

x = SpatialCoordinate(mesh)
ic = project(as_vector([sin(pi*x[0]), 0]), V)

# Start with current value of u set to
# initial condition as our starting gue

u_.assign(ic)
u.assign(ic)

# :math:`\nu` is set to a (fairly arbit

nu = 0.000

timestep =

# Define t

F = (inner
    + inn

outfile =

outfile.wr

# Finally,

t = 0.0
end = 0.5
while (t <
    solve(F == 0, u)
    u_.assign(u)
    t += timestep
    outfile.write(project(u, V_out, name="Velocity"))
```

```
mesh = UnitSquareMesh(n, n)
```

```
V = VectorFunctionSpace(mesh, "CG", 2)
V_out = VectorFunctionSpace(mesh, "CG", 1)
```

```
u_ = Function(V, name="Velocity")
u = Function(V, name="VelocityNext")
```

*# set up initial conditions for u and u_*

```
# Define the residual of the equation::

F = (inner((u - u_)/timestep, v)
    + inner(dot(u,nabla_grad(u)), v) + nu*inner(grad(u), grad(v)))*dx
```

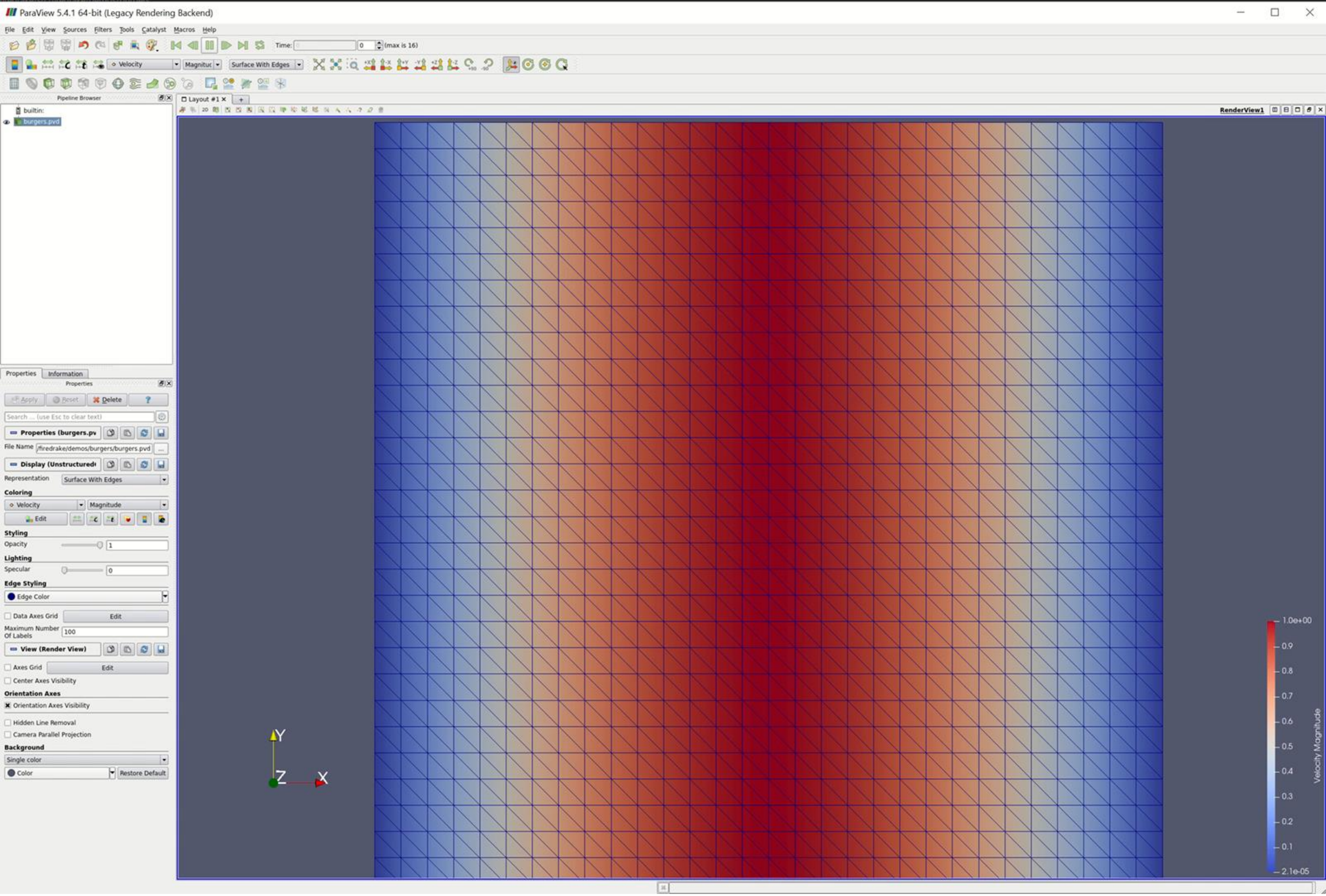
```
t = 0.0
end = 0.5
while (t <= end):
    solve(F == 0, u)
    u_.assign(u)
    t += timestep
    outfile.write(project(u, V_out, name="Velocity"))
```

ation

What does its DSL actually look like?

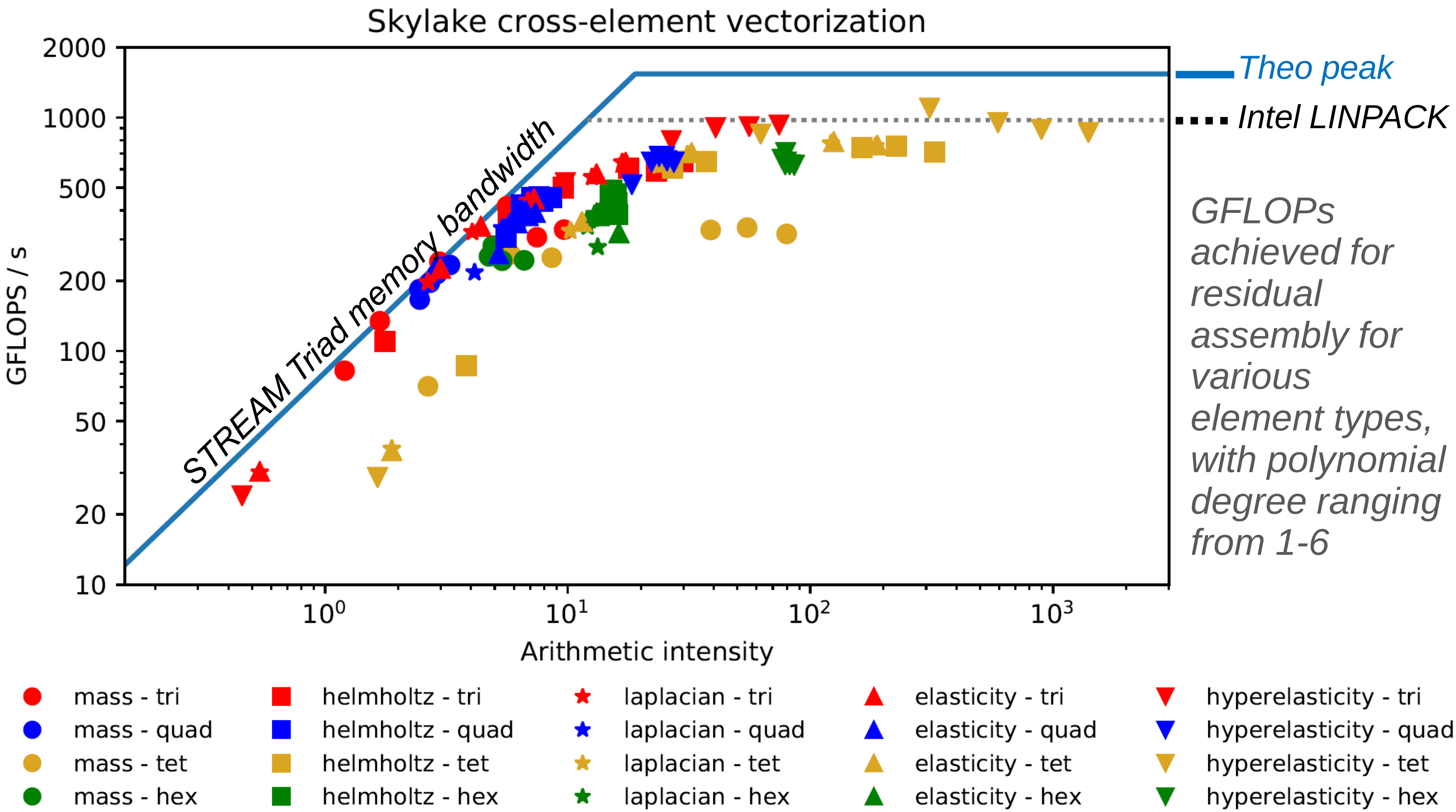
```c
#include <math.h>
#include <petsc.h>

void wrap_form00_cell_integral_otherwise(int const start, int const end, Mat const mat0, double const *__restrict__ dat1, double const *__restrict__ dat0, int const *__restrict__ map0, int const *__restrict__ map1)
{
  double form_t0...t16;
  double const form_t17[7] = { ... };
  double const form_t18[7 * 6] = { ... };
  double const form_t19[7 * 6] = { ... };
  double form_t2;
  double const form_t20[7 * 6] = { ... };
  double form_t21...t37;
  double form_t38[6];
  double form_t39[6];
  double form_t4;
  double form_t40...t45;
  double form_t5...t9;
  double t0[6 * 2];
  double t1[3 * 2];
  double t2[6 * 2 * 6 * 2];

