Imperial College London

Towards Cross-Domain Domain-Specific Compiler Architecture

Paul Kelly

Group Leader, Software Performance Optimisation

Department of Computing

Imperial College London

Joint work with David Ham (Imperial Maths), Gerard Gorman (Imperial ESE), Lawrence Mitchell (now with NVIDIA), Sophia Vorderwuelbecke, George Bisbas, Edward Stow (Imperial), Fabio Luporini (Devito Codes Ltd), 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 NVIDIA), Tianjiao Sun (now at Cerebras), Thomas Gibson (NCSA Illinois), Kaushik Kulkarni (UIUC), Andreas Klockner (UIUC), Tobias Grosser, Michel Steuwer (University of Edinburgh), Larisa Stolzfus, Amrey Krause, Nick Brown (EPCC), Navjot Kukreja (University of Liverpool)

And many others....

Imperial College London

Who am I and what do I do?

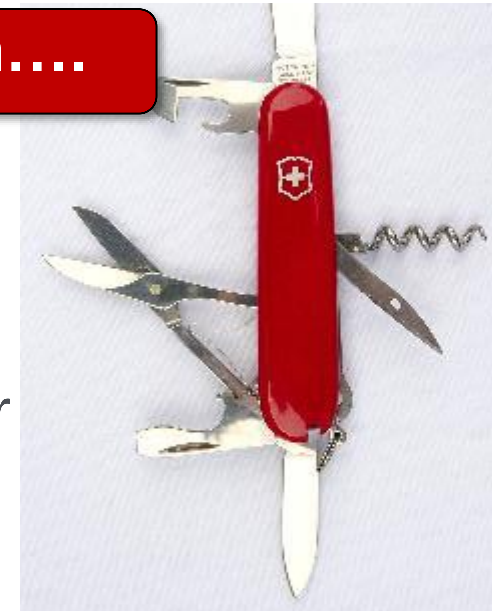
- I've worked on a lot of things….
  - GPUs, FPGAs, vector/matrix ISAs, cache coherency, large-scale SIMD, precision optimisation…
- I have worked on general-purpose compilers
  - Notably pointer analysis
  - adopted into GCC, go compiler
    - (actually the work of my PhD student David Pearce)
- But the benefits were incremental
- Meanwhile I engaged with applications specialists
  - Who know they have major performance optimisation opportunities
  - So I got interested in automating domain-specific optimisations

File:Victorinox Swiss Army Knife - Climber (15554551505).jpg - Wikimedia Commons

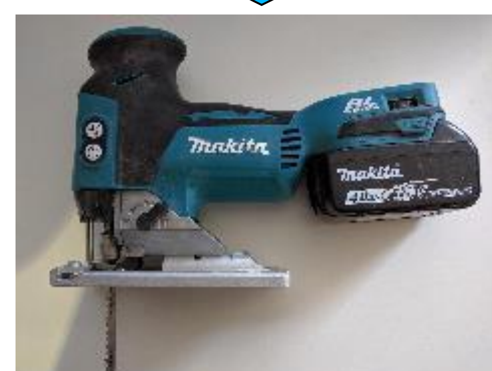
Power tools for performance programming

By capturing domain-specific representation….

- We can deliver domain-specific **optimisations**
- We collect and automate all the performance techniques that are known for a **family of problems**



- If we get it right…. we get
  - **Productivity** – by generating low-level code from a high-level specification
  - **Performance** – by automating optimisations
  - **Performance portability** – with multiple back-ends



https://commons.wikimedia.org/wiki/File:Victorinox_Swiss_Army_Knife_-_Climber_(15554551505).jpg
https://commons.wikimedia.org/wiki/File:Makita_DJV181.jpg

Imperial College London

The work of my research group

Automating domain-specific performance optimisations

Exploiting higher-level language to get better performance than low level code

Technologies

- Vectorisation, parametric polyhedral tiling
- Tiling for unstructured-mesh stencils
- Lazy, data-driven compute-communicate
- Runtime code generation
- Multicore graph worklists
- Tensor contractions
- Generalised loop-invariant code motion
- Sparsity in Fourier transforms
- Functional Variational Inference
- Search-based optimisation
- Processor/accelerator microarchitecture, co-design
- MLIR

