Compiler technology for solving PDEs with performance portability

Paul H J Kelly
Group Leader, Software Performance Optimisation
Co-Director, Centre for Computational Methods in Science and Engineering
Department of Computing, Imperial College London

Joint work with :

David Ham (Imperial Computing/Maths/Grantham Inst for Climate Change)
Gerard Gorman, (Imperial Earth Science Engineering – Applied Modelling and Computation Group)
Mike Giles, Gihan Mudalige, Istvan Reguly (Mathematical Inst, Oxford)
Doru Bercea, Fabio Luporini, Graham Markall, Lawrence Mitchell, Florian Rathgeber, George Rokos (Software Perf Opt Group, Imperial Computing)
Spencer Sherwin (Aeronautics, Imperial), Chris Cantwell (Cardio-mathematics group, Mathematics, Imperial)
Michelle Mills Strout, Chris Krieger, Cathie Olschanowsky (Colorado State University)
Carlo Bertolli (IBM Research)
Ram Ramanujam (Louisiana State University)

Have your cake and eat it too

This talk is about the following idea:

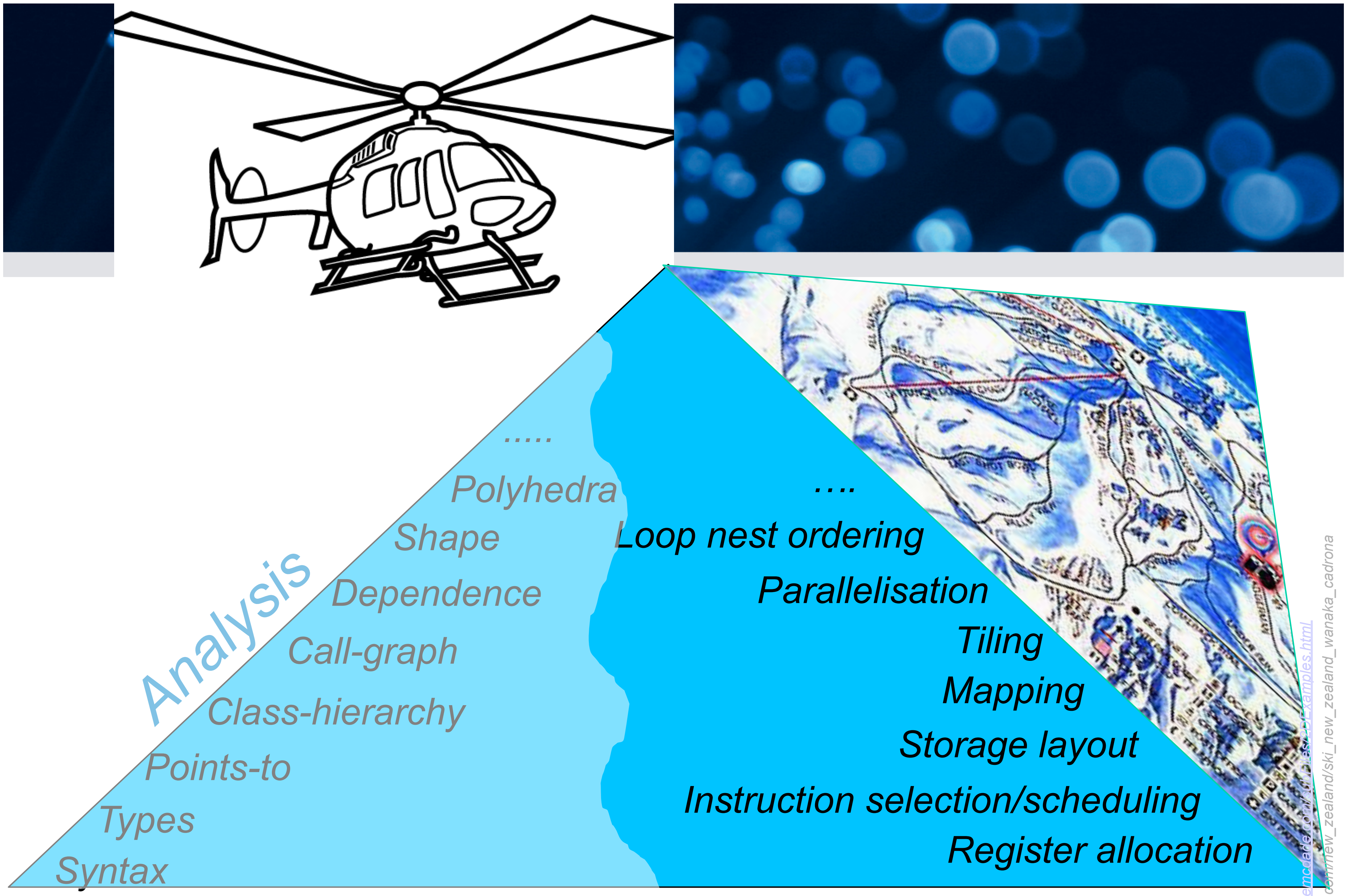
- can we simultaneously
  - raise the level at which programmers can reason about code,
  - provide the compiler with a model of the computation that enables it to generate faster code than you could reasonably write by hand?

Imperial College London

Have your cake and eat it too

This talk is about the following idea:

- can we 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?



- Compilation is like skiiing
- Analysis is not always the interesting part....

What we are doing….

Targetting MPI, OpenMP, OpenCL, Dataflow/ FPGA, from supercomputers to mobile, embedded and wearable

Projects

- ***PyOP2/OP2*** *Unstructured-mesh stencils*
- ***Firedrake*** *Finite-element assembly*
- ***PAMELA*** *Dense SLAM – 3D vision*
- ***PRAgMaTIc*** *Dynamic mesh adaptation*
- ***GiMMiK*** *Small-matrix multiplication*
- ***TINTL*** *Fourier interpolation*

Contexts

- ***Finite-volume CFD***
- ***Finite-element***
- *Real-time 3D scene understanding*
- *Adaptive-mesh CFD*
- *Unsteady CFD - higher-order flux-reconstruction*
- *Ab-initio computational chemistry (ONETEP)*

Technologies

- *Vectorisation, parametric polyhedral tiling*
- *Tiling for unstructured-mesh stencils*
- *Lazy, data-driven compute-communicate*
- *Runtime code generation*
- *Multicore graph worklists*
- *Massive common sub-expressions*
- *Optimisation of composite transforms*

Applications

- *Aeroengine turbo-machinery*
- *Weather and climate*
- *Domestic robotics, augmented reality*
- *Tidal turbines*
- *Formula-1, UAVs*
- *Solar energy, drug design*

What we are doing…

Targetting MPI, OpenMP, OpenCL, Dataflow/ FPGA, from supercomputers to mobile, embedded and wearable

Projects

- ***PyOP2/OP2*** *Unstructured-mesh stencils*
- ***Firedrake*** *Finite-element assembly*
- ***PAMELA*** *Dense SLAM – 3D vision*
- ***PRAgMaTIc*** *Dynamic mesh adaptation*
- ***GiMMiK*** *Small-matrix multiplication*
- ***TINTL*** *Fourier interpolation*

Contexts

- ***Finite-volume CFD***
- ***Finite-element***
- *Real-time 3D scene understanding*
- *Adaptive-mesh CFD*
- *Unsteady CFD - higher-order flux-reconstruction*
- *Ab-initio computational chemistry (ONETEP)*

Technologies

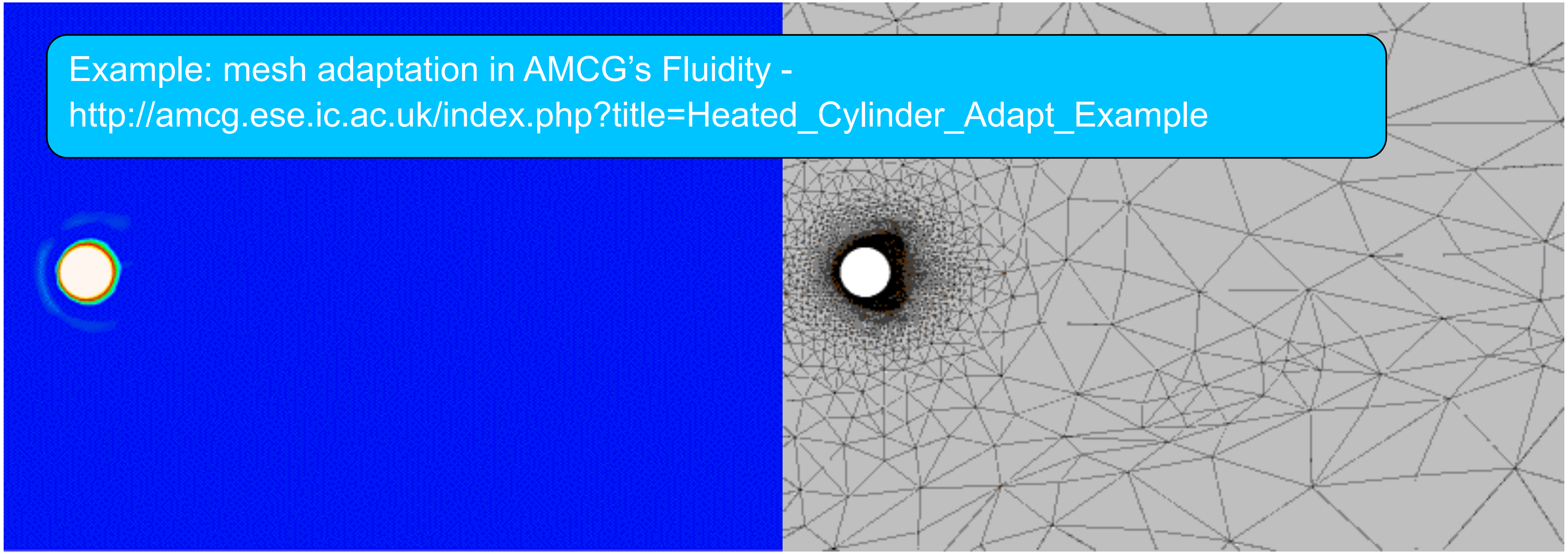
- *Vectorisation, parametric polyhedral tiling*
- *Tiling for unstructured-mesh stencils*
- *Lazy, data-driven compute-communicate*
- *Runtime code generation*
- *Multicore graph worklists*
- *Massive common sub-expressions*
- *Optimisation of composite transforms*