  for (int n = start; n <= -1 + end; ++n)
  {
    for (int i4 = 0; i4 <= 5; ++i4)
      for (int i5 = 0; i5 <= 1; ++i5)
        for (int i6 = 0; i6 <= 5; ++i6)
          for (int i7 = 0; i7 <= 1; ++i7)
            t2[24 * i4 + 12 * i5 + 2 * i6 + i7] = 0.0;
    for (int i2 = 0; i2 <= 2; ++i2)
      for (int i3 = 0; i3 <= 1; ++i3)
        t1[2 * i2 + i3] = dat1[2 * map1[3 * n + i2] + i3];
    for (int i0 = 0; i0 <= 5; ++i0)
      for (int i1 = 0; i1 <= 1; ++i1)
        t0[2 * i0 + i1] = dat0[2 * map0[6 * n + i0] + i1];
    form_t0 = -1.0 * t1[1];
    form_t1 = form_t0 + t1[3];
    form_t2 = -1.0 * t1[0];
    form_t3 = form_t2 + t1[2];
    form_t4 = form_t0 + t1[5];
    form_t5 = form_t2 + t1[4];
    form_t6 = form_t3 * form_t4 + -1.0 * form_t5 * form_t1;
    form_t7 = 1.0 / form_t6;
    form_t8 = form_t7 * -1.0 * form_t1;
    form_t9 = form_t4 * form_t7;
    form_t10 = form_t3 * form_t7;
    form_t11 = form_t7 * -1.0 * form_t5;
    form_t12 = 0.0001 * (form_t8 * form_t9 + form_t10 * form_t11);
    form_t13 = 0.0001 * (form_t8 * form_t8 + form_t10 * form_t10);
    form_t14 = 0.0001 * (form_t9 * form_t9 + form_t11 * form_t11);
    form_t15 = 0.0001 * (form_t9 * form_t8 + form_t11 * form_t10);
    form_t16 = fabs(form_t6);
    for (int form_ip = 0; form_ip <= 6; ++form_ip)
    {
      form_t26 = 0.0; form_t25 = 0.0; form_t24 = 0.0; form_t23 = 0.0; form_t22 = 0.0; form_t21 = 0.0;
      for (int form_i = 0; form_i <= 5; ++form_i)
      {
        form_t21 = form_t21 + form_t20[6 * form_ip + form_i] * t0[1 + 2 * form_i];
        form_t22 = form_t22 + form_t19[6 * form_ip + form_i] * t0[1 + 2 * form_i];
        form_t23 = form_t23 + form_t20[6 * form_ip + form_i] * t0[2 * form_i];
        form_t24 = form_t24 + form_t19[6 * form_ip + form_i] * t0[2 * form_i];
        form_t25 = form_t25 + form_t18[6 * form_ip + form_i] * t0[1 + 2 * form_i];
        form_t26 = form_t26 + form_t18[6 * form_ip + form_i] * t0[2 * form_i];
      }
      form_t27 = form_t17[form_ip] * form_t16;
      form_t28 = form_t27 * form_t15;
      form_t29 = form_t27 * form_t14;
      form_t30 = form_t27 * (form_t26 * form_t9 + form_t25 * form_t11);
      form_t31 = form_t27 * form_t13;
      form_t32 = form_t27 * form_t12;
      form_t33 = form_t27 * (form_t26 * form_t8 + form_t25 * form_t10);
      form_t34 = form_t27 * (form_t11 * form_t24 + form_t10 * form_t23);
      form_t35 = form_t27 * (form_t9 * form_t22 + form_t8 * form_t21);
      form_t36 = form_t27 * (50.0 + form_t9 * form_t24 + form_t8 * form_t23);
      form_t37 = form_t27 * (50.0 + form_t11 * form_t22 + form_t10 * form_t21);
      for (int form_k0 = 0; form_k0 <= 5; ++form_k0)
      {
        form_t38[form_k0] = form_t18[6 * form_ip + form_k0] * form_t37;
        form_t39[form_k0] = form_t18[6 * form_ip + form_k0] * form_t36;
      }
      for (int form_j0 = 0; form_j0 <= 5; ++form_j0)
      {
        form_t40 = form_t18[6 * form_ip + form_j0] * form_t35;
        form_t41 = form_t18[6 * form_ip + form_j0] * form_t34;
        form_t42 = form_t20[6 * form_ip + form_j0] * form_t31 + form_t18[6 * form_ip + form_j0] * form_t33 + form_t19[6 * form_ip + form_j0] * form_t32;
        form_t43 = form_t20[6 * form_ip + form_j0] * form_t28 + form_t18[6 * form_ip + form_j0] * form_t30 + form_t19[6 * form_ip + form_j0] * form_t29;
        for (int form_k0_0 = 0; form_k0_0 <= 5; ++form_k0_0)
        {
          form_t44 = form_t43 * form_t19[6 * form_ip + form_k0_0];
          form_t45 = form_t42 * form_t20[6 * form_ip + form_k0_0];
          t2[24 * form_j0 + 2 * form_k0_0] = t2[24 * form_j0 + 2 * form_k0_0] + form_t45 + form_t18[6 * form_ip + form_j0] * form_t39[form_k0_0] + form_t44;
          t2[13 + 24 * form_j0 + 2 * form_k0_0] = t2[13 + 24 * form_j0 + 2 * form_k0_0] + form_t45 + form_t18[6 * form_ip + form_j0] * form_t38[form_k0_0] + form_t44;
          t2[1 + 24 * form_j0 + 2 * form_k0_0] = t2[1 + 24 * form_j0 + 2 * form_k0_0] + form_t18[6 * form_ip + form_k0_0] * form_t41;
          t2[12 + 24 * form_j0 + 2 * form_k0_0] = t2[12 + 24 * form_j0 + 2 * form_k0_0] + form_t18[6 * form_ip + form_k0_0] * form_t40;
        }
      }
    }
    MatSetValuesBlockedLocal(mat0, 6, &(map0[6 * n]), 6, &(map0[6 * n]), &(t2[0]), ADD_VALUES);
  }
}
```

- Generated code to assemble the resulting linear system matrix
- Executed at each triangle in the mesh
- Accesses degrees of freedom shared with neighbour triangles through indirection map


ParaView 5.4.1 64-bit (Legacy Rendering Backend)
File Edit View Sources Filters Tools Catalyst Macros Help
Time: 0 (max is 16)
Velocity
Magnitude
Surface With Edges