Contexts

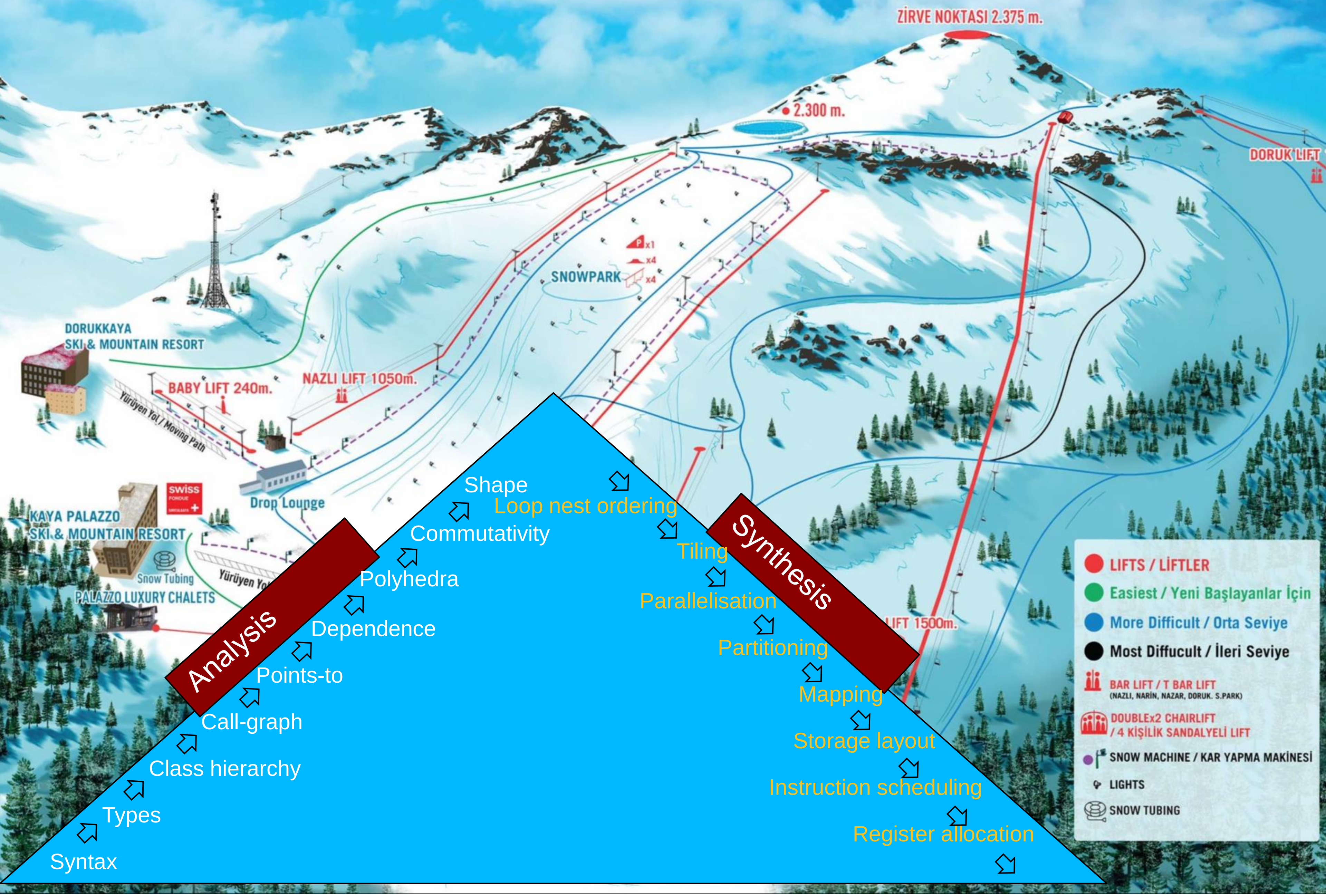
- Finite-volume CFD
- Finite-element
- Finite-difference
- Real-time 3D scene understanding
- Adaptive-mesh CFD
- Contour trees, Reeb graphs
- Unsteady CFD - higher-order flux-reconstruction
- Ab-initio computational chemistry (ONETEP)
- Gaussian belief propagation
- Uncertainty in DNNs
- Near-camera processing
- Quantum computing