Applications

- *Aeroengine turbo-machinery*
- *Weather and climate*
- *Tidal turbines*
- *Domestic robotics, augmented reality*
- *Formula-1, UAVs*
- *Solar energy, drug design*

This talk

- *OP2 and PyOP2: parallel loops over unstructured meshes*
- *How well does it work?*
- *Loop fusion and tiling for unstructured-meshes*
- *Firedrake: a compiler for a higher-level DSL*
- *COFFEE: a compiler for a lower-level DSL*
- ***This talk's message:***
  - ***Avoid analysis for transformational optimisation***
  - ***Transform at the right level of abstraction***
  - ***Design representations that* get the abstraction right**

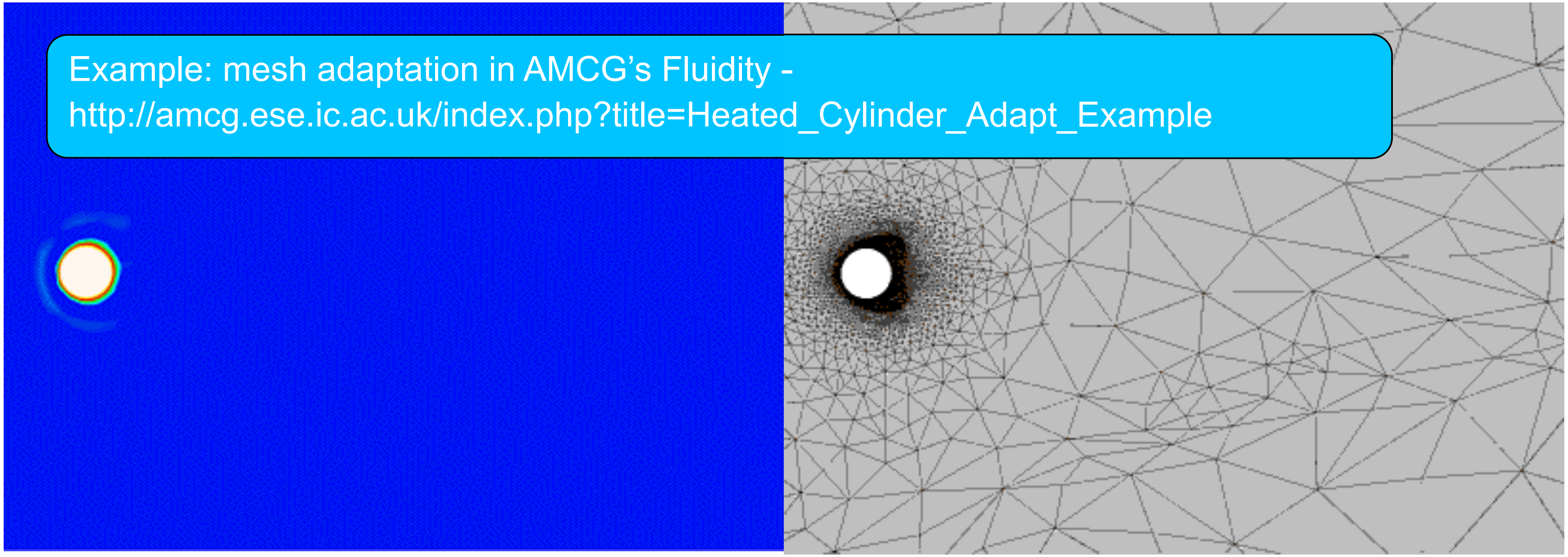


- Unstructured mesh
- Sometimes adaptive (not in the rest of this talk)

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

- Generates highly-optimised CUDA, OpenMP and MPI code

- Key idea: parallel loop has access descriptor providing declarative specification of the data access



- Unstructured mesh
- Sometimes adaptive (not in the rest of this talk)

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

- Generates highly-optimised CUDA, OpenMP and MPI code

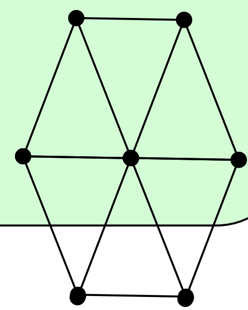
- Key idea: parallel loop has access descriptor providing declarative specification of the data access

Imperial College London

PyOP2 – an active library for unstructured mesh computations

```
# declare sets, maps, and datasets
nodes = op2.Set(nnode)
edges = op2.Set(nedge)

ppedge = op2.Map(edges, nodes, 2, pp)

p_A = op2.Dat(edges, data=A)
p_r = op2.Dat(nodes, data=r)
p_u = op2.Dat(nodes, data=u)
p_du = op2.Dat(nodes, data=du)

# global variables and constants declarations
alpha = op2.Const(1, data=1.0, np.float32)
beta = op2.Global(1, data=1.0, np.float32)
```

```
for iter in xrange(0, NITER):
  op2.par_loop(res, edges,
          p_A(op2.READ),
          p_u(op2.READ, ppedge[1]),
          p_du(op2.INC, ppedge[0]),
          beta(op2.READ))

  u_sum = op2.Global(1, data=0.0, np.float32)
  u_max = op2.Global(1, data=0.0, np.float32)

  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))
```



https://github.com/OP2/PyOP2/blob/master/demo/jacobi.py

Example – Jacobi solver

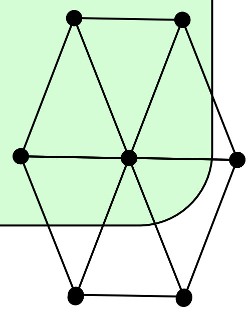
PyOP2 – an active library structured mesh computations

Example – Jacobi solver

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

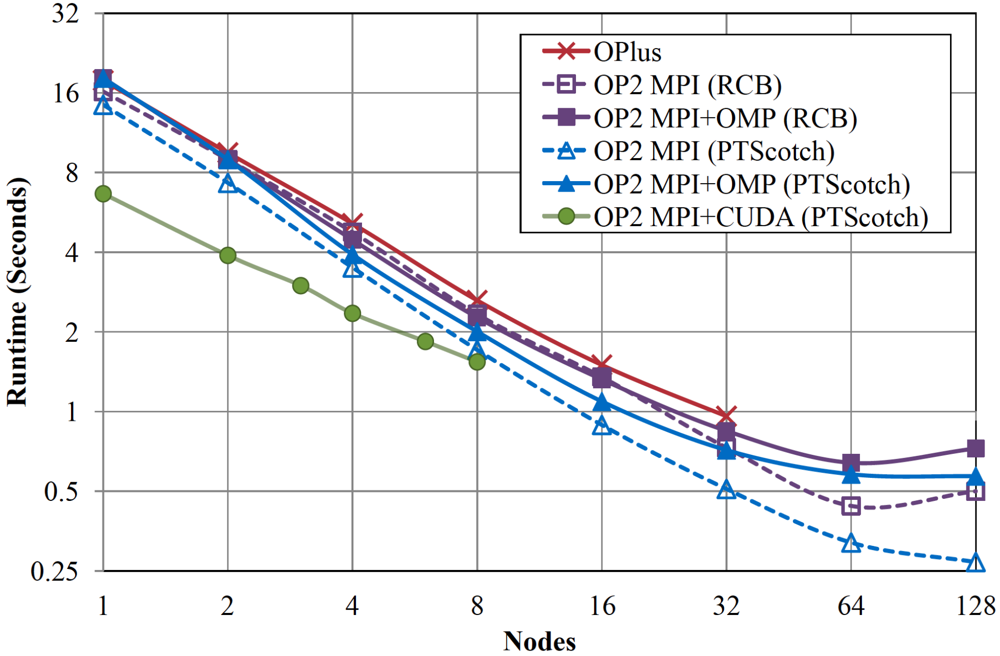
```
void update(float *r, float *du, float *u, float
       *u_sum, float *u_max) {
  *u += *du + alpha * (*r);
  *du = 0.0f;
  *u_sum += (*u) * (*u);
  *u_max = *u_max > *u ? *u_max : *u;
}
```

```
for iter in xrange(0, NITER):
    op2.par_loop(res, edges,
                 p_A(op2.READ),
                 p_u(op2.READ, ppedge[1]),
                 p_du(op2.INC, ppedge[0]),
                 beta(op2.READ))

    u_sum = op2.Global(1, data=0.0, np.float32)
    u_max = op2.Global(1, data=0.0, np.float32)

    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))
```

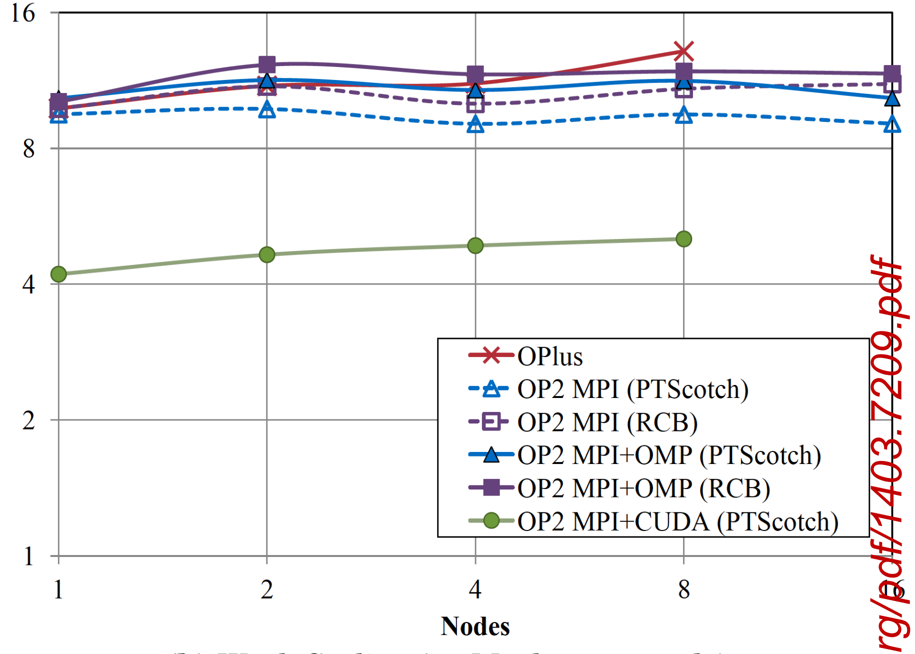


- In this simple example, the kernels are given as C strings
- In most of our work, the kernels are automatically generated
- And passed as ASTs