Pipeline Browser
builtin:
burgers.pvd
Layout #1
RenderView1
Properties Information
Apply Reset Delete
Search ... (use Esc to clear text)
Properties (burgers.pv
File Name /firedrake/demos/burgers/burgers.pvd
Display (Unstructured
Representation Surface With Edges
Coloring
Velocity Magnitude
Edit
Styling
Opacity 1
Lighting
Specular 0
Edge Styling
Edge Color
Data Axes Grid Edit
Maximum Number Of Labels 100
View (Render View)
Axes Grid Edit
Center Axes Visibility
Orientation Axes
Orientation Axes Visibility
Hidden Line Removal
Camera Parallel Projection
Background
Single color
Color Restore Default
Y
Z
X
1.0e+00
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
2.1e-05
Velocity Magnitude

Firedrake: single-node AVX512 performance

- Does it generate good code?

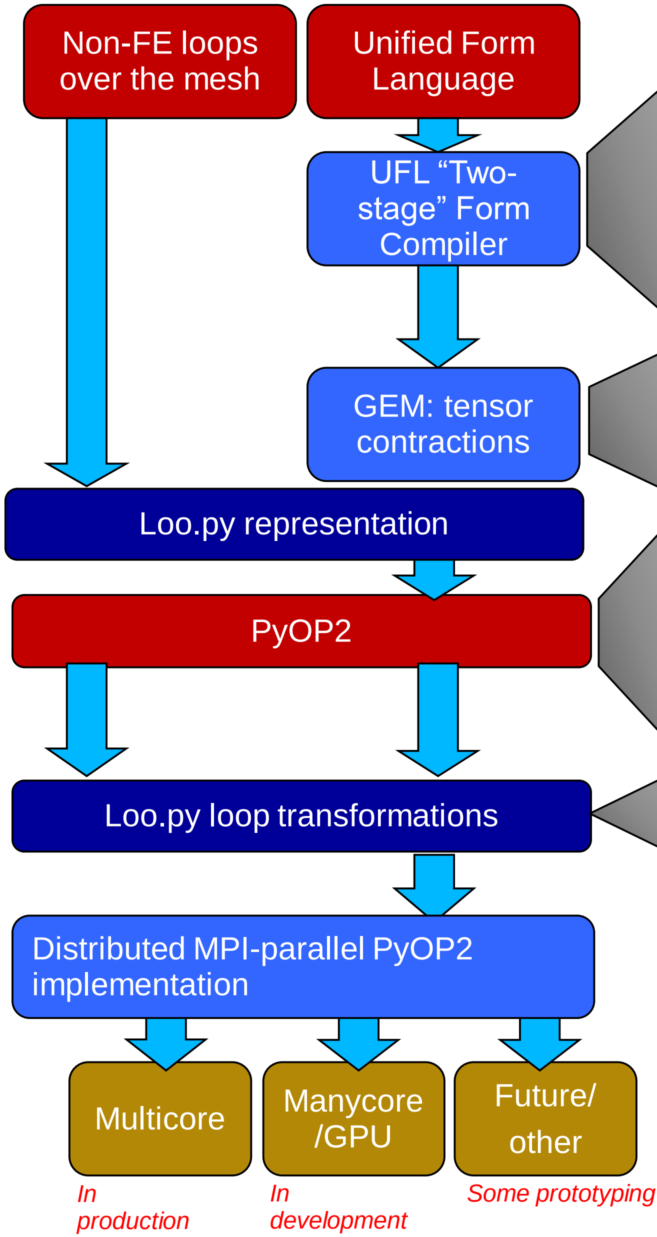


GFLOPs achieved for residual assembly for various element types, with polynomial degree ranging from 1-6

[Skylake Xeon Gold 6130 (on all 16 cores, 2.1GHz, turboboost off, Stream: 36.6GB/s, GCC7.3 –march=native)]

A study of vectorization for matrix-free finite element methods, Tianjiao Sun et al
https://arxiv.org/abs/1903.08243

Firedrake: compiler architecture



- **UFL** specifies the (weak form of the) partial differential equation and how it is to be discretised
- Compiler generates PyOP2 kernels and access descriptors

- **GEM:** abstract representation supports efficient flop-reduction optimisations

- **PyOP2**: stencil DSL for *unstructured-mesh*
- Explicit *access descriptors* characterise access footprint of kernels

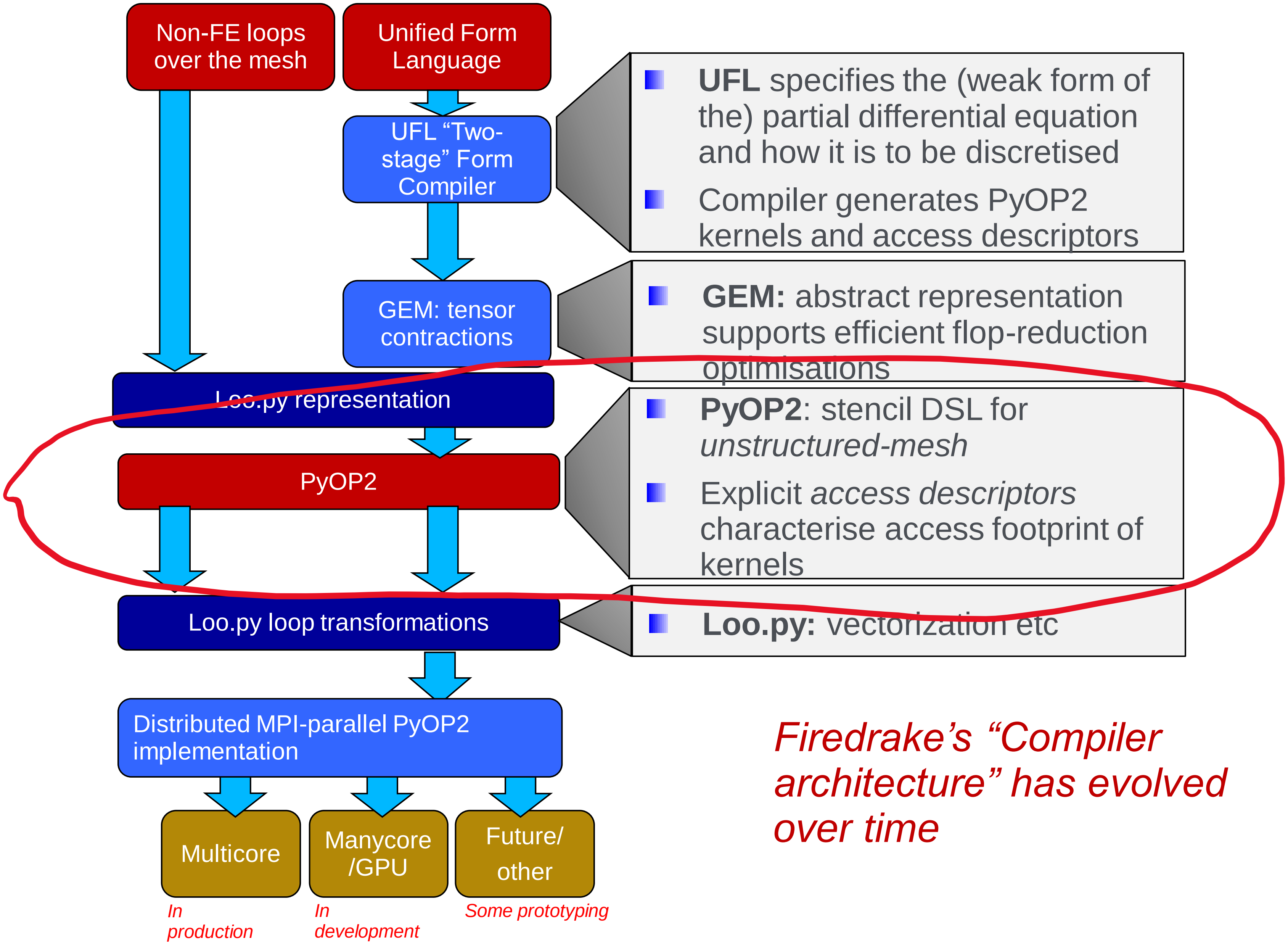
- **Loo.py:** vectorization etc

- **Sequence of intermediate representations**
- **100% Python, runtime code generation, code-caching**

Rathgeber, Ham, Mitchell et al, ACM TOMS 2016, Tianjiao Sun et al https://arxiv.org/pdf/1903.08243.pdf

Firedrake: a finite-element framework

- Automates the finite element method for solving PDEs
- Alternative implementation of FEniCS language, 100% Python using runtime code generation



Firedrake's "Compiler architecture" has evolved over time

Rathgeber, Ham, Mitchell et al, ACM TOMS 2016, Tianjiao Sun et al https://arxiv.org/pdf/1903.08243.pdf

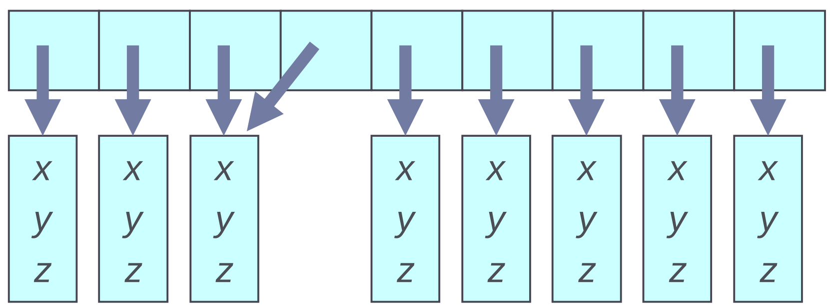
Imperial College London

Easy parallelism

Example:

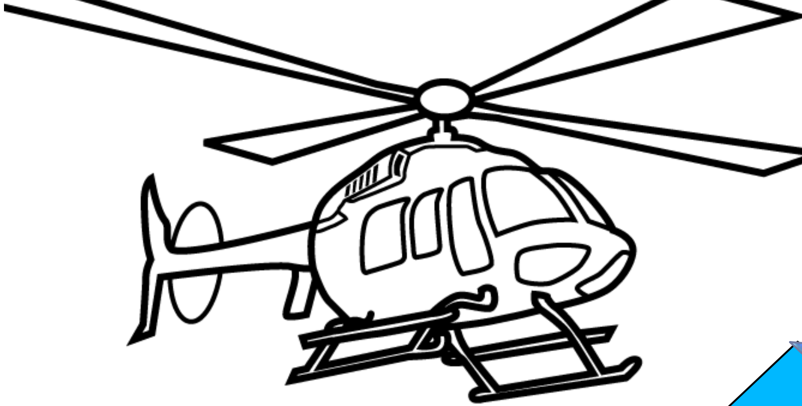
```
for (i=0; i<N; ++i) {
  points[i]->x += 1;
}
```

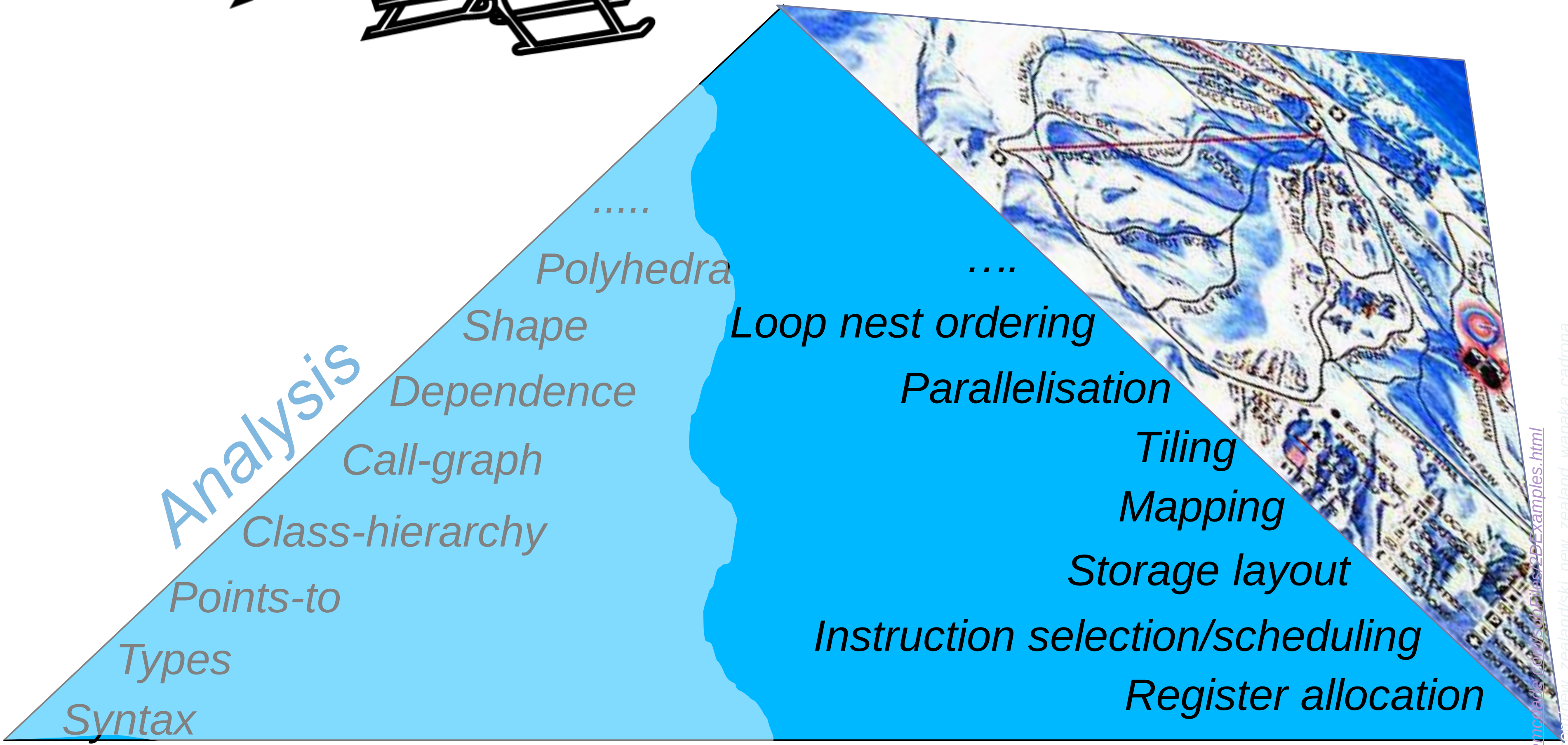
- Can the iterations of this loop be executed in parallel?



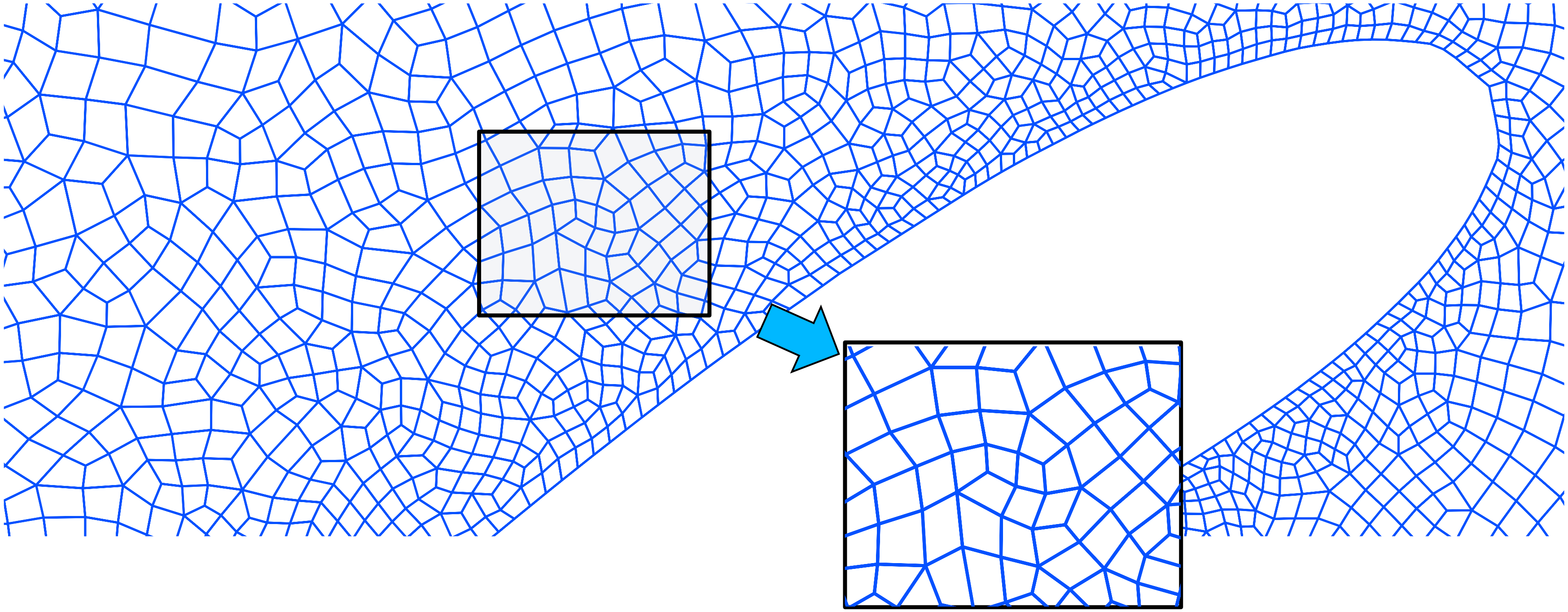
- Oh no: not all the iterations are independent!
  - You want to re-use piece of code in different contexts
  - Whether it’s parallel depends on context!

- Compilation is like skiing





- Analysis is not always the interesting part....
- It’s more fun the higher you start!



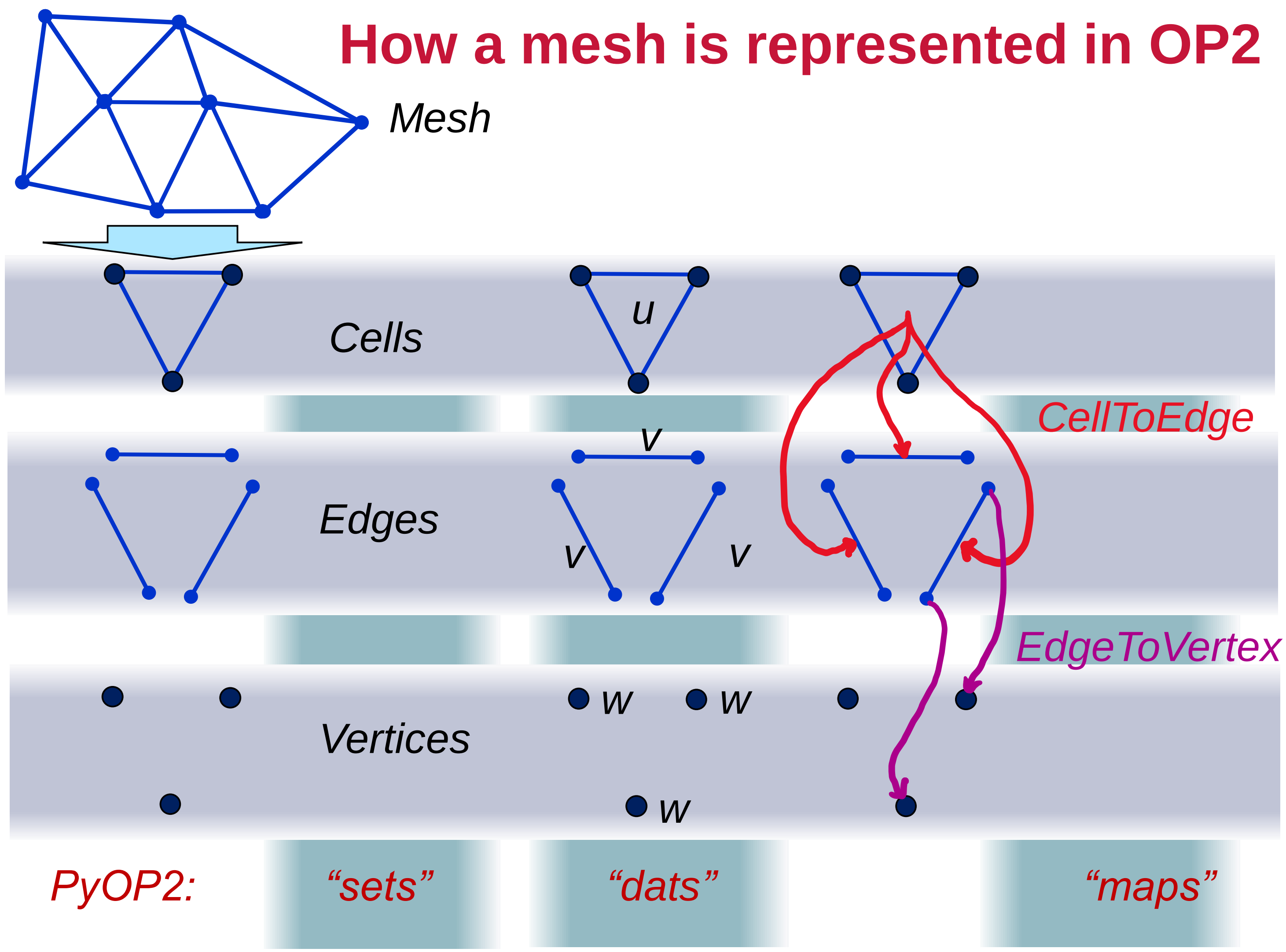
- Unstructured meshes require pointers/indirection because adjacency lists have to be represented explicitly
- A controlled form of pointers (actually a general graph)

- **OP2** is a C++ and Fortran library for parallel loops over the mesh, implemented by source-to-source transformation
- **PyOP2** is the same basic model, implemented in Python using runtime code generation

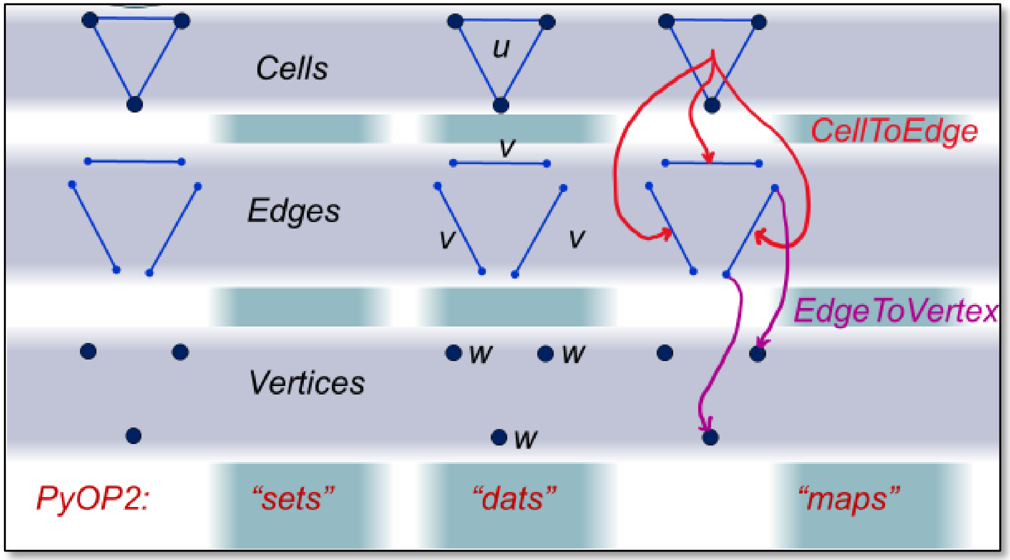
- Enables generation of highly-optimised vectorised, CUDA, OpenMP and MPI code

- The OP2 model originates from Oxford (Mike Giles et al)

How a mesh is represented in OP2



OP2 loops, access descriptors and kernels



op_par_loop(set, kernel, access descriptors)

We specify which **set** to iterate over

We specify a **kernel** to execute – the kernel operates entirely locally, on the **dats** to which it has access

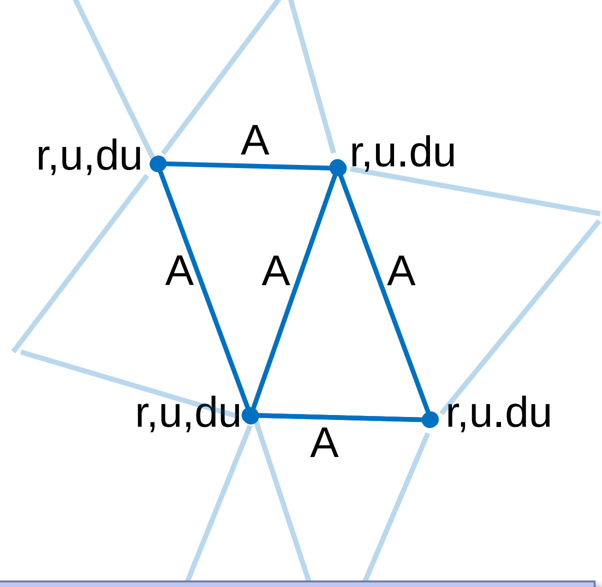
The **access descriptors** specify which dats the kernel has access to:

- **Which dats of the target set**
- **Which dats of sets indexed from this set through specified maps**

- OP2 separates local (kernel) from global (mesh)
- OP2 makes data dependence explicit

PyOP2: "decoupled access-execute"

- Parallel loops, over sets (nodes, edges etc)
- Access descriptors specify how to pass data to and from the C kernel
- The kernel operates only on local data



Access descriptors specify how to feed the kernel from the mesh

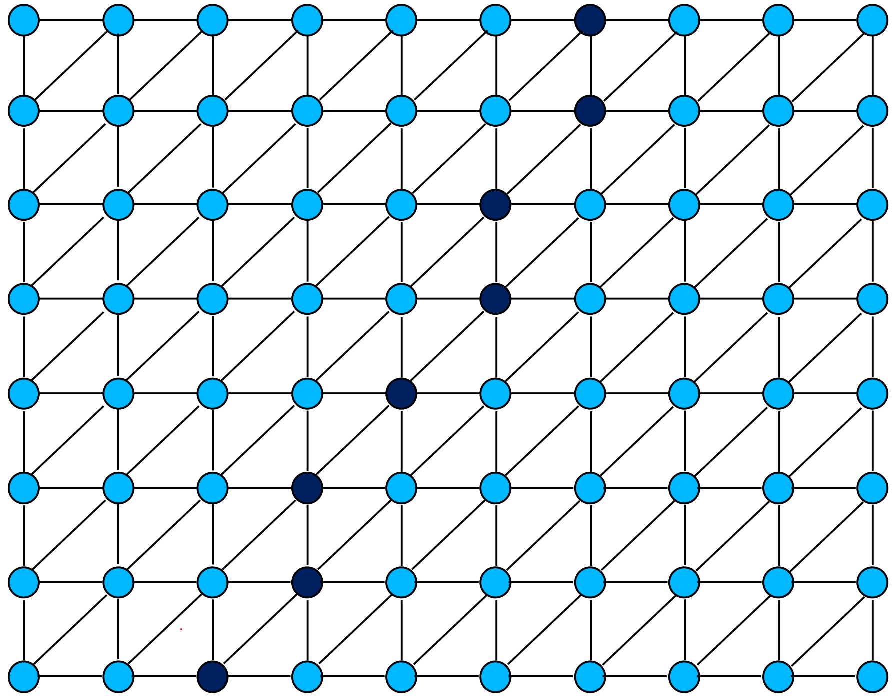
```
for iter in xrange(0, NITER):
    u_sum = op2.Global(1, data=0.0, np.float32)
    u_max = op2.Global(1, data=0.0, np.float32)
    op2.par_loop(res, edges,
        p_A(op2.READ),
        p_u(op2.READ, edge2vertex[1]),
        p_du(op2.INC, edge2vertex[0]),
        beta(op2.READ))