Projects

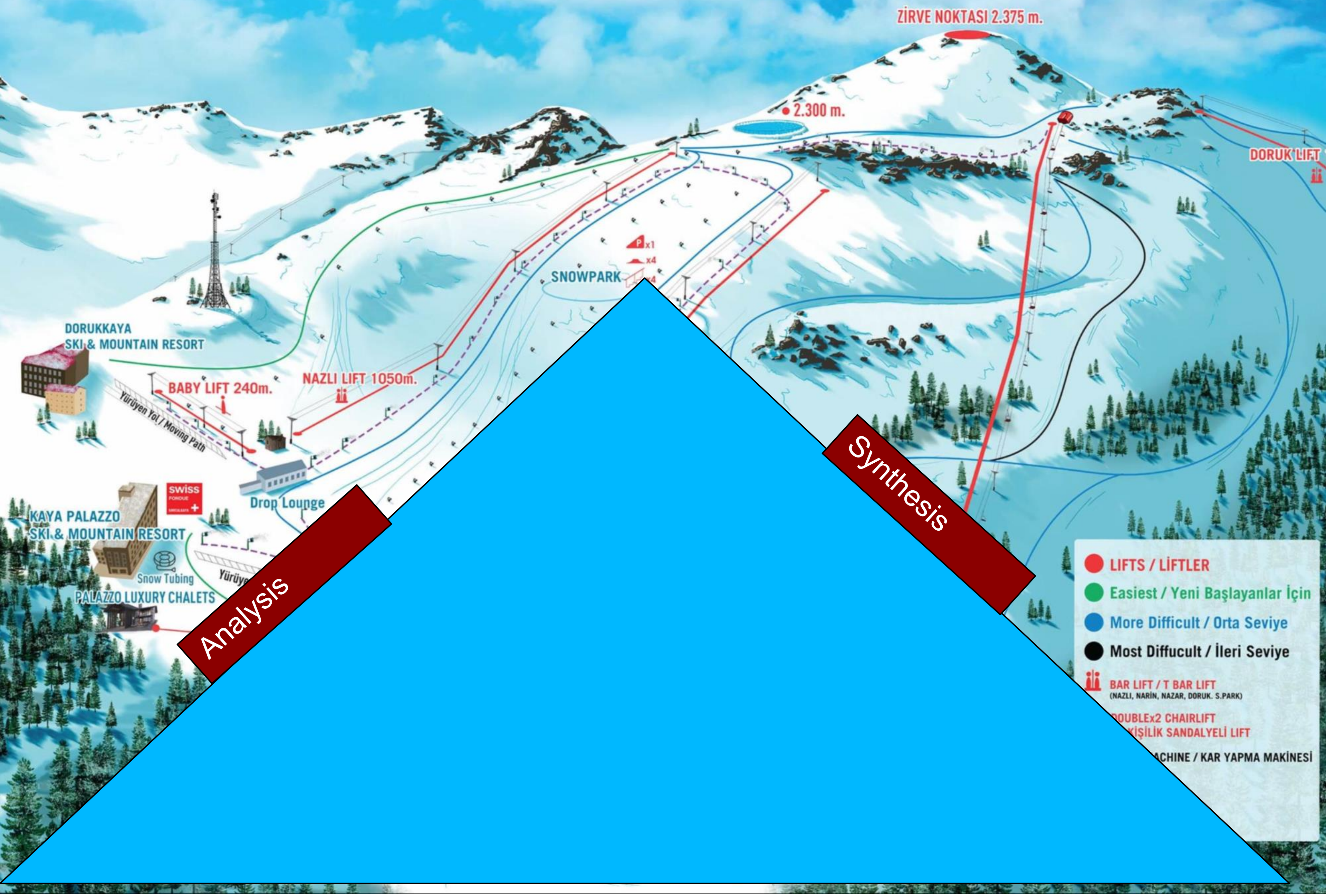
- **PyOP2/OP2** Unstructured-mesh stencils
- **Firedrake** Finite-element
- **Devito:** finite difference
- **SLAMBench:** 3D vision, dense SLAM
- **SuperEight:** octree SLAM
- **PRAgMaTIc:** Unstructured mesh adaptation
- **GiMMiK:** small matrix multiply
- **TINTL:** Fourier interpolation
- **Hypermapper:** design optimisation
- **RobotWeb:** distributed localisation