HYDRA: Full-scale industrial CFD



(a) Strong Scaling (2.5M edges)



(b) Weak Scaling (0.5M edges per node)

- ***Unmodified Fortran OP2 source code exploits inter-node parallelism using MPI, and intra-node parallelism using OpenMP and CUDA***
- ***Application is a proprietary, full-scale, in-production fluids dynamics package***
- ***Developed by Rolls Royce plc and used for simulation of aeroplane engines***

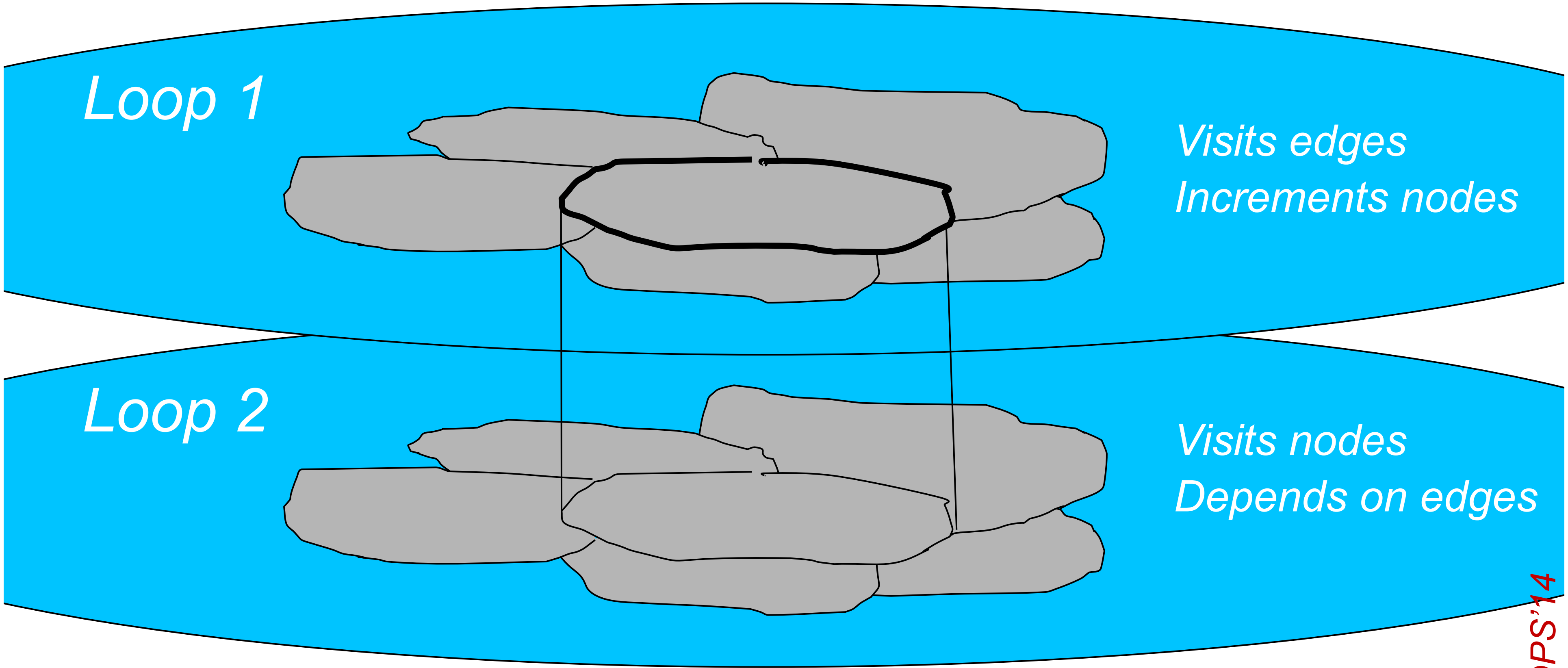
(joint work with Mike Giles, Istvan Reguly, Gihan Mudalige at Oxford)

- ***"Performance portability"***

HECToR (Cray XE6)	Jade (NVIDIA GPU Cluster)
2×16-core AMD Opteron 6276 (Interlagos)2.3GHz	2×Tesla K20m + Intel Xeon E5-1650 3.2GHz
32GB	5GB/GPU (ECC on)
128	8
Cray Gemini	FDR InfiniBand
CLE 3.1.29	Red Hat Linux Enterprise 6.3
Cray MPI 8.1.4	PGI 13.3, ICC 13.0.1, OpenMPI 1.6.4
-O3 -h fp3 -h ipa5	-O2 -xAVX -arch=sm_35 -use_fast_math

Reguly, Mudalige et al, http://arxiv.org/pdf/1403.7209.pdf

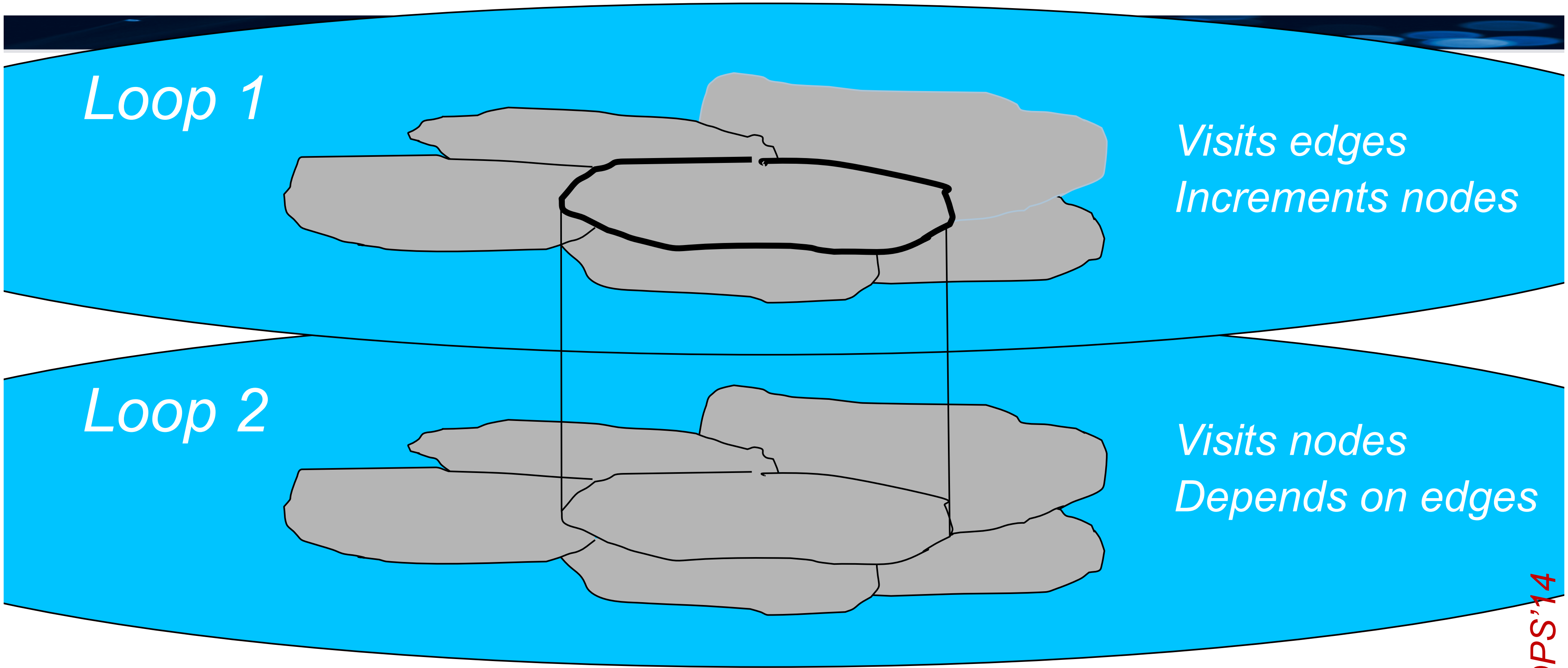
Sparse tiling on an unstructured mesh, for locality



- How can we fuse two loops, when there is a “halo” dependence?
- Ie 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

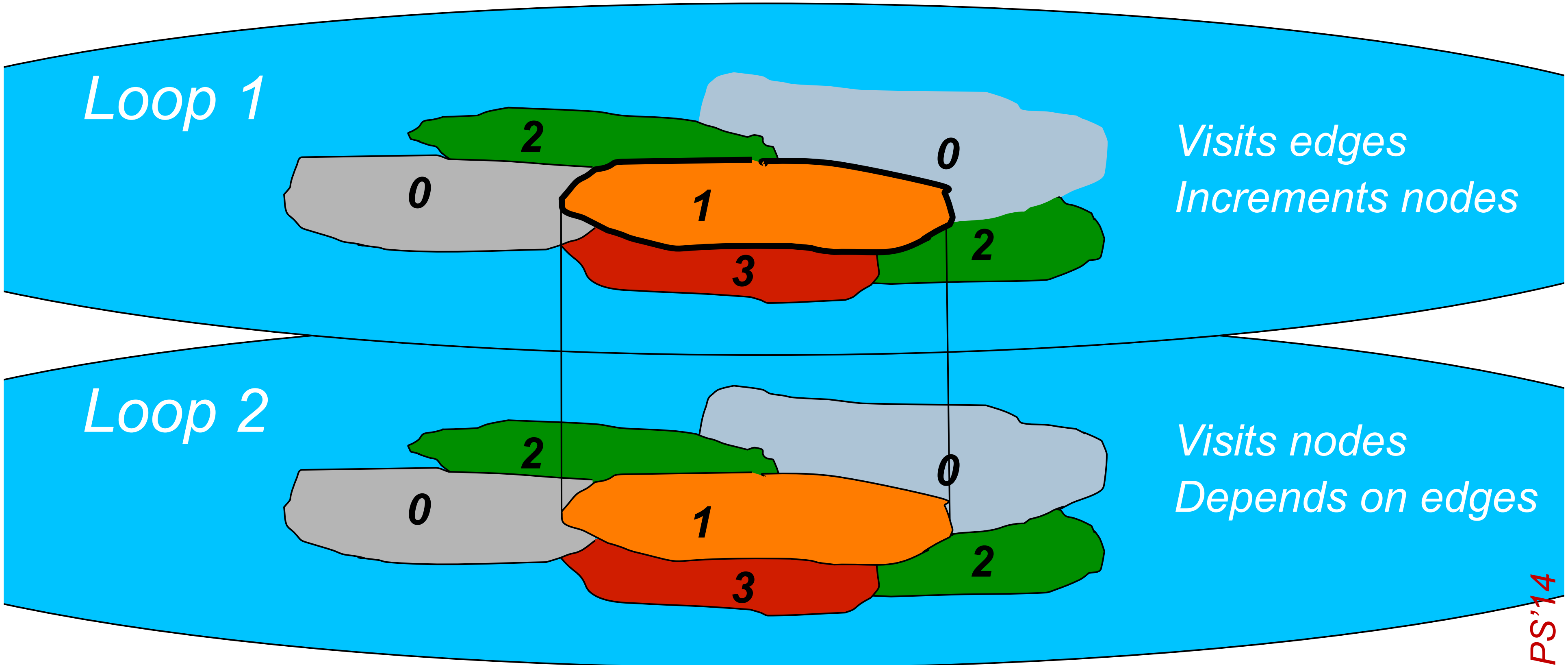
Tiling an unstructured mesh for locality