    op2.par_loop(update, nodes,
        p_r(op2.READ),
        p_du(op2.RW),
        p_u(op2.INC),
        u_sum(op2.INC),
        u_max(op2.MAX))
```

```
void res(float *A, float *u, float *du,
         const float *beta) {
  *du += (*beta) * (*A) * (*u);
}
```

```
void update(float *r, float *du, float *u, float
        *u_sum, float *u_max) {
  *u += *du + alpha * (*r);
  *du = 0.0f;
  *u_sum += (*u) * (*u);
  *u_max = *u_max > *u ? *u_max : *u;
}
```

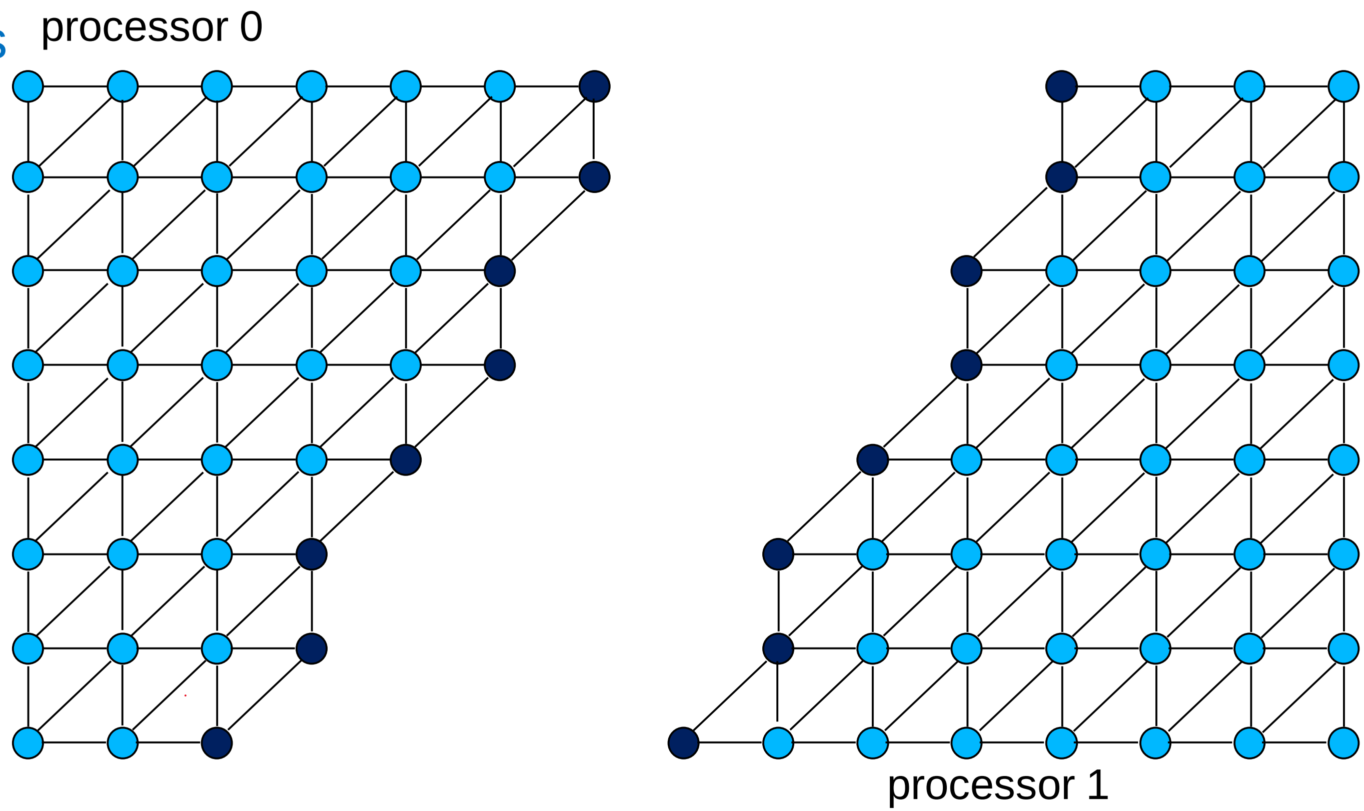
Code generation for indirect loops in PyOP2

- For MPI we precompute partitions & haloes
- Derived from PyOP2 access descriptors, implemented using PetSC DMPlex
- At partition boundaries, the entities (vertices, edges, cells) form layered halo region



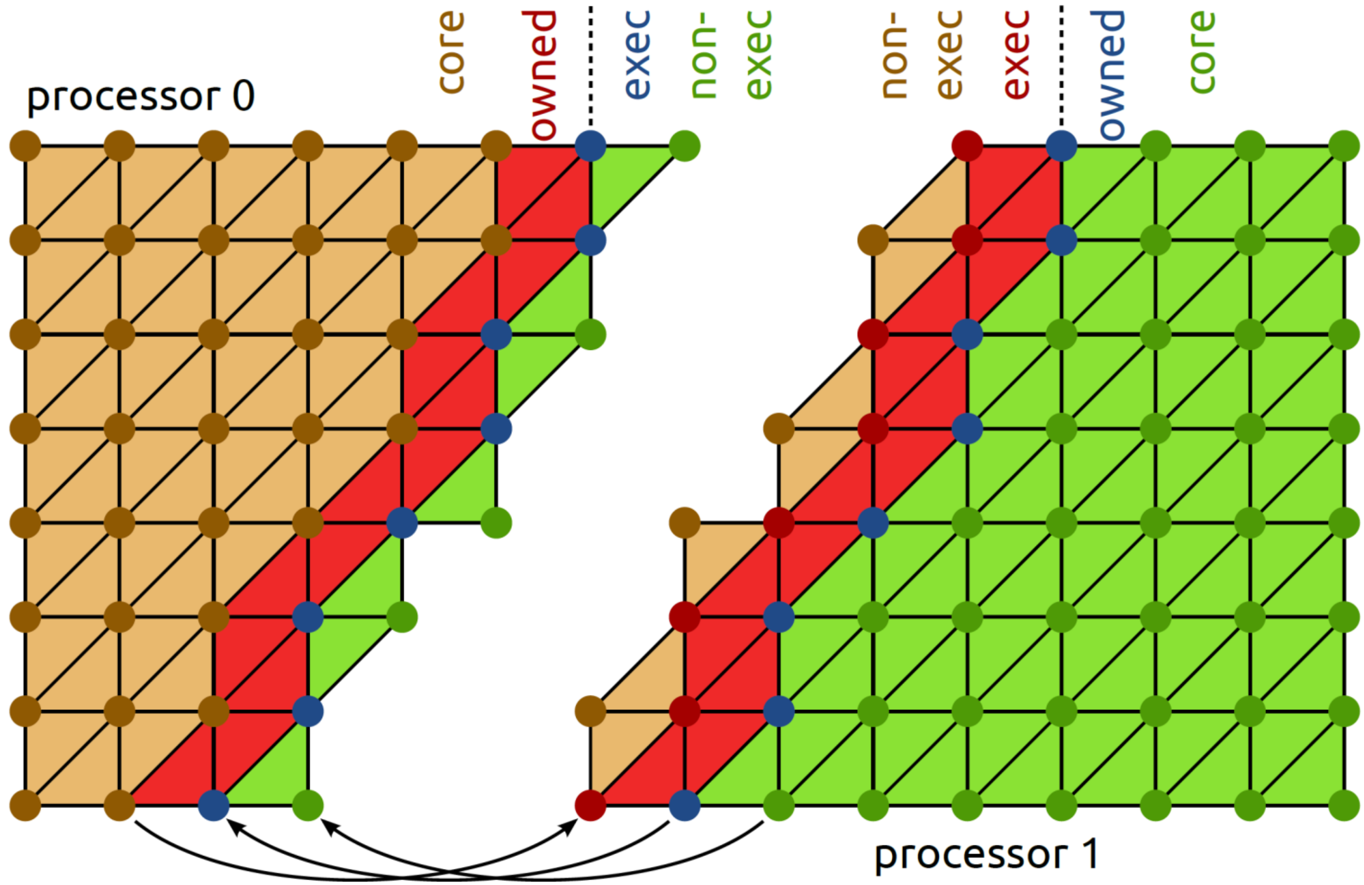
Code generation for indirect loops in PyOP2

- For MPI we precompute partitions & haloes
- Derived from PyOP2 access descriptors, implemented using PetSC DMPlex
- At partition boundaries, the entities (vertices, edges, cells) form layered halo region



Code generation for indirect loops in PyOP2

- For MPI we precompute partitions & haloes
- Derived from PyOP2 access descriptors, implemented using PetSC DMPlex
- At partition boundaries, the entities (vertices, edges, cells) form layered halo region



- **Core:** entities owned which can be processed without accessing halo data.
- **Owned:** entities owned which access halo data when processed
- **Exec halo:** off-processor entities which are redundantly executed over because they touch owned entities
- **Non-exec halo:** off-processor entities which are not processed, but read when computing the exec halo

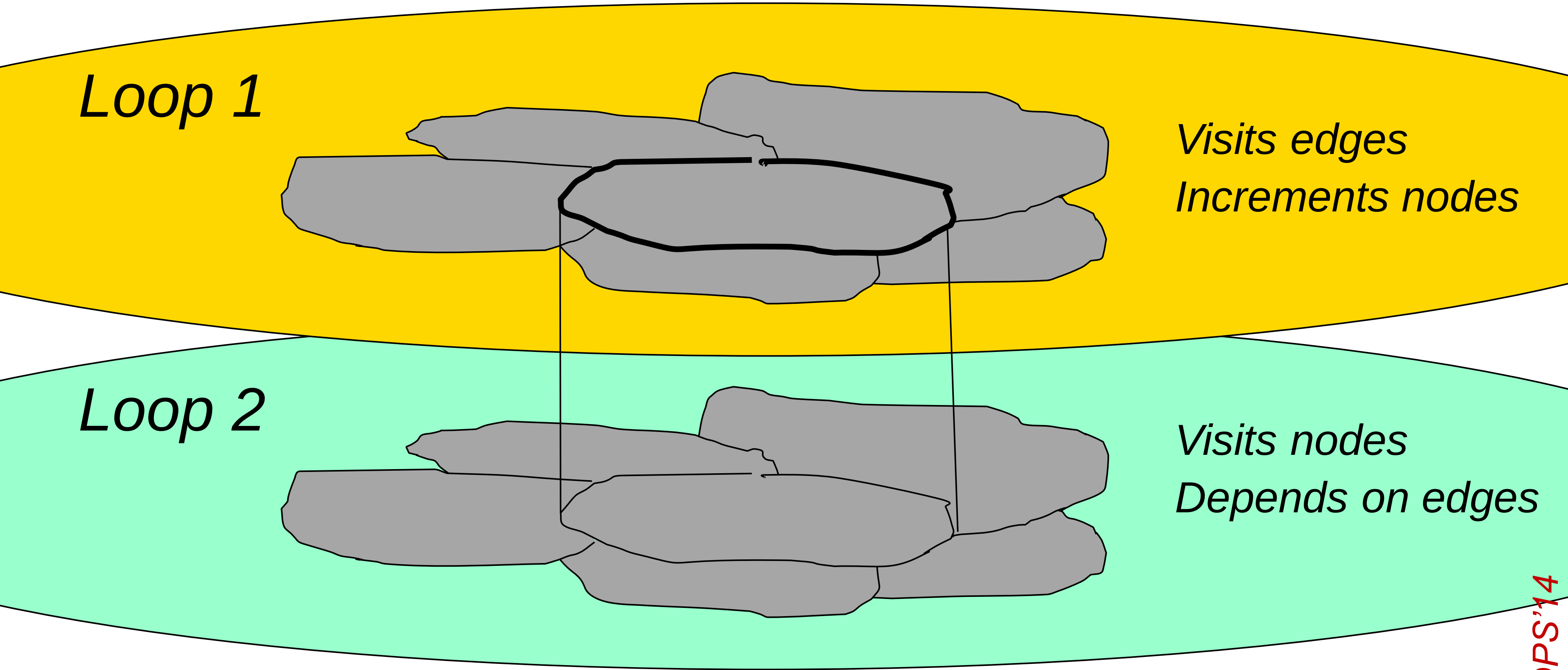