Applications

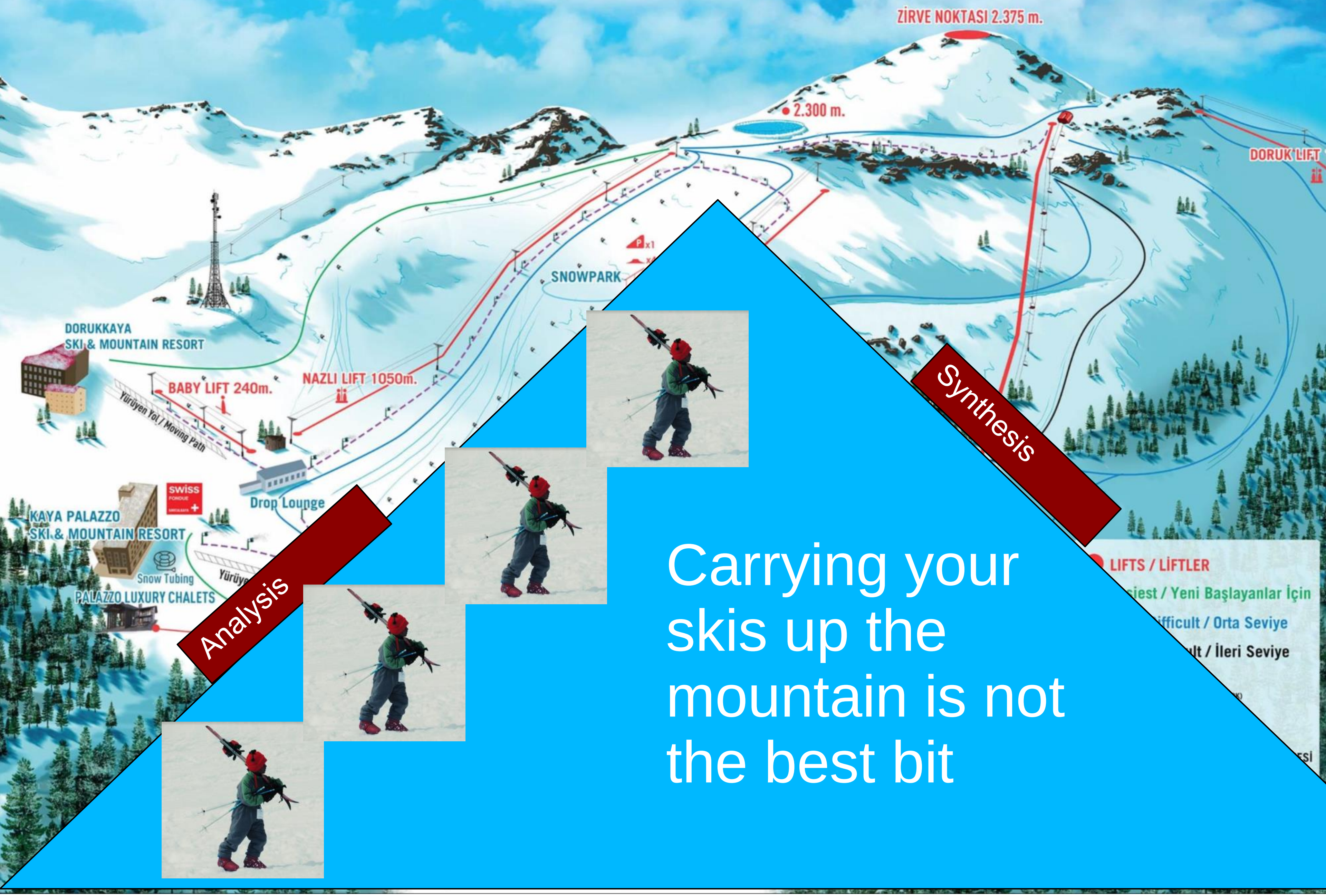
- Aeroengine turbo-machinery
- Weather and climate
- Glaciers
- Domestic robotics, augmented reality
- Tidal turbine placement
- Formula-1, UAVs, buildings
- Solar energy, drug design
- Medical imaging