- Partition the iteration space of loop 1

Strout, Luporini et al, IPDPS'14

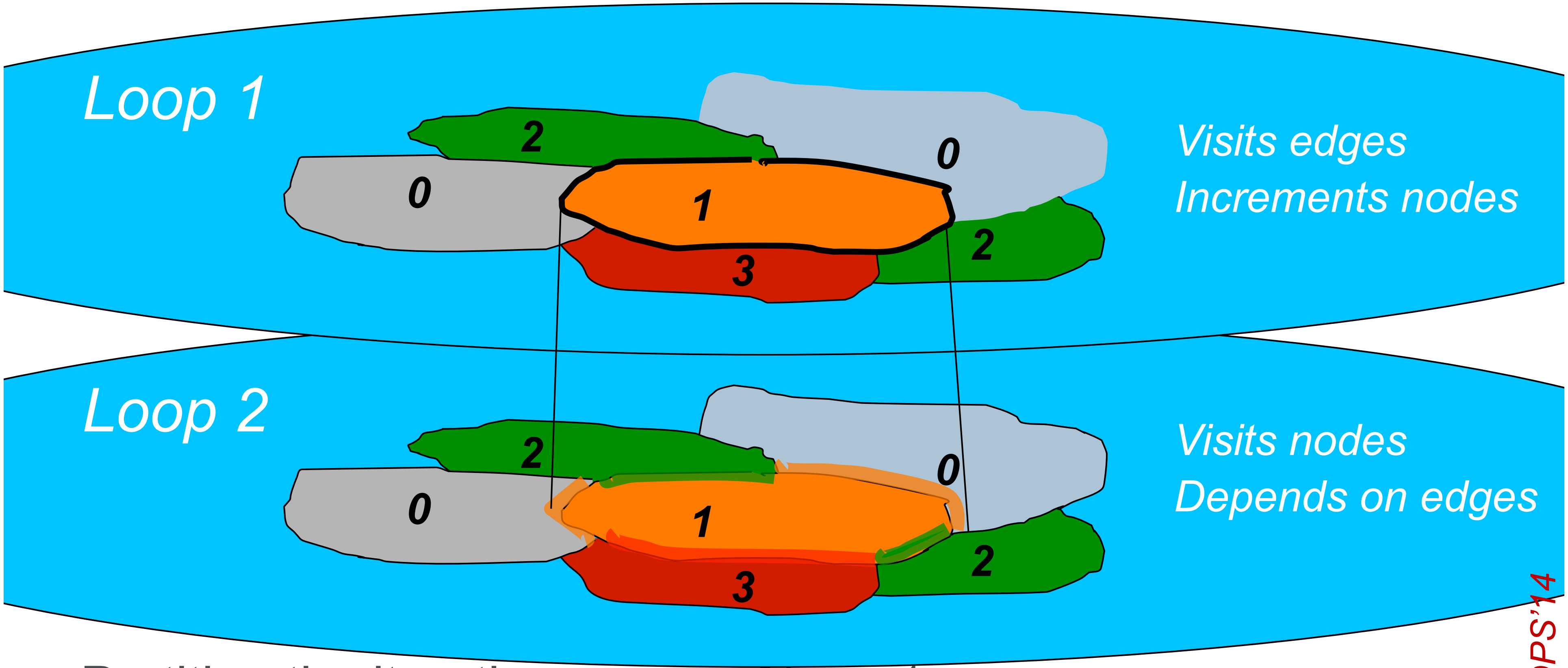
Tiling an unstructured mesh for locality



- Partition the iteration space of loop 1
- Colour the partitions

Strout, Luporini et al, IPDPS'14

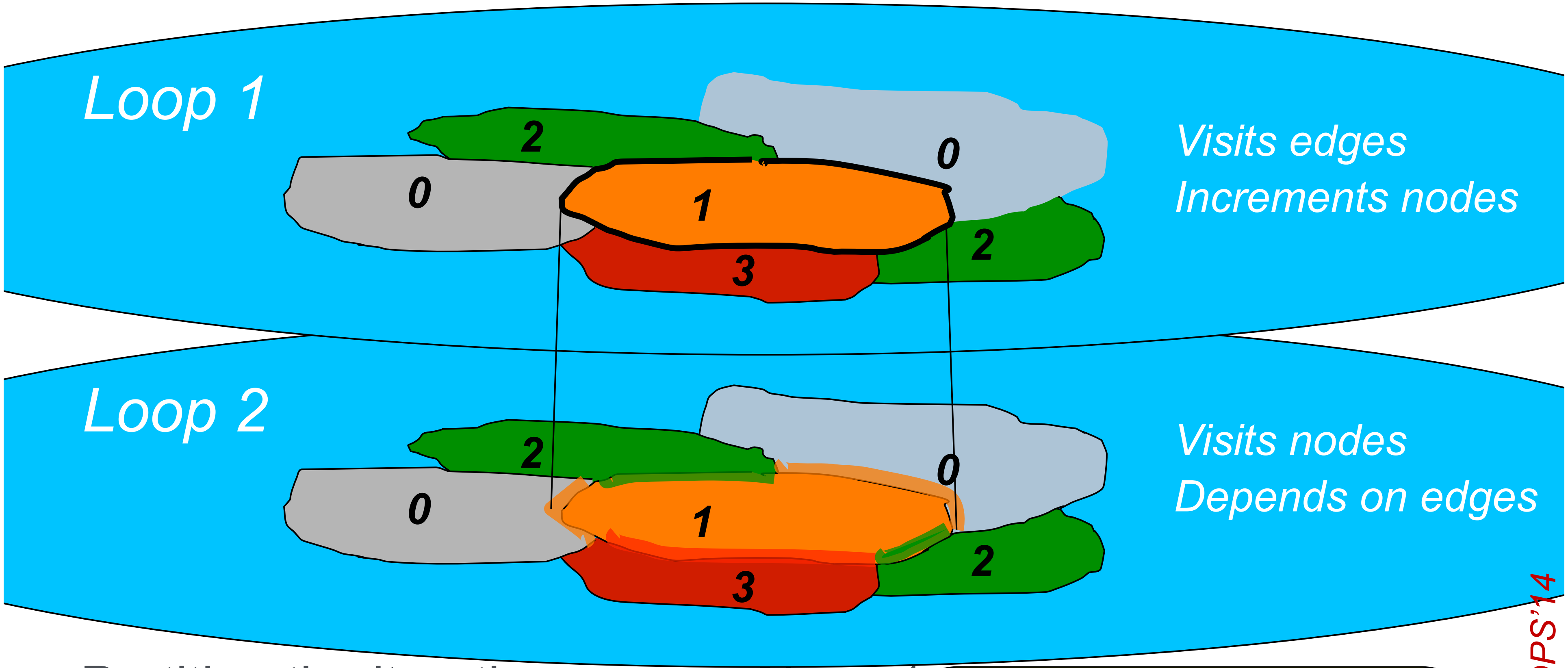
Tiling an unstructured mesh for locality



- Partition the iteration space of loop 1
- Colour the partitions
- 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