Florian Rathgeber PhD thesis, Productive and Efficient Computational Science Through Domain-specific Abstractions

Imperial College London

- Can we automate interesting optimisations that would be hard to do by hand?

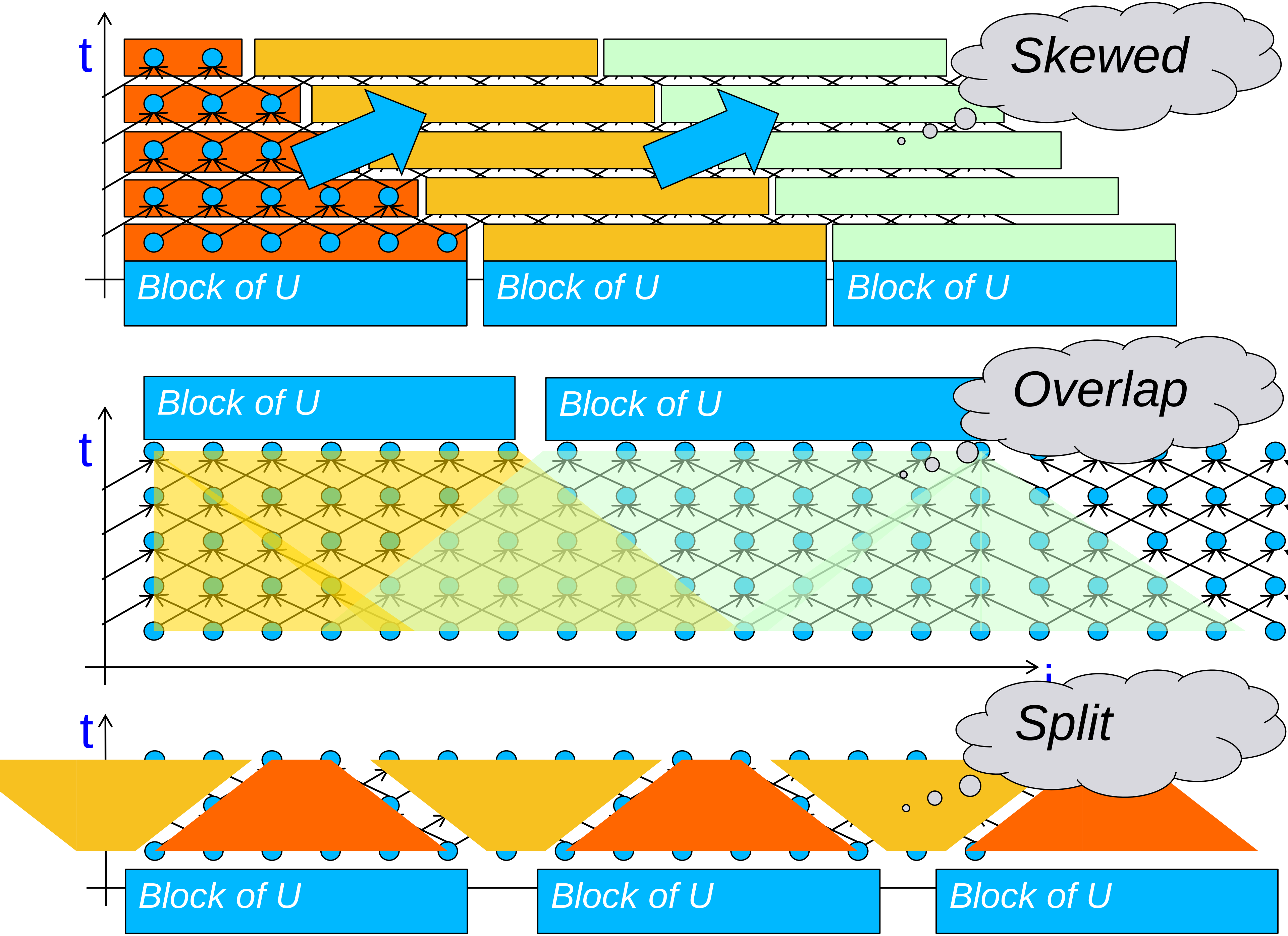
- First example:
  - Tiling for cache locality
  - (This optimisation has been implemented – and automated – but does not currently form part of the standard distribution)

Sparse split tiling on an unstructured mesh, for locality



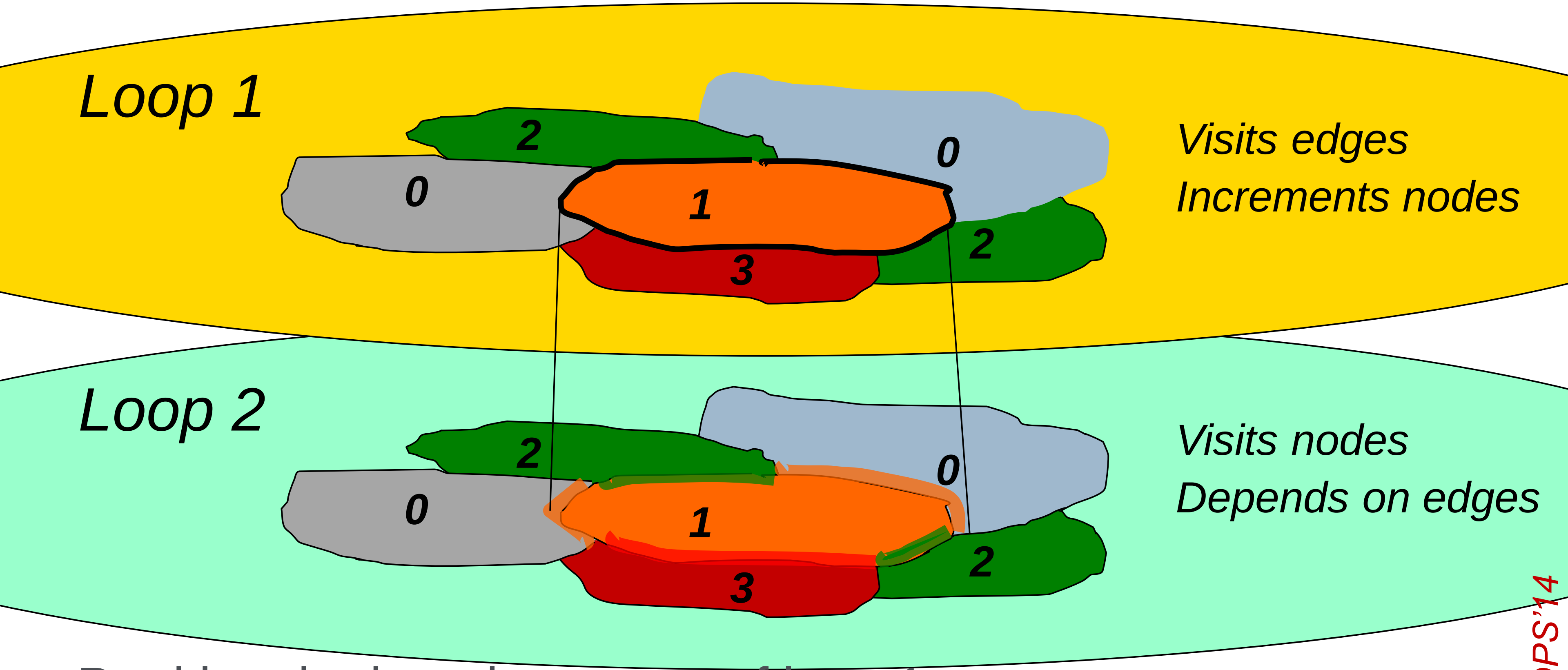
- How can we load a block of mesh and do the iterations of loop 1, then the iterations of loop 2, before moving to the next block?
- If we could, we could dramatically improve the memory access behaviour!

Strout, Luporini et al, IPDPS'14



Skewed
t
Block of U
Block of U
Block of U
Overlap
Block of U
Block of U
t
i
Split
t
Block of U
Block of U
Block of U

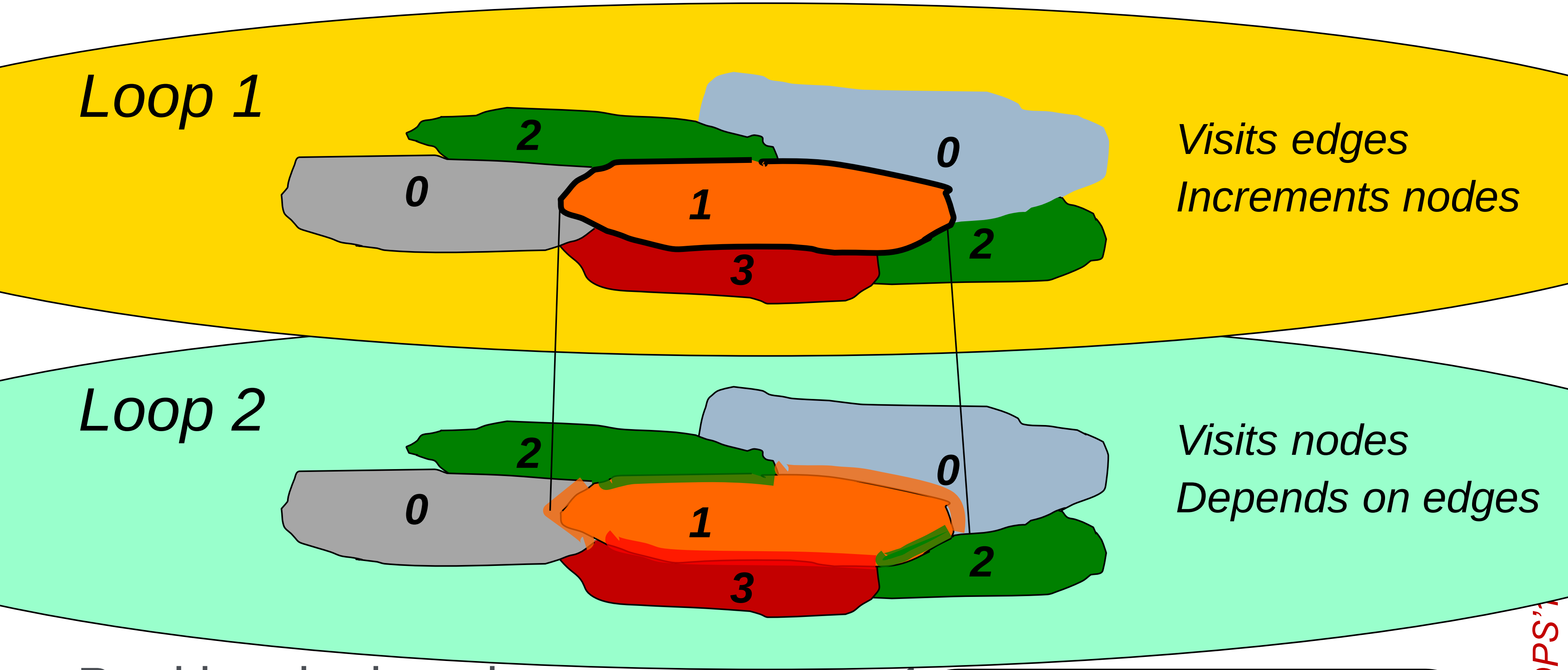
Sparse split tiling



- Partition the iteration space of loop 1
- Colour the partitions, execute the colours in order
- Project the tiles, using the knowledge that colour n can use data produced by colour n-1
- Thus, the tile coloured #1 ***grows*** where it meets colour #0
- And ***shrinks*** where it meets colours #2 and #3

Strout, Luporini et al, IPDPS'14

Sparse split tiling

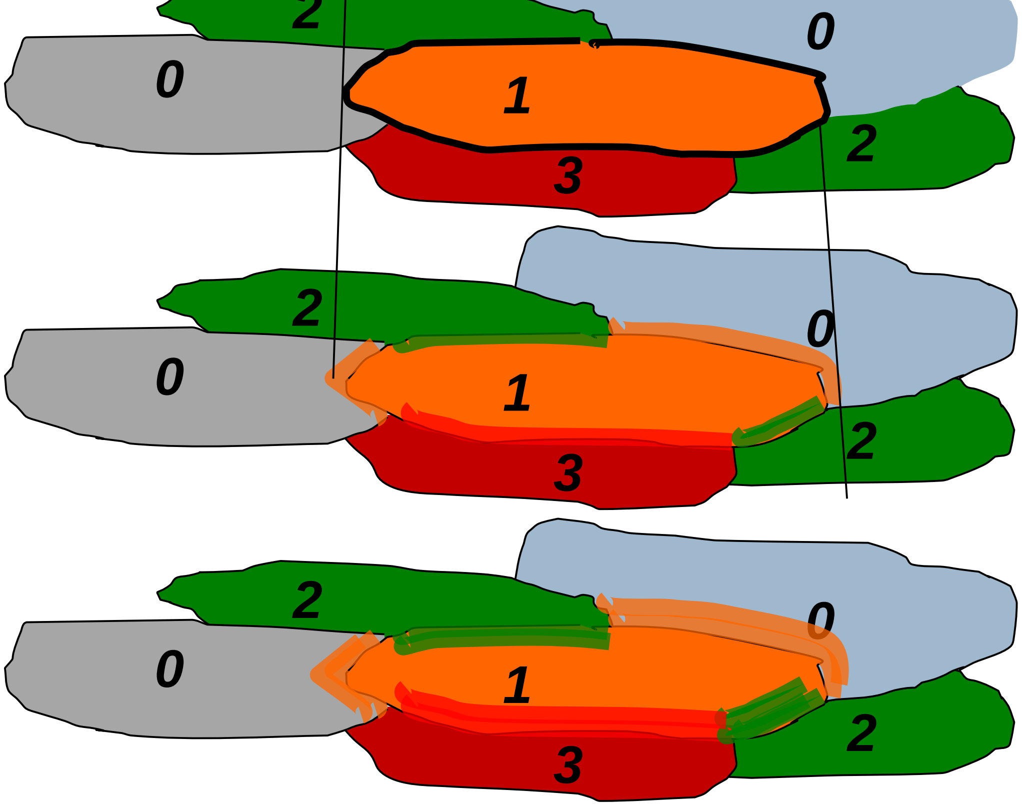


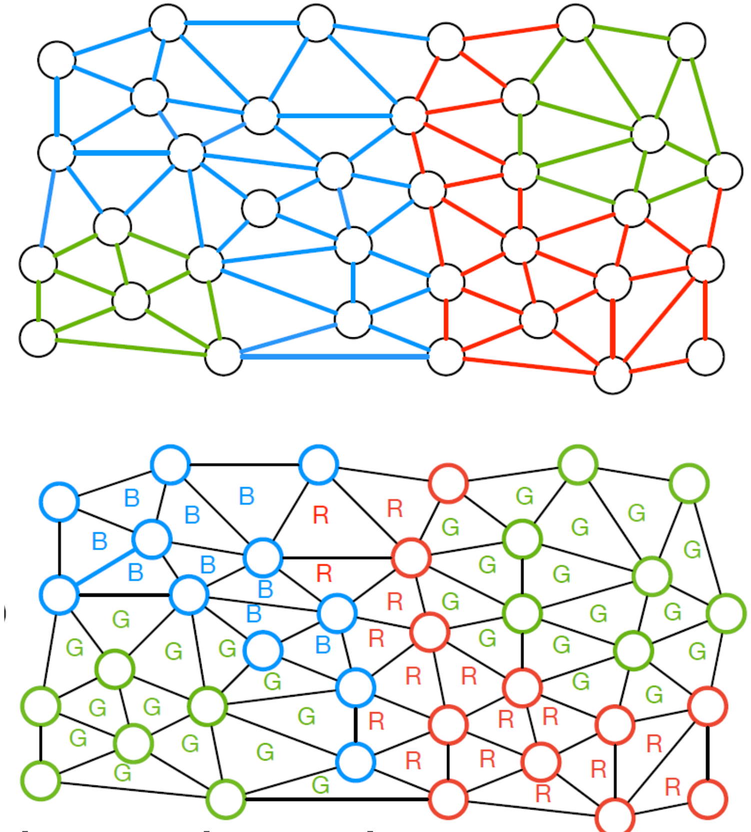
- Partition the iteration space of loop 1
- Colour the partitions
- Project the tiles, using the knowledg
  data produced by colour n-1
- Thus, the tile coloured #1 grows wh
- And shrinks where it meets colours #2 and #3

*Inspector-executor: derive tasks and task graph from the mesh, **at runtime***

Strout, Luporini et al, IPDPS'1

Tiles grow

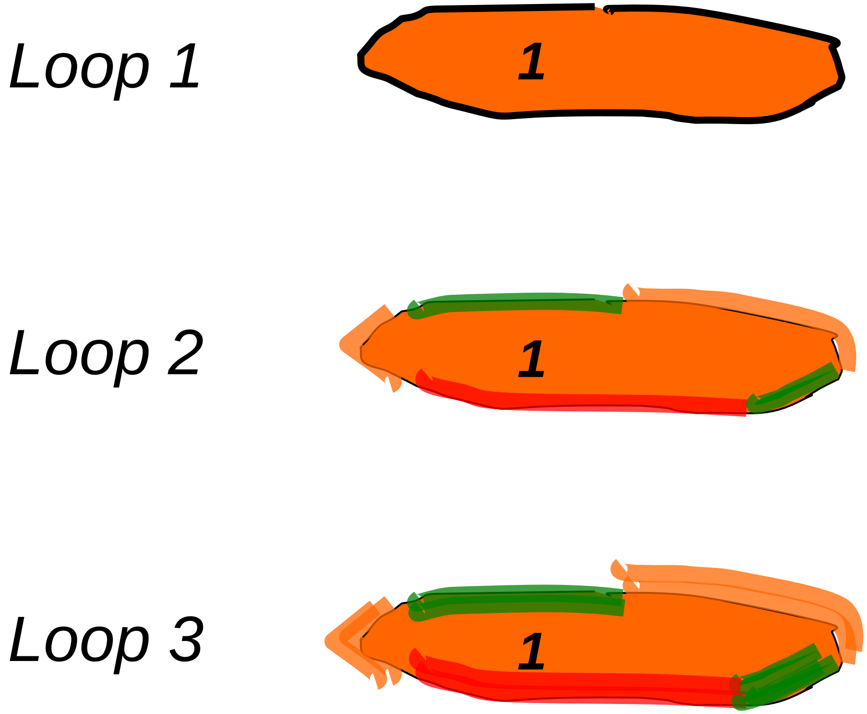