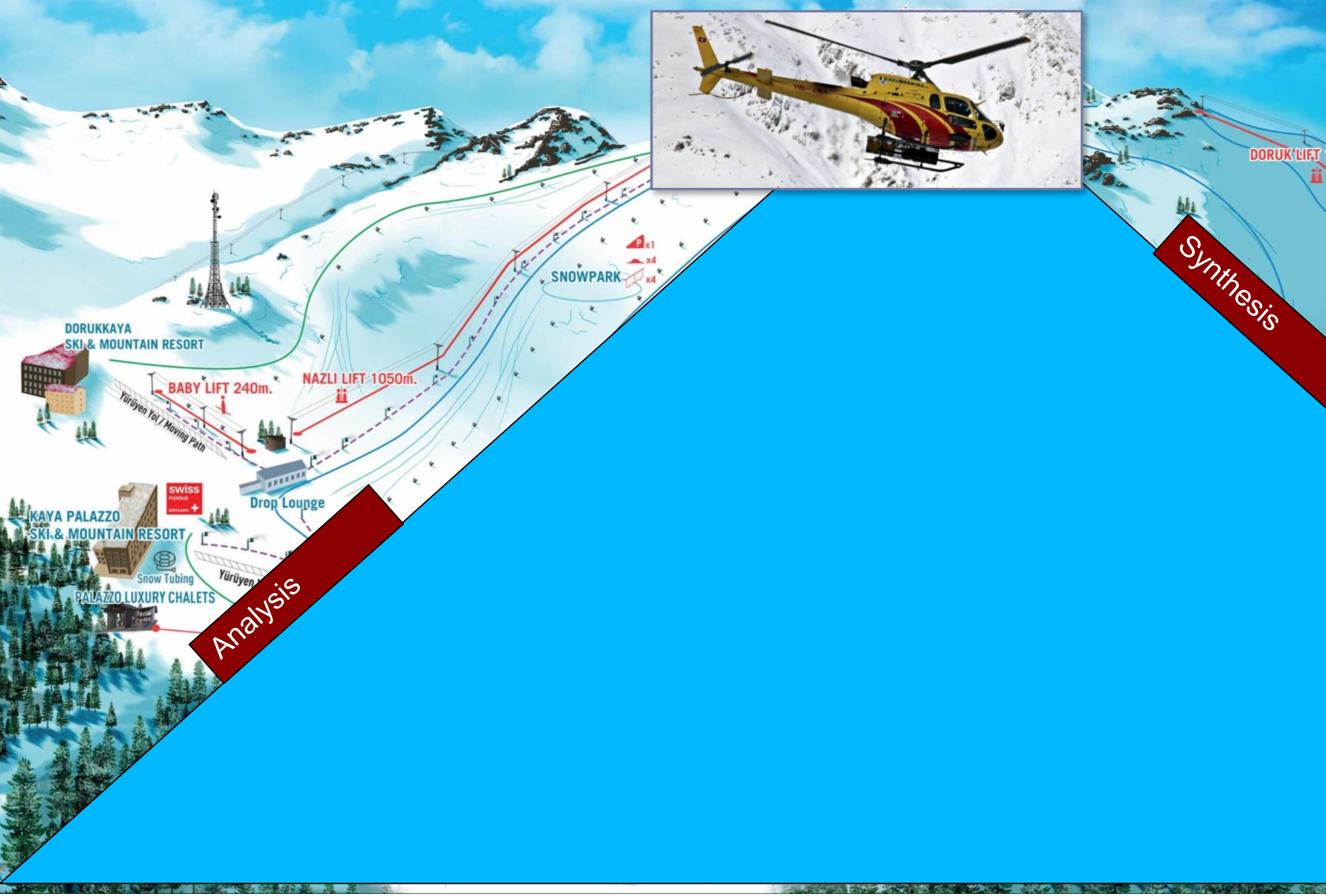
Compilation is like skiing



Compilation is like skiing



Compilation is like skiing



Using a DSL is like helicopter skiing

Plan

- Compiling is like skiing
  - Analysis is an uphill struggle
- "Turing Tax"
  - The price you pay for running on a general-purpose computer rather than a specialised one
- What do we call…
  - ***The price you pay for using a general-purpose programming language rather than a DSL?***
- This talk:
  - DSLs really can deliver
  - DSL compiler architecture: how do DSLs win?
  - **Why we have to make the DSL ecosystem work!**

(For more on the Turing Tax see The von Neumann Bottleneck and the Turing Tax - YouTube)

Compilation is like skiing

Example DSL:

Tensor Decompositi... The Conversation: I... Paul Kelly (paulhjkel... MS mymake The power of listeni... Tubes Computing | The Br... » Other bookmarks

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?