Tiling an unstructured mesh for locality

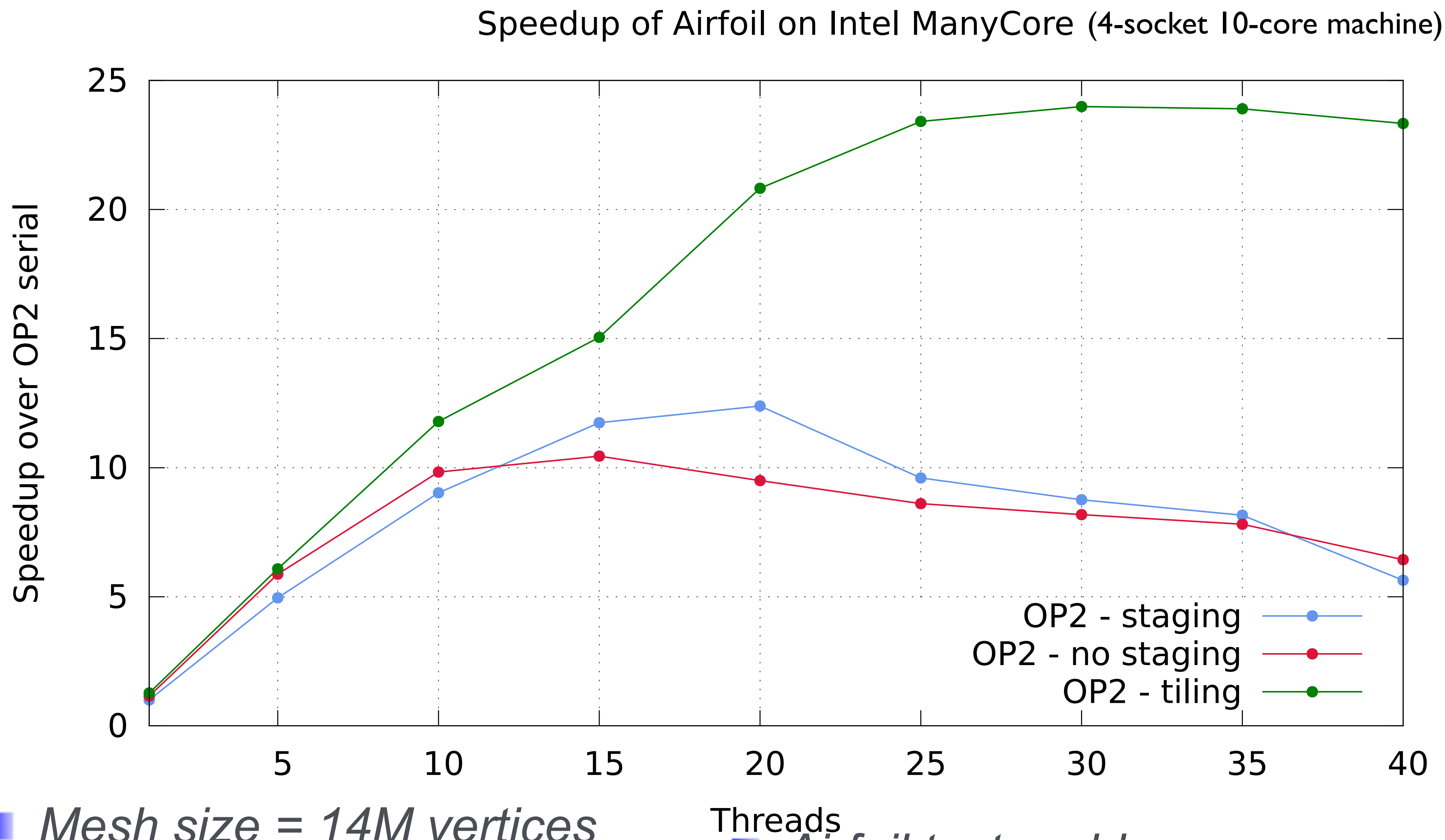


- Partition the iteration space of loop 1
- Colour the partitions
- Project the tiles, using the knowledg data produced by colour n-1
- Thus, the tile coloured #1 grows wh
- And shrinks where it meets colours #2 and #3

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

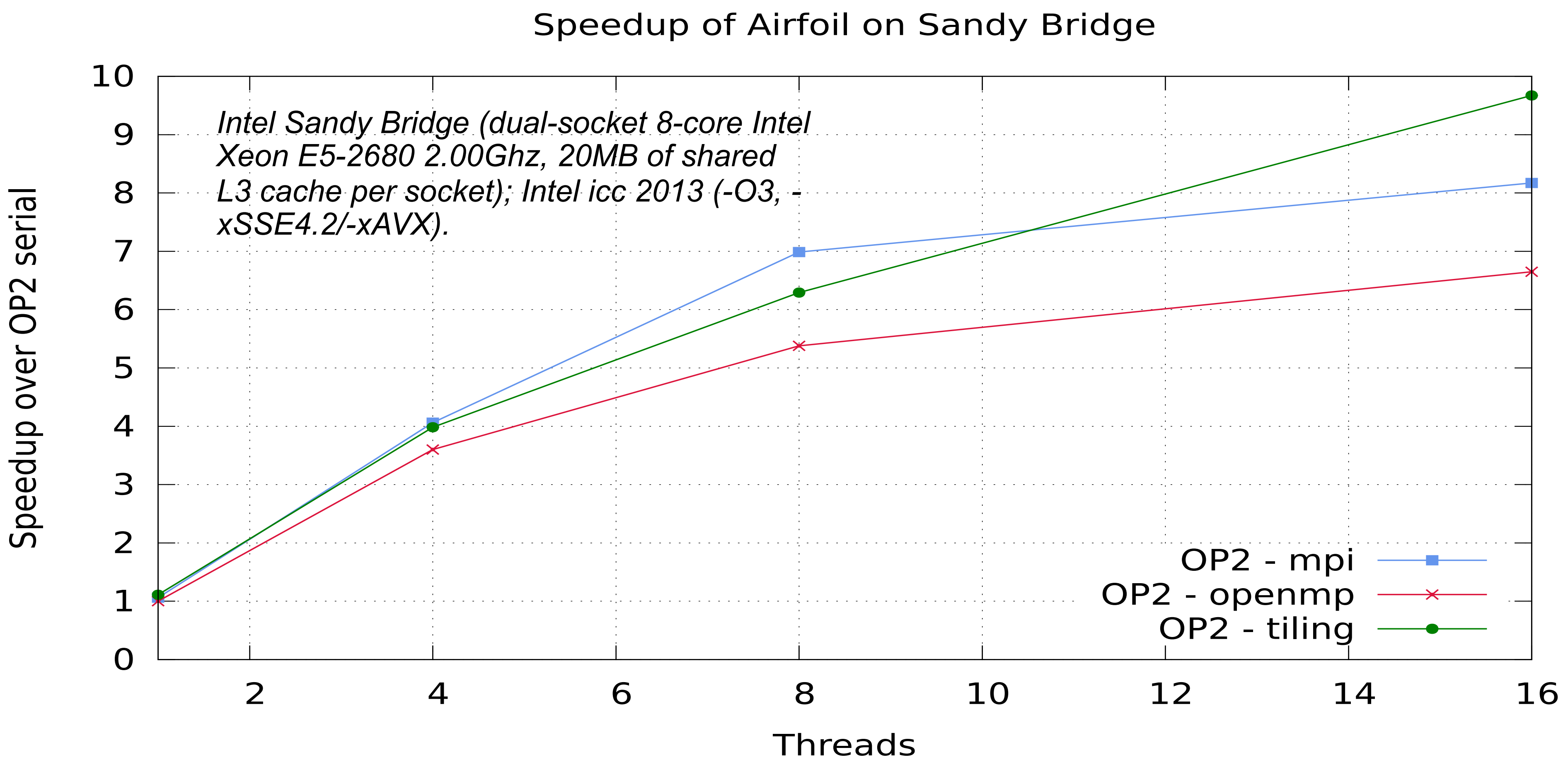
Strout, Luporini et al, IPDPS'14

Extreme results – OP2 loop fusion



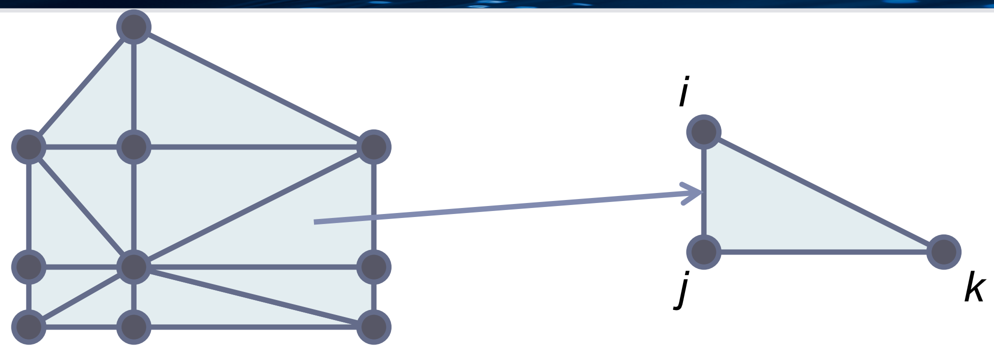
- *Mesh size = 14M vertices*
- *# Loop chain = 2 loops*
- *No inspector/plans overhead*
- *Airfoil test problem*
- *Unstructured-mesh finite-volume*

More realistic results – OP2 loop fusion

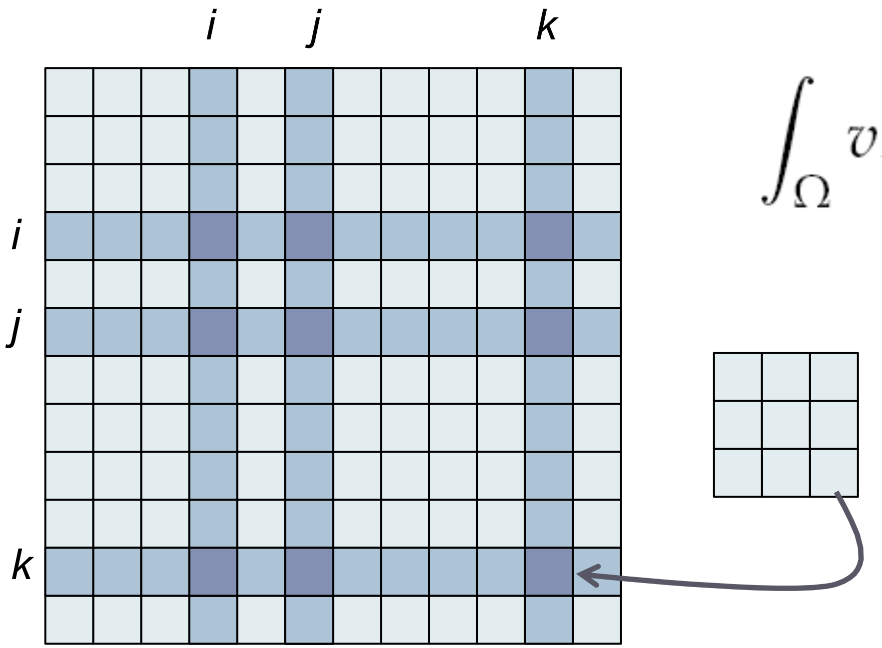


- *Mesh size = 1.5M edges*
- *# Loop chain = 6 loops*
- *No inspector/plans overhead*
- *Airfoil test problem*
- *Unstructured-mesh finite-volume*