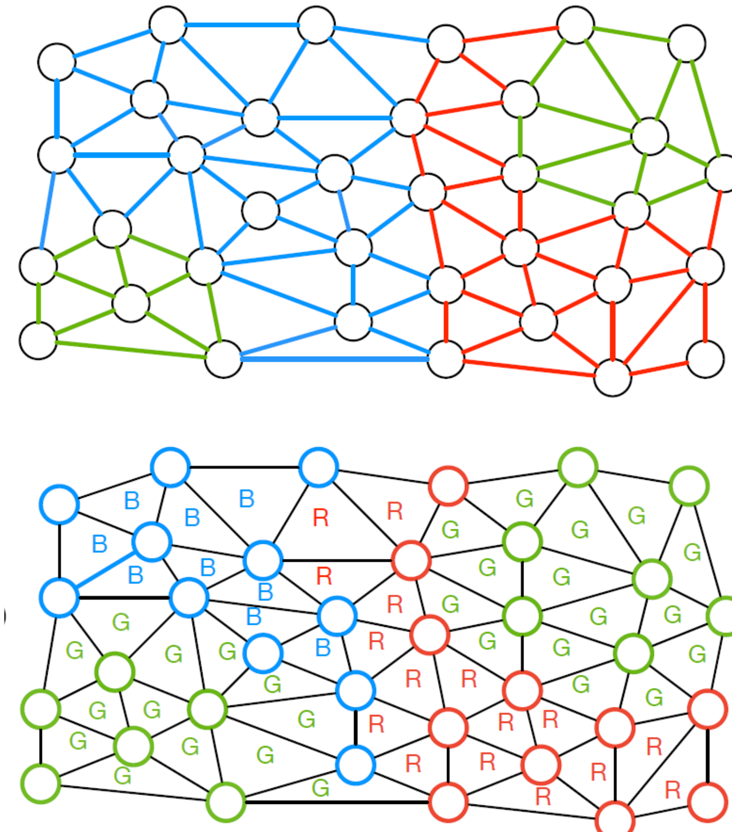


- As we project the tiles forward, tile shape degrades
- Perimeter-volume ratio gets worse

Strout, Luporini et al, IPDPS'14

Tiles grow

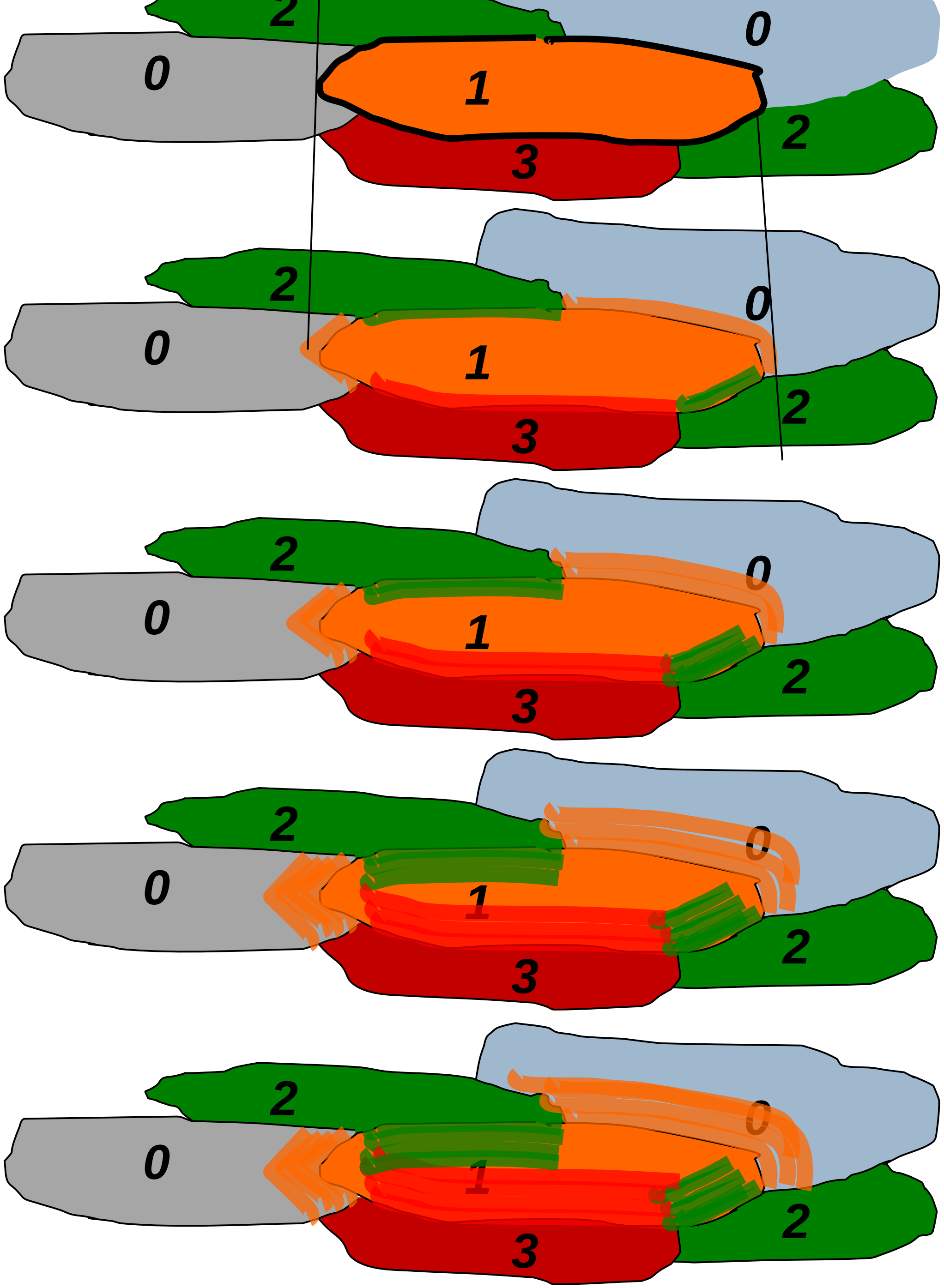


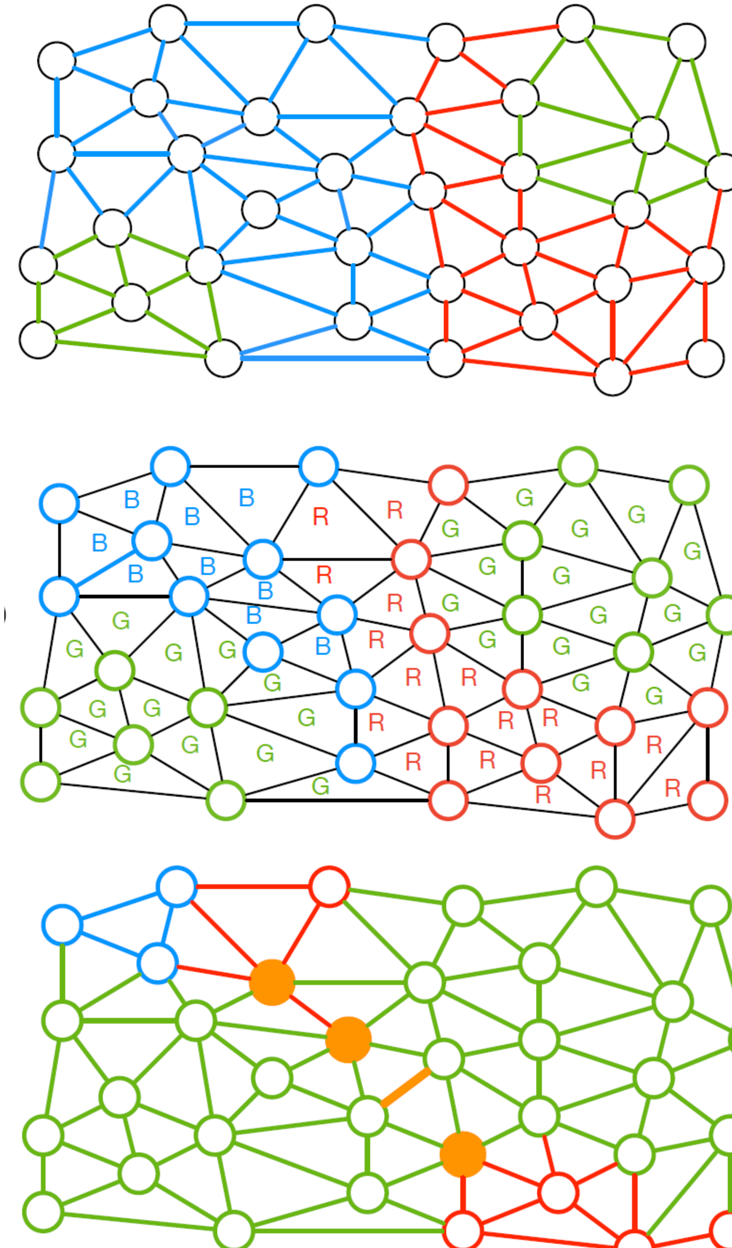


- As we project the tiles forward, tile shape degrades
- Perimeter-volume ratio gets worse
- We could partition Loop 1's data for the cache
- But Loop 2 and Loop 3 have different footprints
- So we rely on good (ideally space-filling-curve) numbering

Strout, Luporini et al, IPDPS'14

Tiles can collide





(1) Blue, (2) Red, (3) Green

(Luporini, et al, Automated Tiling of Unstructured Mesh Computations with Application to Seismological Modeling. ACM TOMS 2019)

Loop chains

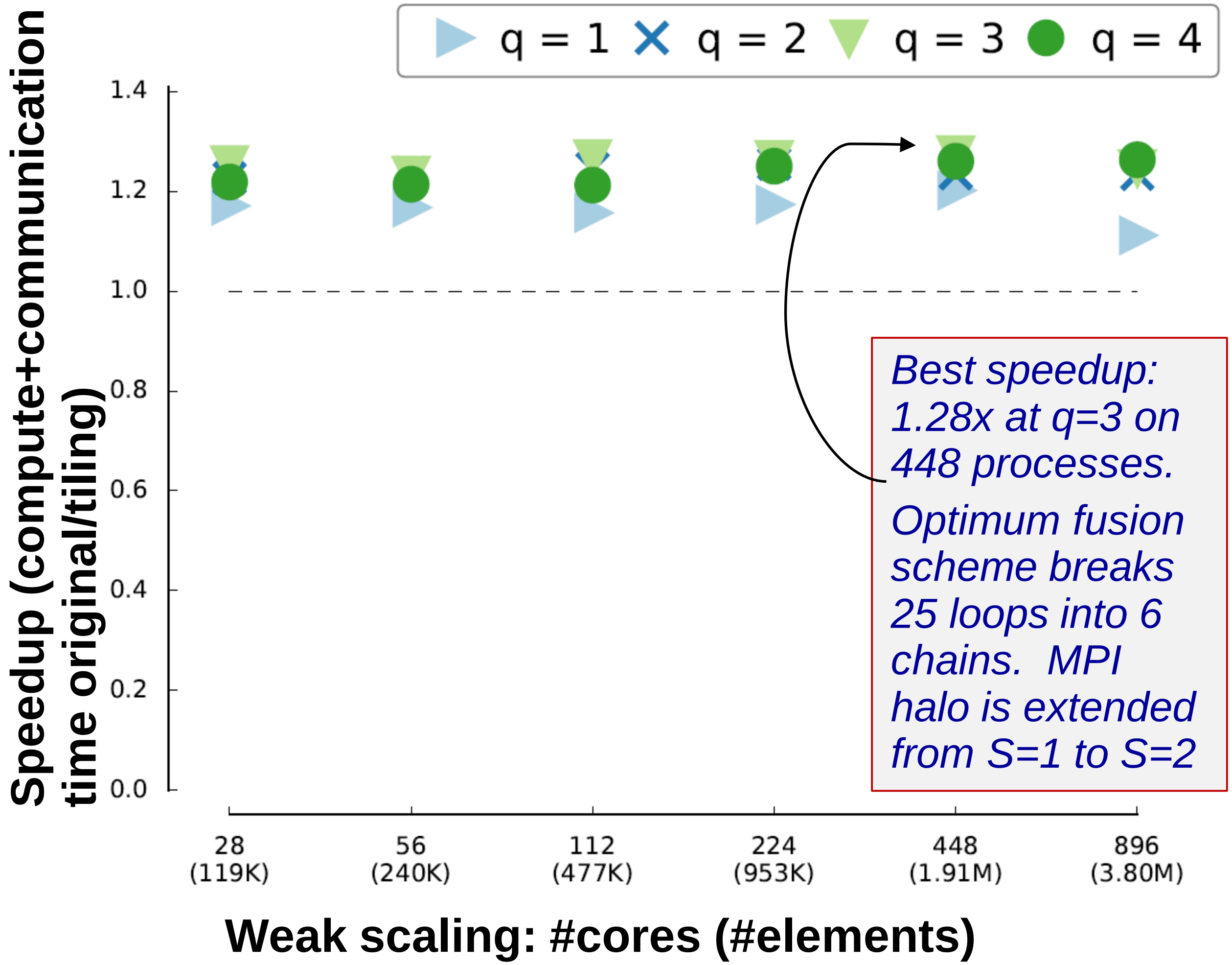
```
while t <= T + 1e-12 and timestep < ntimesteps:
    if op2.MPI.COMM_WORLD.rank == 0 and timestep % self.output == 0:
        info("t = %f, (timestep = %d)" % (t, timestep))
    with loop_chain("main1",
                    tile_size=self.tiling_size,
                    num_unroll=self.tiling_uf,
                    mode=self.tiling_mode,
                    extra_halo=self.tiling_halo,
                    explicit=self.tiling_explicit,
                    use_glb_maps=self.tiling_glb_maps,
                    use_prefetch=self.tiling_prefetch,
                    coloring=self.tiling_coloring,
                    ignore_war=True,
                    log=self.tiling_log):
        # In case the source is time-dependent, update the time 't' here.
        if(self.source):
            with timed_region('source term update'):
                self.source_expression.t = t
                self.source = self.source_expression