The finite element method in outline

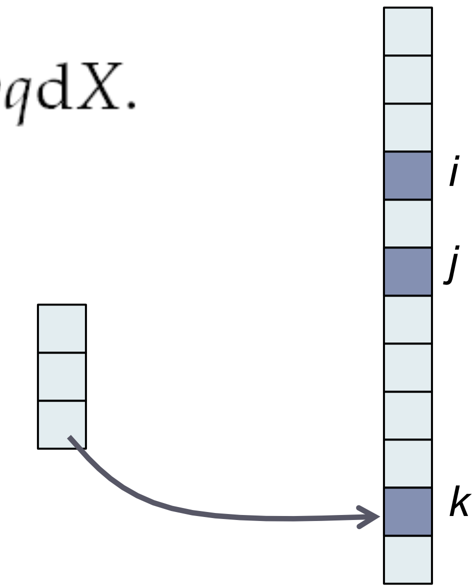



```
do element = 1,N
  assemble(element)
end do
```



$$\int_\Omega vL(u^\delta)\mathrm{d}X = \int_\Omega vq\mathrm{d}X.$$

$$Ax = b$$



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

Multilayered abstractions for FE

- Local assembly:
  - Specified using the FEniCS project's DSL, UFL (the "Unified Form Language")
  - Computes local assembly matrix
  - Key operation is evaluation of expressions over basis function representation of the element
- Mesh traversal:
  - *OP2*
  - *Loops over the mesh*
  - *Key is orchestration of data movement*
- Solver:
  - Interfaces to standard solvers, such as PetSc

The FEniCS project's Unified Form Language (UFL)

A weak form of the shallow water equations

$$\int_\Omega q\nabla\cdot\mathbf{u}\,dV = -\int_{\Gamma E}\mathbf{u}\cdot\mathbf{n}(q^+ - q^-)\,dS$$

$$\int_\Omega \mathbf{v}\cdot\nabla h\,dV = c^2\int_{\Gamma E}(h^+ - h^-)\mathbf{n}\cdot\mathbf{v}\,dS$$

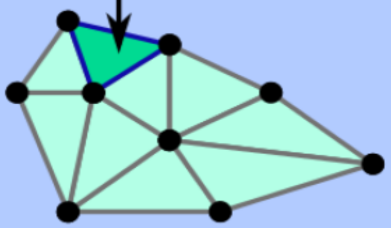
can be represented in UFL as

UFL source

```
V = FunctionSpace(mesh, 'Raviart-Thomas', 1)
H = FunctionSpace(mesh, 'DG', 0)
W = V*H
(v, q) = TestFunctions(W)
(u, h) = TrialFunctions(W)
M_u = inner(v,u)*dx
M_h = q*h*dx
Ct = -inner(avg(u),jump(q,n))*dS
C = c**2*adjoint(Ct)
F = f*inner(v,as_vector([-u[1],u[0]]))*dx
A = assemble(M_u+M_h+0.5*dt*(C-Ct+F))
A_r = M_u+M_h-0.5*dt*(C-Ct+F)
```

Local assembly kernel

```
void Mass(double localTensor[3][3])
{
  const double qw[6] = { ... };
  const double CG1[3][6] = { ... };
  for(int i = 0; i < 3; i++)
    for(int j = 0; j < 3; j++)
      for(int g = 0; g < 6; g++)
        localTensor[i][j]
          += CG1[i][g] * CG1[j][g] * qw[g]);
}
```

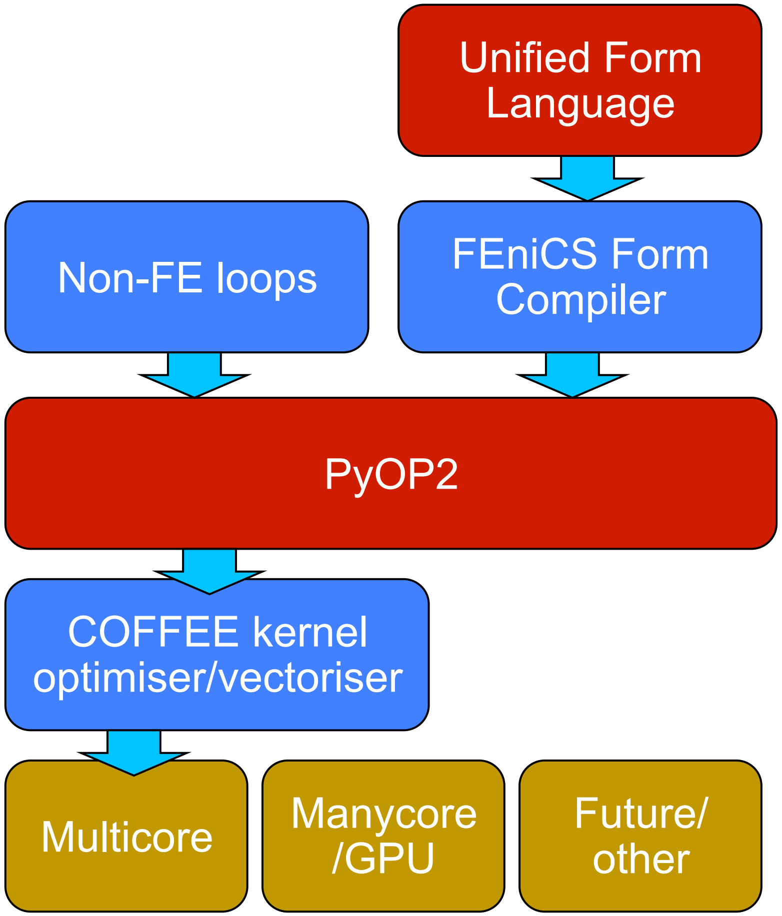


parallel loop
over all grid cells,
in unspecified order,
partitioned

unstructured grid
defined by vertices,
edges and cells

Firedrake: a finite-element framework

- An alternative implementation of the FEniCS language
- Using PyOP2 as an intermediate representation of parallel loops
- All embedded in Python



- The FEniCS project's UFL – DSL for finite element discretisation
- Compiler generates PyOP2 kernels and access descriptors

- Stencil DSL for *unstructured-mesh*
- Explicit *access descriptors* characterise access footprint of kernels
- Runtime code generation

- **The advection-diffusion problem:** $$\frac{\partial T}{\partial t} = \underbrace{D\nabla^2 T}_{\text{Diffusion}} - \underbrace{\mathbf{u}\cdot\nabla T}_{\text{Advection}}$$
- Weak form:

$$\int_\Omega q\frac{\partial T}{\partial t}\,\mathrm{d}X = \int_{\partial\Omega} q(\nabla T - \mathbf{u}T)\cdot\mathbf{n}\,\mathrm{d}s - \int_\Omega \nabla q\cdot\nabla T\,\mathrm{d}X + \int_\Omega \nabla q\cdot\mathbf{u}T\,\mathrm{d}X$$

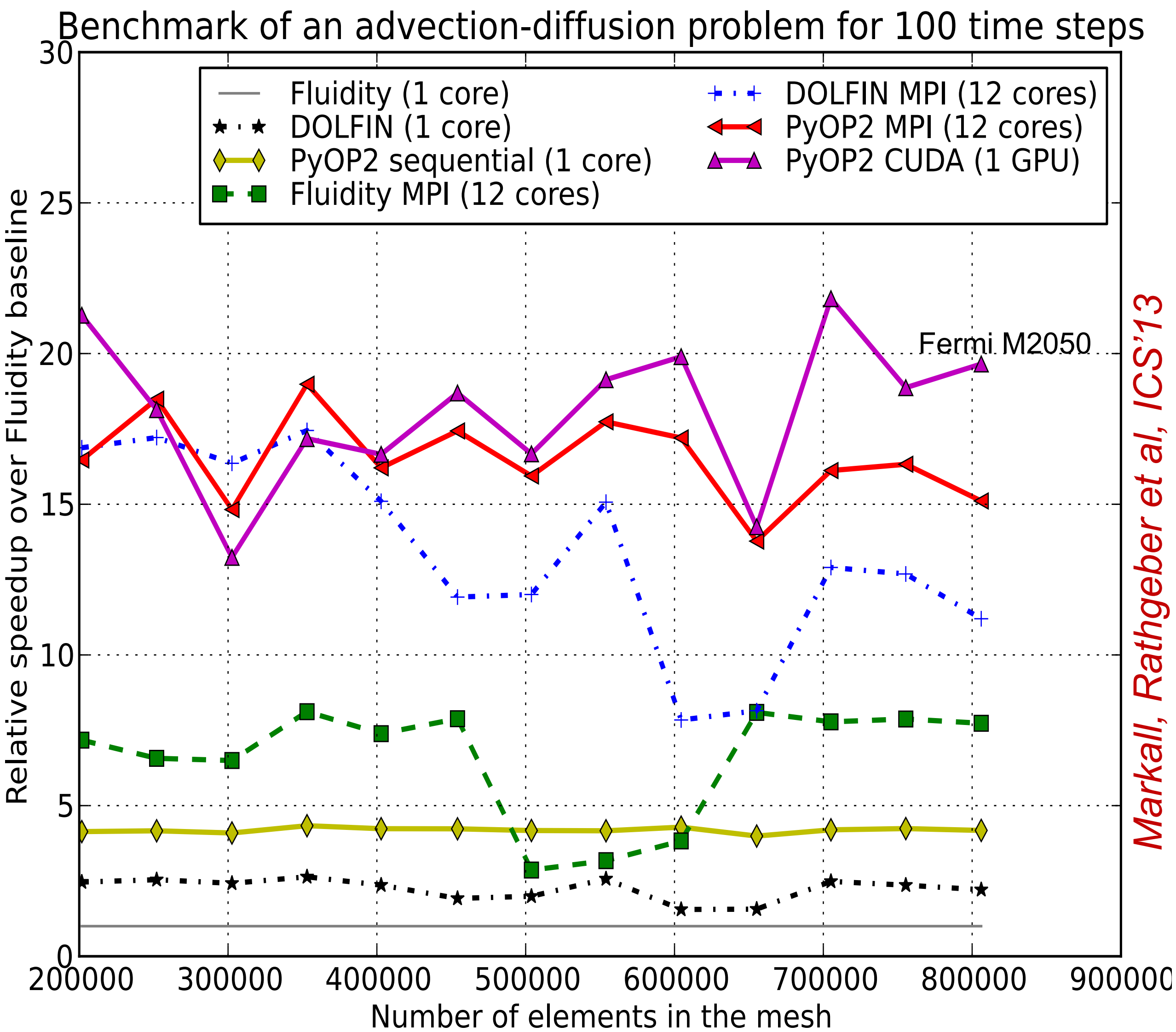
- This is the entire specification for a solver for an advection-diffusion test problem
- Same model implemented in FEniCS/ Dolfin, and also in Fluidity – hand-coded Fortran

```
t=state.scalar_fields["Tracer"]        # Extract fields
u=state.vector_fields["Velocity"]      # from Fluidity

p=TrialFunction(t)                     # Setup test and
q=TestFunction(t)                      # trial functions

M=p*q*dx                               # Mass matrix
d=-dt*dfsvty*dot(grad(q),grad(p))*dx # Diffusion term
D=M-0.5*d                              # Diffusion matrix

adv = (q*t+dt*dot(grad(q),u)*t)*dx     # Advection RHS
diff = action(M+0.5*d,t)               # Diffusion RHS

solve(M == adv, t)                     # Solve advection
solve(D == diff, t)                    # Solve diffusion
```

Imperial College London

Firedrake – single-node performance

- Here we compare performance against two production codes solving the same problem on the same mesh:
  - Fluidity: Fortran/ C++
  - DOLFIN: the FEniCS project's implementation of UFL



Markall, Rathgeber et al, ICS'13

- Graph shows speedup over Fluidity on one core of a 12-core Westmere node

Imperial College London

Firedrake – larger core counts

- Phase separation of the two components of a binary fluid
- Fourth-order parabolic time-dependent non-linear Cahn-Hilliard equation

- GMRES solver with a fieldsplit preconditioner using a lower Schur complement factorisation
- HYPRE Boomeramg
- algebraic multigrid preconditioner
- Example is in the demo suite

- 8M DOF mesh
- Ten timesteps

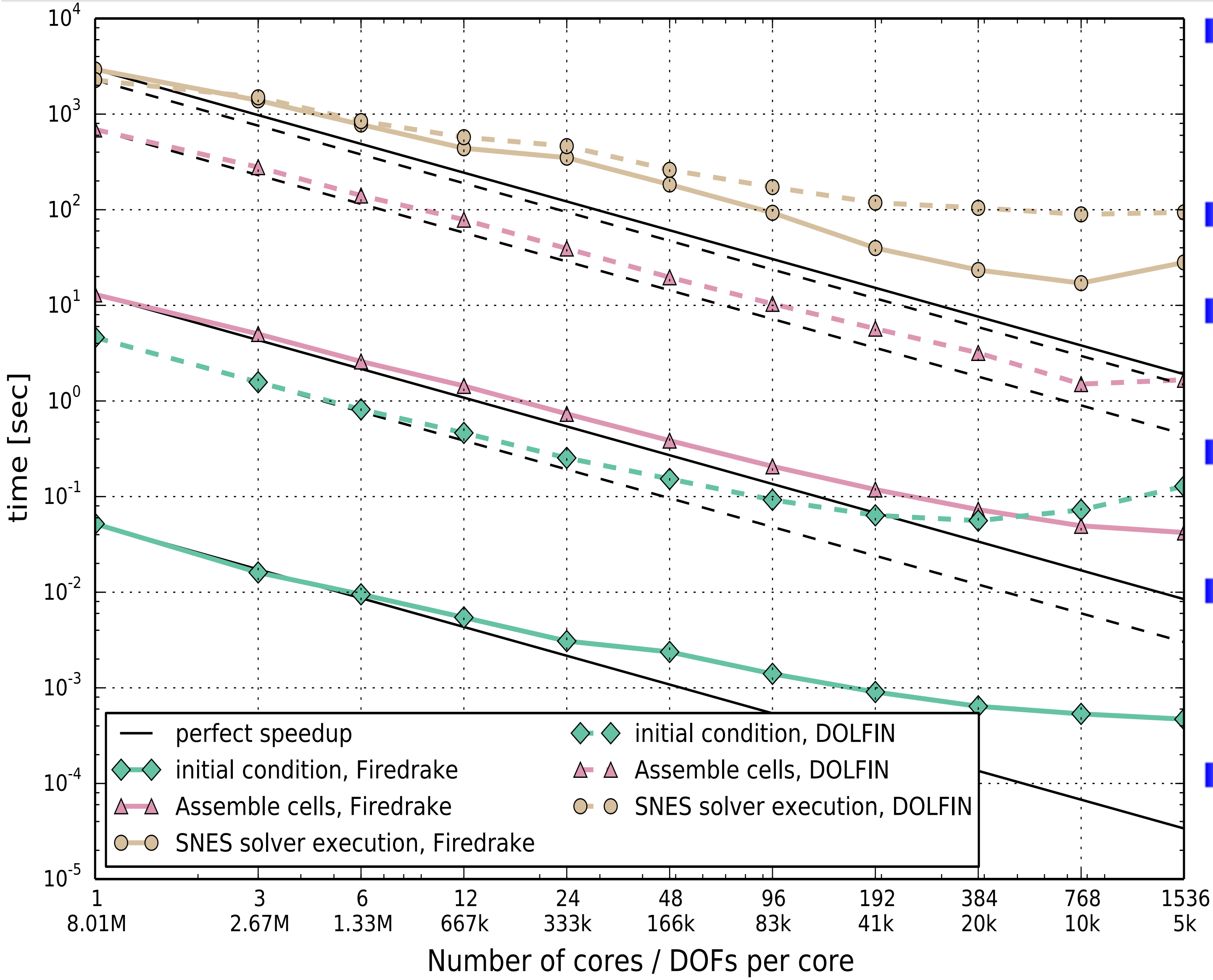
- Up to 1536 cores
- Down to 5K DOFs per core

- Running on ARCHER, a Cray XC30
- Compute nodes contain two 2.7 GHz, 12-core E5-2697 v2 (Ivy Bridge) processors and 64GB of RAM in two 32GB NUMA regions.

- Firedrake and PETSc were compiled with version 4.8.2 of the GNU Compilers and Cray MPICH2 6.3.1 with the asynchronous progress feature enabled was used for parallel runs. Generated code was compiled with the -O3 -mavx flags. The software revisions used were Firedrake revision c8ed154 from September 25 2014, PyOP2 revision f67fd39 from September 24 2014 with PETSc revision 42857b6 from August 21 2014 and DOLFIN revision 30bbd31 from August 22 2014 with PETSc revision d7ebadd from August 13 2014.
- Generated code is compiledwith -O3 -fno-tree-vectorize in Firedrake and -O3 -ffast-math -march=native in DOLFIN

http://fenicsproject.org/documentation/dolfin/1.4.0/python/demo/documented/cahn-hilliard/ python/documentation.html

Firedrake – Scaling



- Both Firedrake and Dolfin scale down to 10K DOFs/core
- But Firedrake is much faster:
- Better implementation of mixed spaces
- Residuals and Jacobians are cached
- Inlining and loop nest optimisations/ vectorization
- Solver is faster thanks to nested matrix handling of mixed spaces and Schur complement

http://fenicsproject.org/documentation/dolfin/1.4.0/python/demo/documented/cahn-hilliard/ python/documentation.html

Optimisation of kernels

```
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
- Essentially computing (for example) integrals of flows across facets
- These are combined to form a global system of simultaneous equations capturing the discretised conservation laws expressed by the PDE

Optimisation of kernels

```
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

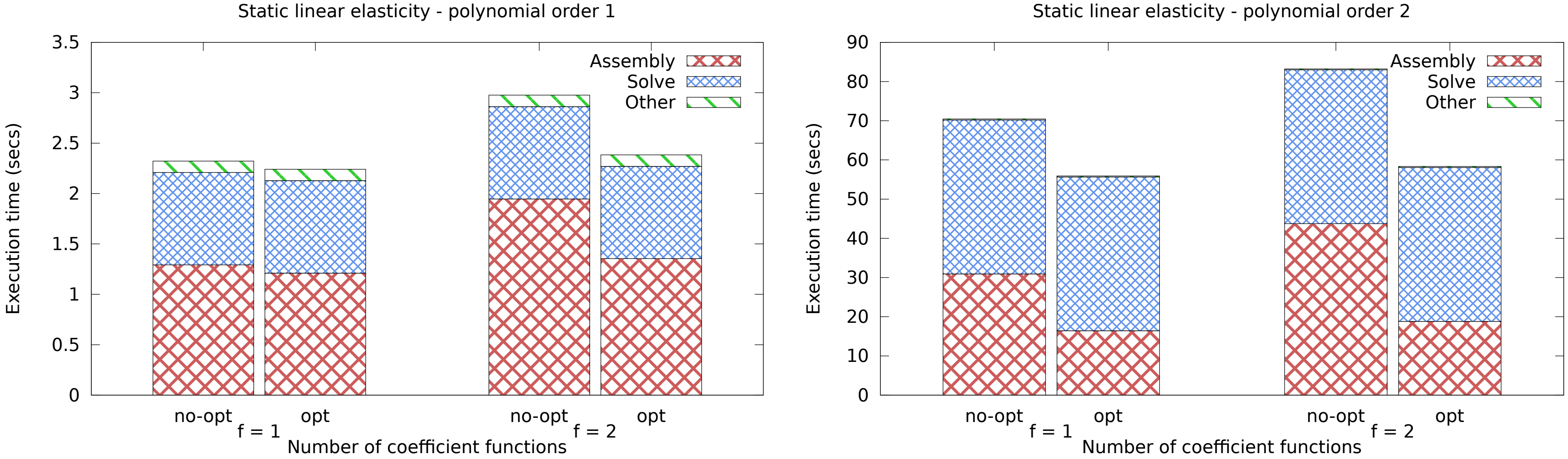
Optimisation of kernels

```
void burgers(double A[12][12], double **coords, double **w) {
 // K, det = Compute Jacobian (coords)

 static const double W[5] = {...}
 static const double X1_D001[5][12] = {{...}}
 static const double X2_D001[5][12] = {{...}}
 //11 other basis functions definitions.
 ...
 for (int i = 0; i<5; i++) {
  double F0 = 0.0;
  //10 other declarations (F1, F2,...)
  ...
  for (int r = 0; r<12; r++) {
   F0 += (w[r][0]*X1_D100[i][r]);
   //10 analogous statements (F1, F2, ...)
   ...
  }
  for (int j = 0; j<12; j++)
   for (int k = 0; k<12; k++)
    A[j][k] += (..(K5*F9)+(K8*F10))*Y1[i][j])+
     +(((K0*X1_D100[i][k])+(K3*X1_D010[i][k])+(K6*X1_D001[i][k]))*Y2[i][j]))*F11)+
     +(..((K2*X2_D100[i][k])+...+(K8*X2_D001[i][k]))*((K2*X2_D100[i][j])+...+(K8*X2_D001[i][j]))..
     + <roughly a hundred sum/muls go here>)..)*
     *det*W[i]);
 }
}
```