        # Solve for the velocity vector field.
        self.solve(self.rhs_uh1, self.velocity_mass_asdat, self.uh1)
        self.solve(self.rhs_stemp, self.stress_mass_asdat, self.stemp)
        self.solve(self.rhs_uh2, self.velocity_mass_asdat, self.uh2)
        self.solve(self.rhs_u1, self.velocity_mass_asdat, self.u1)

        # Solve for the stress tensor field.
        self.solve(self.rhs_sh1, self.stress_mass_asdat, self.sh1)
        self.solve(self.rhs_utemp, self.velocity_mass_asdat, self.utemp)
        self.solve(self.rhs_sh2, self.stress_mass_asdat, self.sh2)
        self.solve(self.rhs_s1, self.stress_mass_asdat, self.s1)

    self.u0.assign(self.u1)
    self.s0.assign(self.s1)

    # Write out the new fields
    self.write(self.u1, self.s1, self.tofile and timestep % self.output == 0)

    # Move onto next timestep
    t += self.dt
    timestep += 1
```

```
with loop_chain(tile_size=,....):
    # solve for velocity vector field
    self.solve(....);
    self.solve(....);
    self.solve(....);
    self.solve(....);
    # solve for stress tensor field
    self.solve(....);
    self.solve(....);
    self.solve(....);
    self.solve(....);
```

(25 op_par_loops per timestep, all tilable)

(Luporini, Lange, Jacobs, Gorman, Ramanujam, Kelly. Automated Tiling of Unstructured Mesh Computations with Application to Seismological Modeling. ACM TOMS 2019 https://doi.org/10.1145/3302256)

- **Example: Seigen**
- Elastic wave solver
- 2d triangular mesh
- Velocity-stress formulation
- 4th-order explicit leapfrog timestepping scheme
- Discontinuous-Galerkin, order q=1-4
- 32 nodes, 2x14-core E5-2680v4, SGI MPT 2.14
- 1000 timesteps (ca.1.15s/timestep)



- Up to 1.28x speedup
- Inspection about as much time as 2 timesteps
- Using RCM numbering – space-filling curve should lead to better results

(ACM TOMS 2019)

- Can we automate interesting optimisations that would be hard to do by hand?

- Second example:
  - Generalised loop-invariant code motion
  - (This optimisation has been implemented, automated, and re-implemented – and forms part of the standard distribution)

Recall:

```
t__ dat1, double const *__restrict__ dat0, int const *__restrict__ map0, int const *__restrict__ map1)
double form_t2;
double const form_t20[7 * 6] = { ... };
double form_t21...t37;
double form_t38[6];
double form_t39[6];
double form_t4;
double form_t40...t45;
double form_t5...t9;
double t0[6 * 2];
double t1[3 * 2];
double t2[6 * 2 * 6 * 2];

for (int n = start; n <= -1 + end; ++n)
{
  for (int i4 = 0; i4 <= 5; ++i4)
    for (int i5 = 0; i5 <= 1; ++i5)
      for (int i6 = 0; i6 <= 5; ++i6)
        for (int i7 = 0; i7 <= 1; ++i7)
          t2[24 * i4 + 12 * i5 + 2 * i6 + i7] = 0.0;
  for (int i2 = 0; i2 <= 2; ++i2)
    for (int i3 = 0; i3 <= 1; ++i3)
      t1[2 * i2 + i3] = dat1[2 * map1[3 * n + i2] + i3];
  for (int i0 = 0; i0 <= 5; ++i0)
    for (int i1 = 0; i1 <= 1; ++i1)
      t0[2 * i0 + i1] = dat0[2 * map0[6 * n + i0] + i1];
  form_t0 = -1.0 * t1[1];
  form_t1 = form_t0 + t1[3];
  form_t2 = -1.0 * t1[0];
  form_t3 = form_t2 + t1[2];
  form_t4 = form_t0 + t1[5];
  form_t5 = form_t2 + t1[4];
  form_t6 = form_t3 * form_t4 + -1.0 * form_t5 * form_t1;
  form_t7 = 1.0 / form_t6;
  form_t8 = form_t7 * -1.0 * form_t1;
  form_t9 = form_t4 * form_t7;
  form_t10 = form_t3 * form_t7;
  form_t11 = form_t7 * -1.0 * form_t5;
  form_t12 = 0.0001 * (form_t8 * form_t9 + form_t10 * form_t11);
  form_t13 = 0.0001 * (form_t8 * form_t8 + form_t10 * form_t10);
  form_t14 = 0.0001 * (form_t9 * form_t9 + form_t11 * form_t11);
  form_t15 = 0.0001 * (form_t9 * form_t8 + form_t11 * form_t10);
  form_t16 = fabs(form_t6);
  for (int form_ip = 0; form_ip <= 6; ++form_ip)
  {
    form_t26 = 0.0; form_t25 = 0.0; form_t24 = 0.0; form_t23 = 0.0; form_t22 = 0.0; form_t21 = 0.0;
    for (int form_i = 0; form_i <= 5; ++form_i)
    {
      form_t21 = form_t21 + form_t20[6 * form_ip + form_i] * t0[1 + 2 * form_i];
      form_t22 = form_t22 + form_t19[6 * form_ip + form_i] * t0[1 + 2 * form_i];
      form_t23 = form_t23 + form_t20[6 * form_ip + form_i] * t0[2 * form_i];
      form_t24 = form_t24 + form_t19[6 * form_ip + form_i] * t0[2 * form_i];
      form_t25 = form_t25 + form_t18[6 * form_ip + form_i] * t0[1 + 2 * form_i];
      form_t26 = form_t26 + form_t18[6 * form_ip + form_i] * t0[2 * form_i];
    }
    form_t27 = form_t17[form_ip] * form_t16;
    form_t28 = form_t27 * form_t15;
    form_t29 = form_t27 * form_t14;
    form_t30 = form_t27 * (form_t26 * form_t9 + form_t25 * form_t11);
    form_t31 = form_t27 * form_t13;
    form_t32 = form_t27 * form_t12;
    form_t33 = form_t27 * (form_t26 * form_t8 + form_t25 * form_t10);
    form_t34 = form_t27 * (form_t11 * form_t24 + form_t10 * form_t23);
    form_t35 = form_t27 * (form_t9 * form_t22 + form_t8 * form_t21);
    form_t36 = form_t27 * (50.0 + form_t9 * form_t24 + form_t8 * form_t23);
    form_t37 = form_t27 * (50.0 + form_t11 * form_t22 + form_t10 * form_t21);
    for (int form_k0 = 0; form_k0 <= 5; ++form_k0)
    {
      form_t38[form_k0] = form_t18[6 * form_ip + form_k0] * form_t37;
      form_t39[form_k0] = form_t18[6 * form_ip + form_k0] * form_t36;
    }
    for (int form_j0 = 0; form_j0 <= 5; ++form_j0)
    {
      form_t40 = form_t18[6 * form_ip + form_j0] * form_t35;
      form_t41 = form_t18[6 * form_ip + form_j0] * form_t34;
      form_t42 = form_t20[6 * form_ip + form_j0] * form_t31 + form_t18[6 * form_ip + form_j0] * form_t33 + form_t19[6 * form_ip + form_j0] * form_t32;
      form_t43 = form_t20[6 * form_ip + form_j0] * form_t28 + form_t18[6 * form_ip + form_j0] * form_t30 + form_t19[6 * form_ip + form_j0] * form_t29;
      for (int form_k0_0 = 0; form_k0_0 <= 5; ++form_k0_0)
      {
        form_t44 = form_t43 * form_t19[6 * form_ip + form_k0_0];
        form_t45 = form_t42 * form_t20[6 * form_ip + form_k0_0];
        t2[24 * form_j0 + 2 * form_k0_0] = t2[24 * form_j0 + 2 * form_k0_0] + form_t45 + form_t18[6 * form_ip + form_j0] * form_t39[form_k0_0] + form_t44;
        t2[13 + 24 * form_j0 + 2 * form_k0_0] = t2[13 + 24 * form_j0 + 2 * form_k0_0] + form_t45 + form_t18[6 * form_ip + form_j0] * form_t38[form_k0_0] + form_t44;
        t2[1 + 24 * form_j0 + 2 * form_k0_0] = t2[1 + 24 * form_j0 + 2 * form_k0_0] + form_t18[6 * form_ip + form_k0_0] * form_t41;
        t2[12 + 24 * form_j0 + 2 * form_k0_0] = t2[12 + 24 * form_j0 + 2 * form_k0_0] + form_t18[6 * form_ip + form_k0_0] * form_t40;
      }
    }
  }
  MatSetValuesBlockedLocal(mat0, 6, &(map0[6 * n]), 6, &(map0[6 * n]), &(t2[0]), ADD_VALUES);
}
}
```

- Generated code to assemble the resulting linear system matrix
- Executed at each triangle in the mesh
- Accesses degrees of freedom shared with neighbour triangles through indirection map

A simpler example:

```
void helmholtz(double A[3][3], double **coords) {
  // K, det = Compute Jacobian (coords)

  static const double W[3] = {...}
  static const double X_D10[3][3] = {{...}}
  static const double X_D01[3][3] = {{...}}

  for (int i = 0; i<3; i++)
    for (int j = 0; j<3; j++)
      for (int k = 0; k<3; k++)
        A[j][k] += ((Y[i][k]*Y[i][j]+
          +((K1*X_D10[i][k]+K3*X_D01[i][k])*(K1*X_D10[i][j]+K3*X_D01[i][j]))+
          +((K0*X_D10[i][k]+K2*X_D01[i][k])*(K0*X_D10[i][j]+K2*X_D01[i][j])))*
          *det*W[i]);
}
```

- Local assembly code generated by Firedrake for a Helmholtz problem on a 2D triangular mesh using Lagrange p = 1 elements.
- The local assembly operation computes a small dense submatrix
- These are combined to form a global system of simultaneous equations capturing the discretised conservation laws expressed by the PDE

Luporini, Varbenescu et al, ACM TACO/HiPEAC 2015

A simpler example:

```
void helmholtz(double A[3][3], double **coords) {
  // K, det = Compute Jacobian (coords)

  static const double W[3] = {...}
  static const double X_D10[3][3] = {{...}}
  static const double X_D01[3][3] = {{...}}

  for (int i = 0; i<3; i++)
    for (int j = 0; j<3; j++)
      for (int k = 0; k<3; k++)
        A[j][k] += ((Y[i][k]*Y[i][j]+
          +((K1*X_D10[i][k]+K3*X_D01[i][k])*(K1*X_D10[i][j]+K3*X_D01[i][j]))+
          +((K0*X_D10[i][k]+K2*X_D01[i][k])*(K0*X_D10[i][j]+K2*X_D01[i][j]))
          *det*W[i]);
}
```

- Local assembly code generated by Firedrake for a Helmholtz problem on a 2D triangular mesh using Lagrange p = 1 elements.
- The local assembly operation computes a small dense submatrix
- These are combined to form a global system of simultaneous equations capturing the discretised conservation laws expressed by the PDE

Luporini, Varbenescu et al, AC TACO/HiPEAC 2015

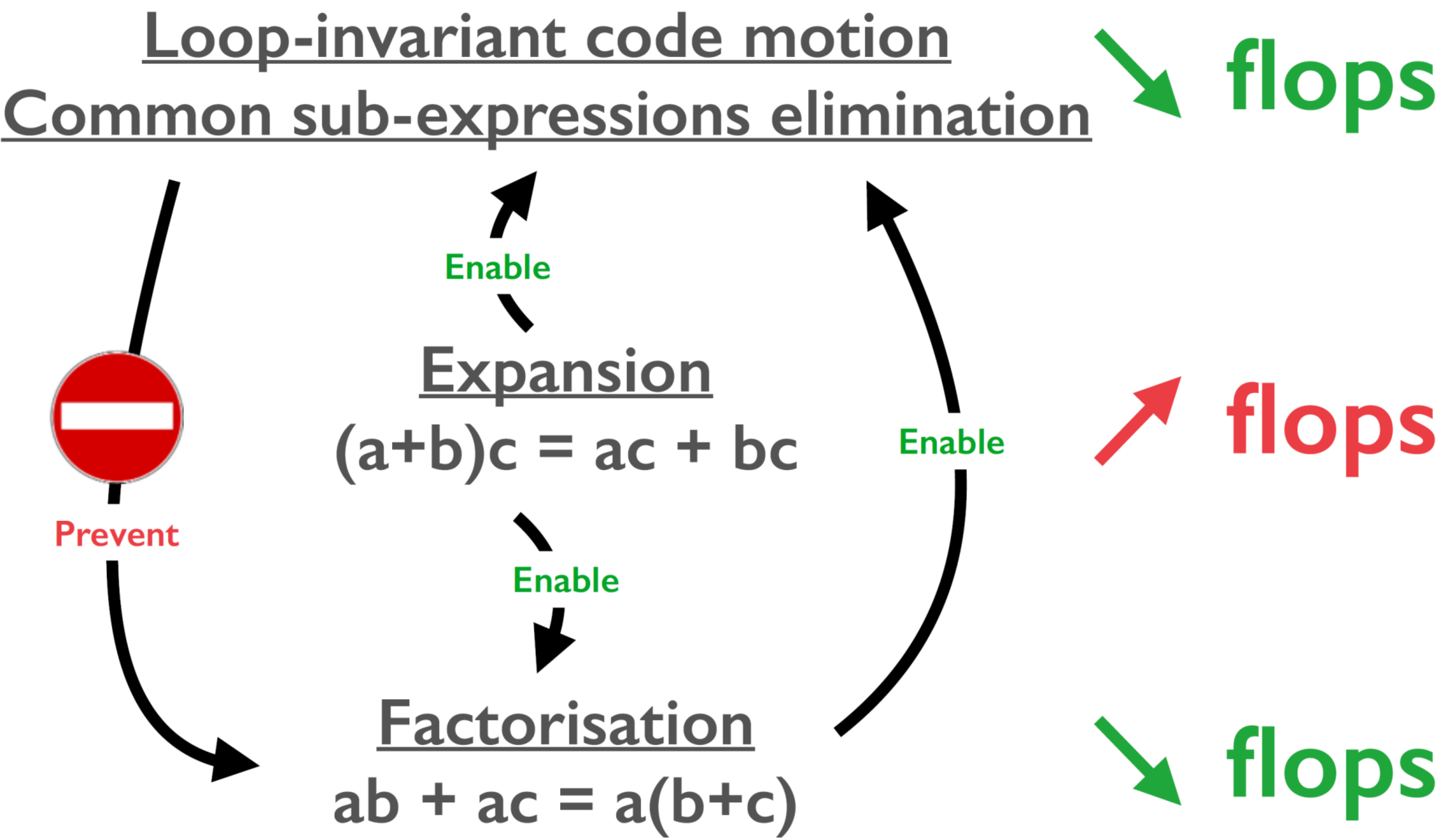
Generalised loop-invariant code motion:

```
void helmholtz(double A[3][4], double **coords) {
  #define ALIGN __attribute__((aligned(32)))
  // K, det = Compute Jacobian (coords)

  static const double W[3] ALIGN = {...}
  static const double X_D10[3][4] ALIGN = {{...}}
  static const double X_D01[3][4] ALIGN = {{...}}

  for (int i = 0; i<3; i++) {
    double LI_0[4] ALIGN;
    double LI_1[4] ALIGN;
    for (int r = 0; r<4; r++) {
      LI_0[r] = ((K1*X_D10[i][r])+(K3*X_D01[i][r]));
      LI_1[r] = ((K0*X_D10[i][r])+(K2*X_D01[i][r]));
    }
    for (int j = 0; j<3; j++)
      #pragma vector aligned
      for (int k = 0; k<4; k++)
        A[j][k] += (Y[i][k]*Y[i][j]+LI_0[k]*LI_0[j]+LI_1[k]*LI_1[j])*det*W[i]);
  }
}
```

- Local assembly code for the Helmholtz problem after application of
  - padding,
  - data alignment,
  - Loop-invariant code motion
- In this example, sub-expressions invariant to j are identical to those invariant to k, so they can be precomputed once in the r loop

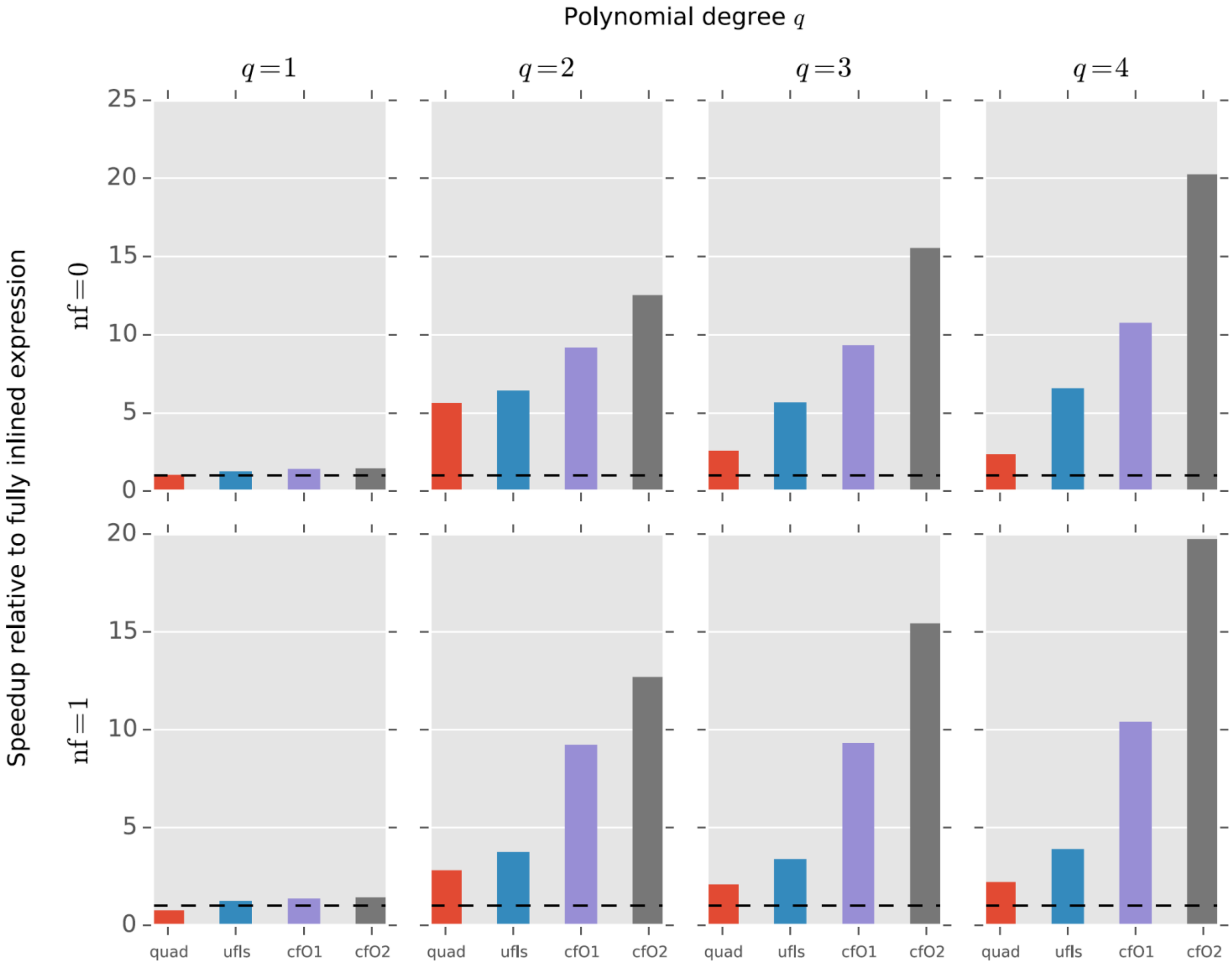
Luporini, Varbenescu et al, AC TACO/HiPEAC 2015

ARSENAL FOR REDUCING FLOPS



We formulate an ILP problem to find the best factorisation strategy

FOCUS ON HYPERELASTICITY



- Hyperelasticity
- Sandy Bridge (icc)
- Small 3D mesh (fit L3)

FFC-opt

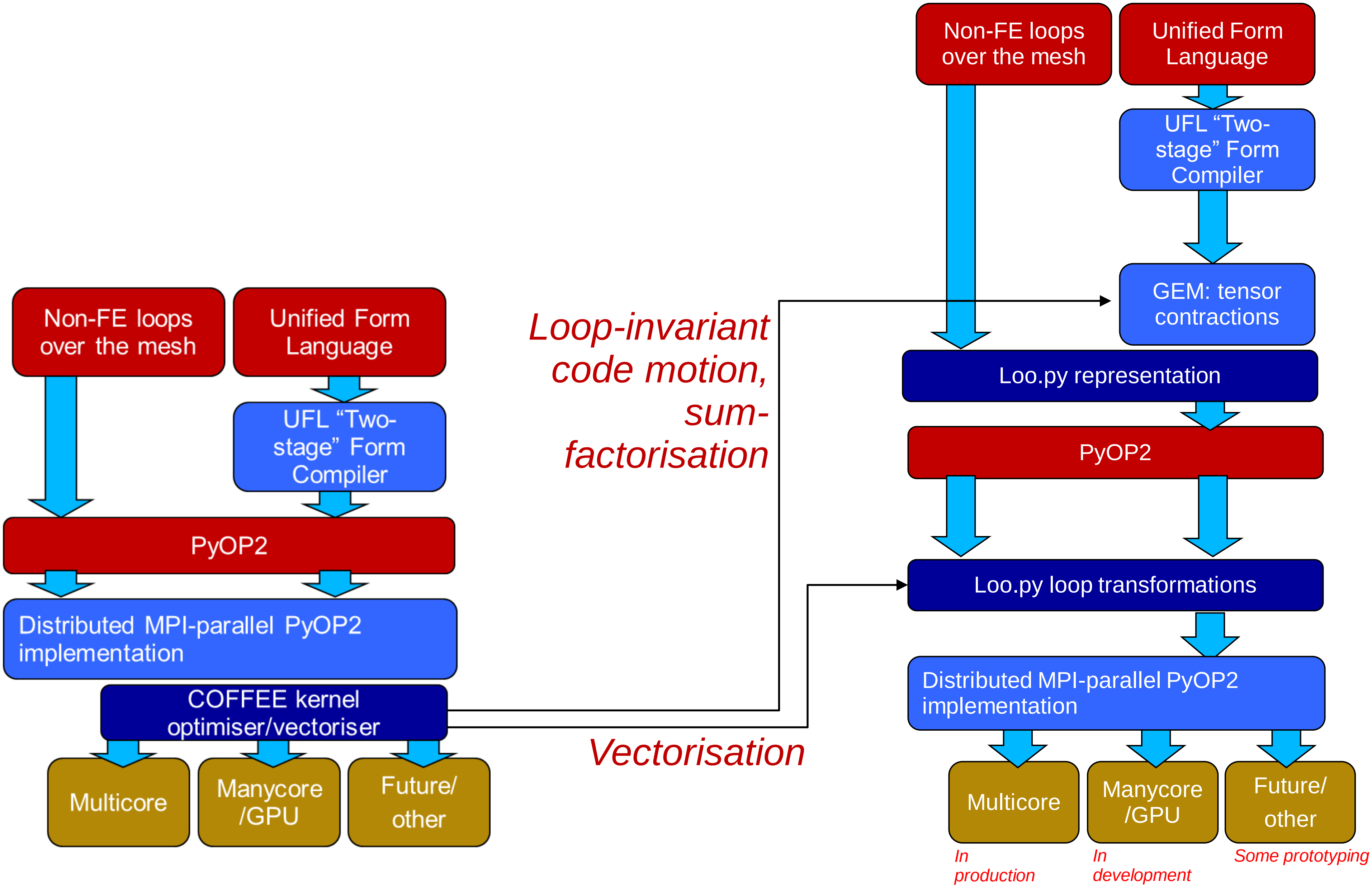
UFLACS

COFFEE-v1

This talk

F. Luporini, D.A. Ham, P.H.J. Kelly. An algorithm for the optimization of finite element integration loops. ACM Transactions on Mathematical Software (TOMS), 2017).

Firedrake's "Compiler architecture" has evolved over time



Why I do what I do, and what I’ve learned

- **Engaging with applications to exploit domain-specific optimisations can be incredibly fruitful**
  - Compiling general purpose languages is worthy but usually incremental
- **Compiler architecture is all about designing intermediate representations – that make hard things look easy**
  - Tools to deliver domain-specific optimisations often have domain-specific representations
  - Premature lowering is the constant enemy (appropriate lowering is great)
- **Along the way, we learn something about building better general-purpose compilers and programming abstractions**
  - Drill vertically, expand horizontally

What are the open research challenges?

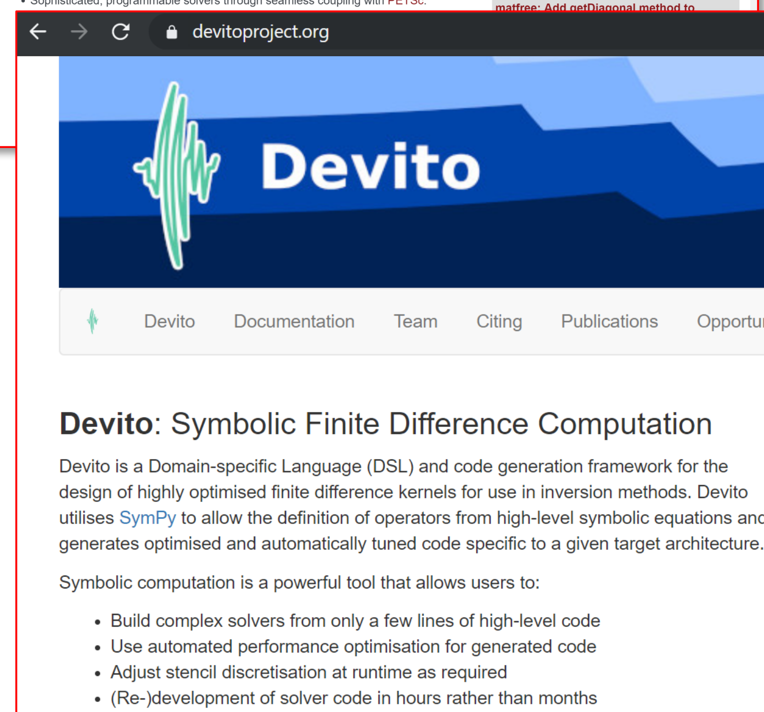
- **Sparse unstructured tiling really works, but didn't make it into the main trunk**
  - **It's just too complicated to justify the additional maintenance burden**
  - **It only helps some applications**
  - **We need to find a way to make it easier!**
- **Improved strong-scaling**
- **GPUs (and other accelerators?)**
- **Coupled problems (in-progress)**
- **Particles, particle transport**
- **Mesh adaptation, load balancing**

Things that I haven't had time to talk about:

- **Automatic adjoints, inverse problems (in-service)**
- **Interface/integration with PetSc (in-service)**
- **Hybridisation, static condensation (in-service, could be faster)**

How can we change the world?

- **The real value of Firedrake is in supporting the applications users in exploring *their* design space**
- **We enable them to navigate rapidly through alternative solutions to their problem**
- **We break down barriers that prevent the right tool being used for the right problem**
- **Firedrake automates the finite element method**
- **The Devito project automates finite difference**
- **In the future, we will have automated pathways from maths to code for many classes of problem, and many alternative solution techniques**



Have your cake and eat it too

- We **can** simultaneously
  - raise the level at which programmers can reason about code,
  - provide the compiler with a model of the computation that enables it to generate faster code than you could reasonably write by hand
  - Program generation is how we do it

Imperial College London

Acknowledgements

Partly funded/supported by

- NERC Doctoral Training Grant (NE/G523512/1)
- EPSRC “MAPDES” project (EP/I00677X/1)
- EPSRC “PSL” project (EP/I006761/1)
- Rolls Royce and the TSB through the SILOET programme
- EPSRC “PAMELA” Programme Grant (EP/K008730/1)
- EPSRC “PRISM” Platform Grants (EP/I006761/1 and EP/R029423/1)
- EPSRC “Custom Computing” Platform Grant (EP/I012036/1)
- EPSRC “Application Customisation” Platform Grant (EP/P010040/1)
- EPSRC “A new simulation and optimisation platform for marine technology” (EP/M011054/1)
- Basque Centre for Applied Mathematics (BCAM)

- Code:
  - http://www.firedrakeproject.org/
  - http://op2.github.io/PyOP2/
  - https://github.com/OP-DSL/OP2-Common