- Local assembly code generated by Firedrake for a Burgers problem on a 3D tetrahedral mesh using Lagrange p = 1 elements
- Somewhat more complicated!
- Examples like this motivate more complex transformations
- Including loop fission

Imperial College London

Performance impact



- Fairly serious, realistic example: static linear elasticity, p=2 tetrahedral mesh, 196608 elements
- Including both assembly time and solve time
- Single core of Intel Sandy Bridge
- Compared with Firedrake loop nest compiled with Intel's icc compiler version 13.1
- At low p, matrix insertion overheads dominate assembly time
- At higher p, and with more coefficient functions (f=2), we get up to 1.47x overall application speedup

Conclusions: PyOP2 layer

- The key idea in OP2/PyOP2 is ***access descriptors***
- OP2's access descriptors are ***declarative specifications*** of how each loop iteration is connected to the abstract mesh
- The kernels do not access the mesh
- The implementation is responsible for connecting the kernel to the data
- The implementation is free to select layout, stage data, schedule loops
  - We can map from data to iterations
- **What *would* a programming abstraction for data locality look like?**

Imperial College London

Conclusions: Firedrake layer

- Dramatically raised level of abstraction
- But we still can match or exceed hand-coded, in-production code
- Costs of abstraction are eliminated by dynamic generation of code specialised to context
- Domain-specific optimisations can yield big speedups over the best available general-purpose compilers

- **The real payoff lies in supporting the users in navigating freely to the best way to model their problem**
- **How can the *barriers to adoption* of DSLs be overcome?**

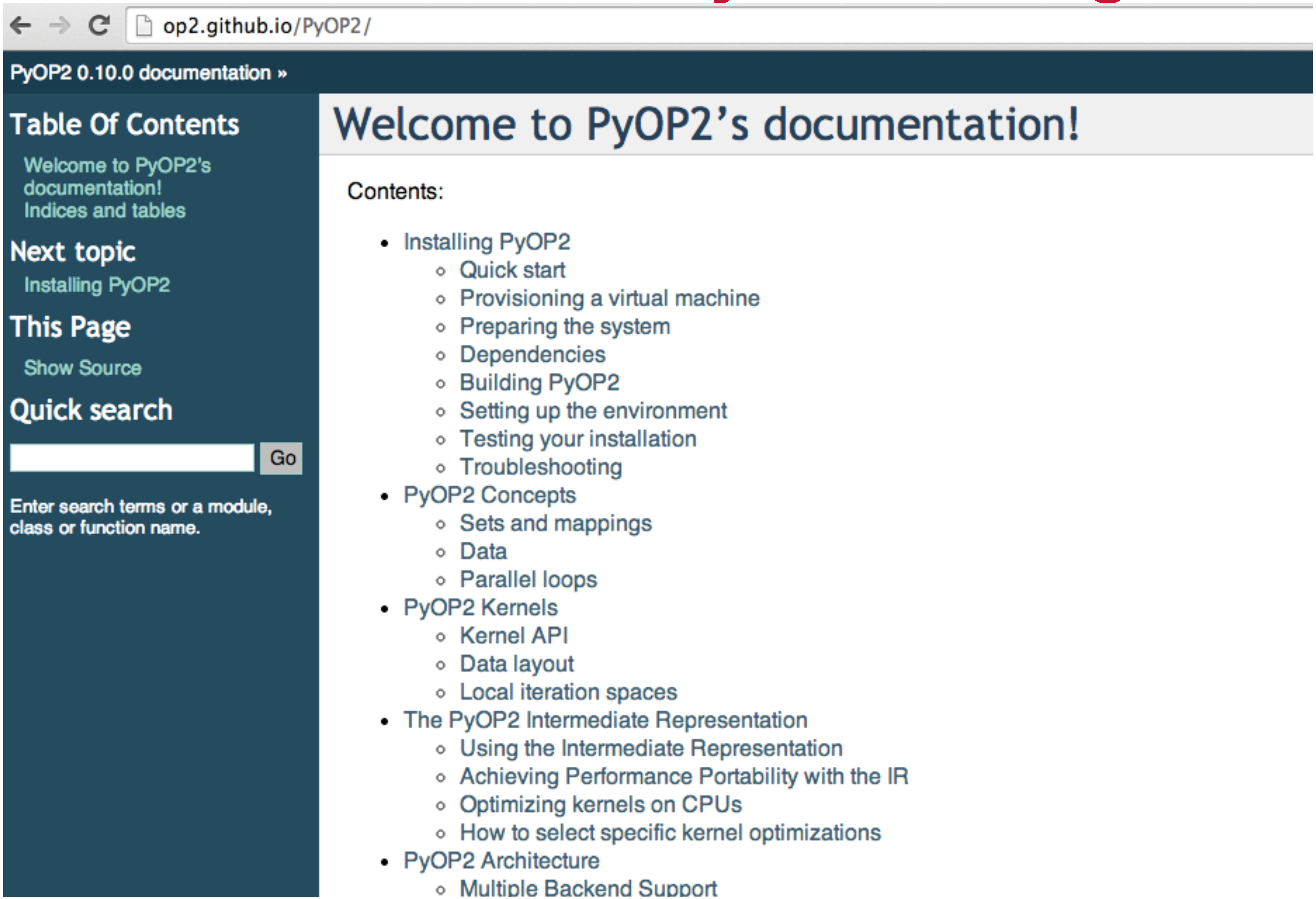
Acknowledgements

Partly funded by

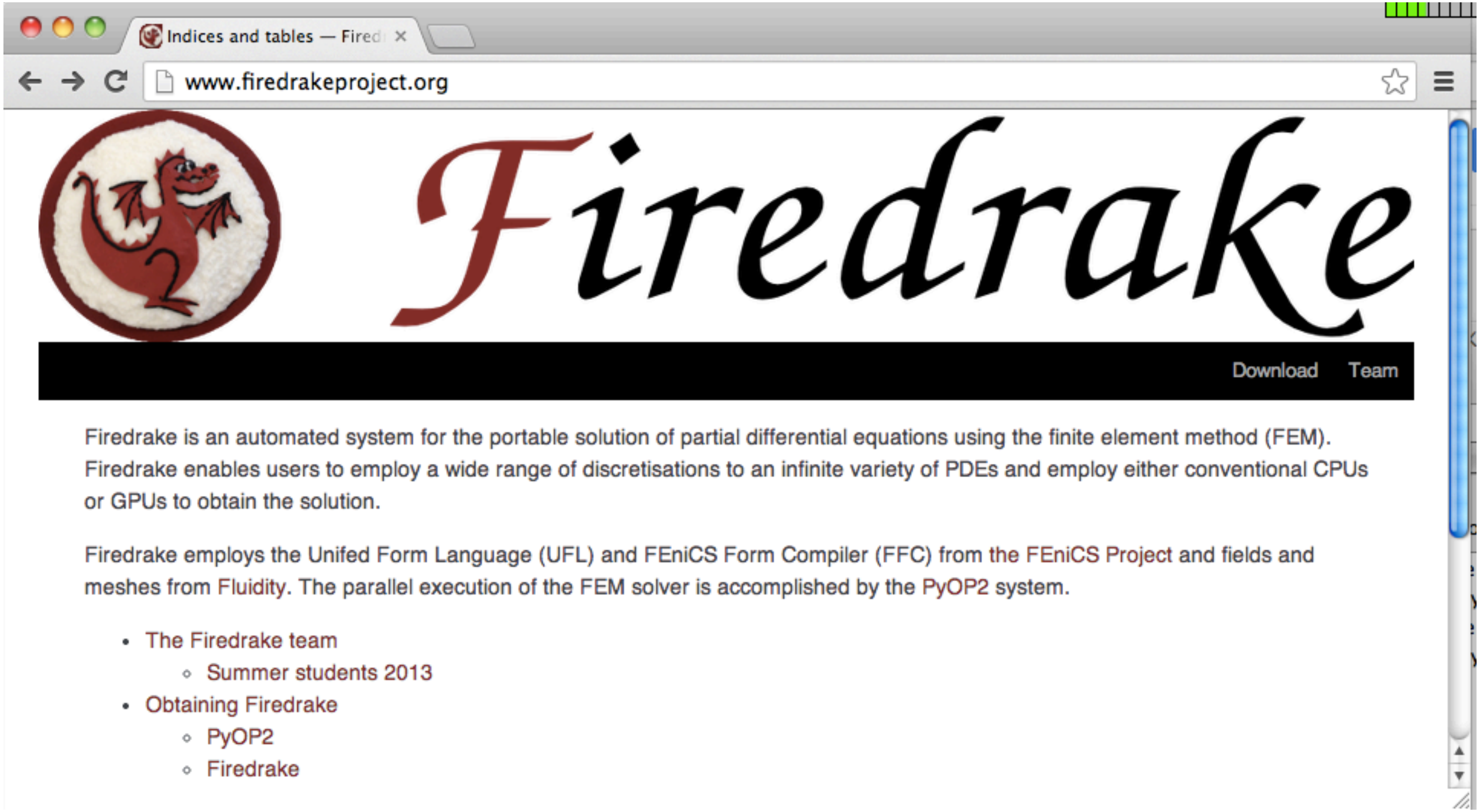
- NERC Doctoral Training Grant (NE/G523512/1)
- EPSRC “MAPDES” project (EP/I00677X/1)
- EPSRC “PSL” project (EP/I006761/1)
- Rolls Royce and the TSB through the SILOET programme
- EPSRC “PAMELA” Programme Grant (EP/K008730/1)
- EPSRC “PRISM” Platform Grant (EP/I006761/1)
- EPSRC “Custom Computing” Platform Grant (EP/I012036/1)
- AMD, Codeplay, Maxeler Technologies

- Code:
  - http://www.firedrakeproject.org/
  - http://op2.github.io/PyOP2/

PyOP2 is on github



Firedrake is on github



Imperial College London

The FEniCS project... The book