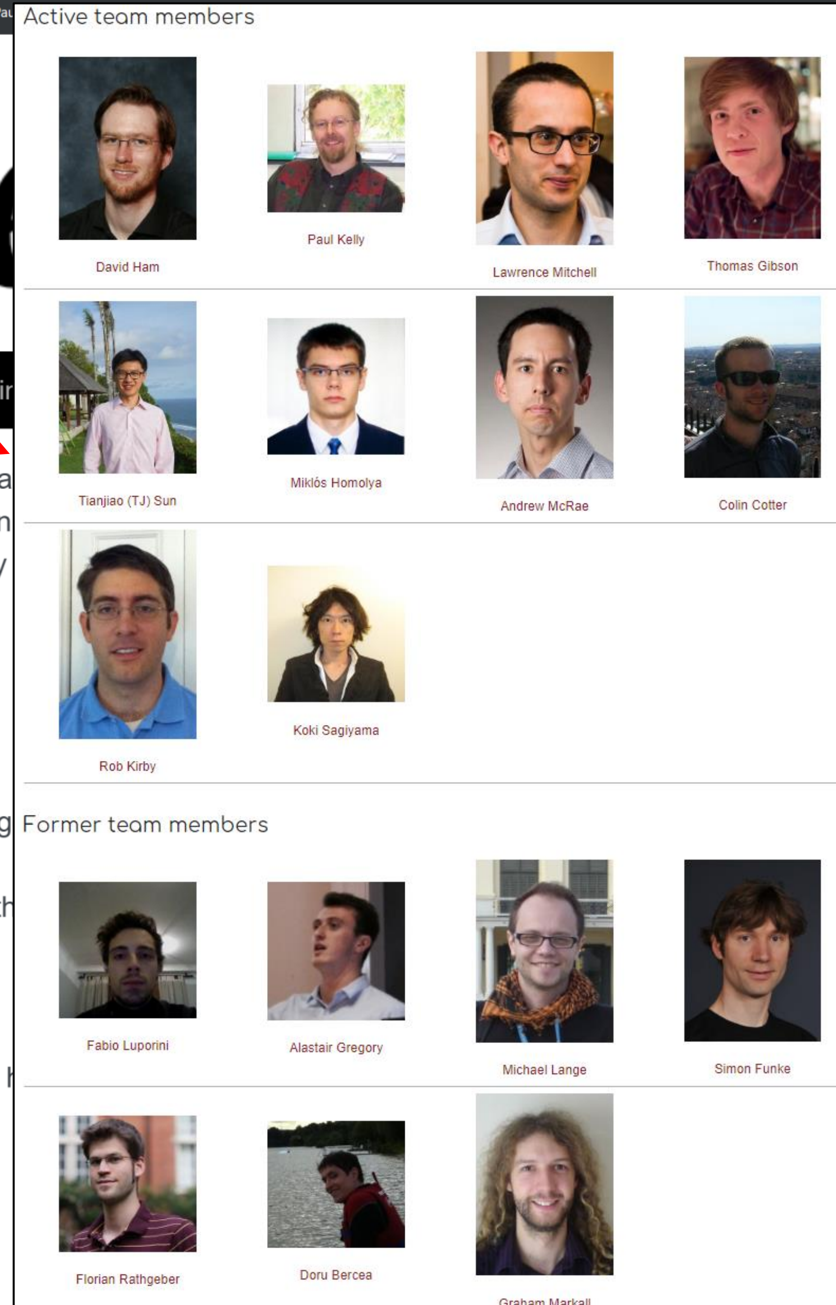
firedrakeproject.org

Documentation Download Team Citi…

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

Features:

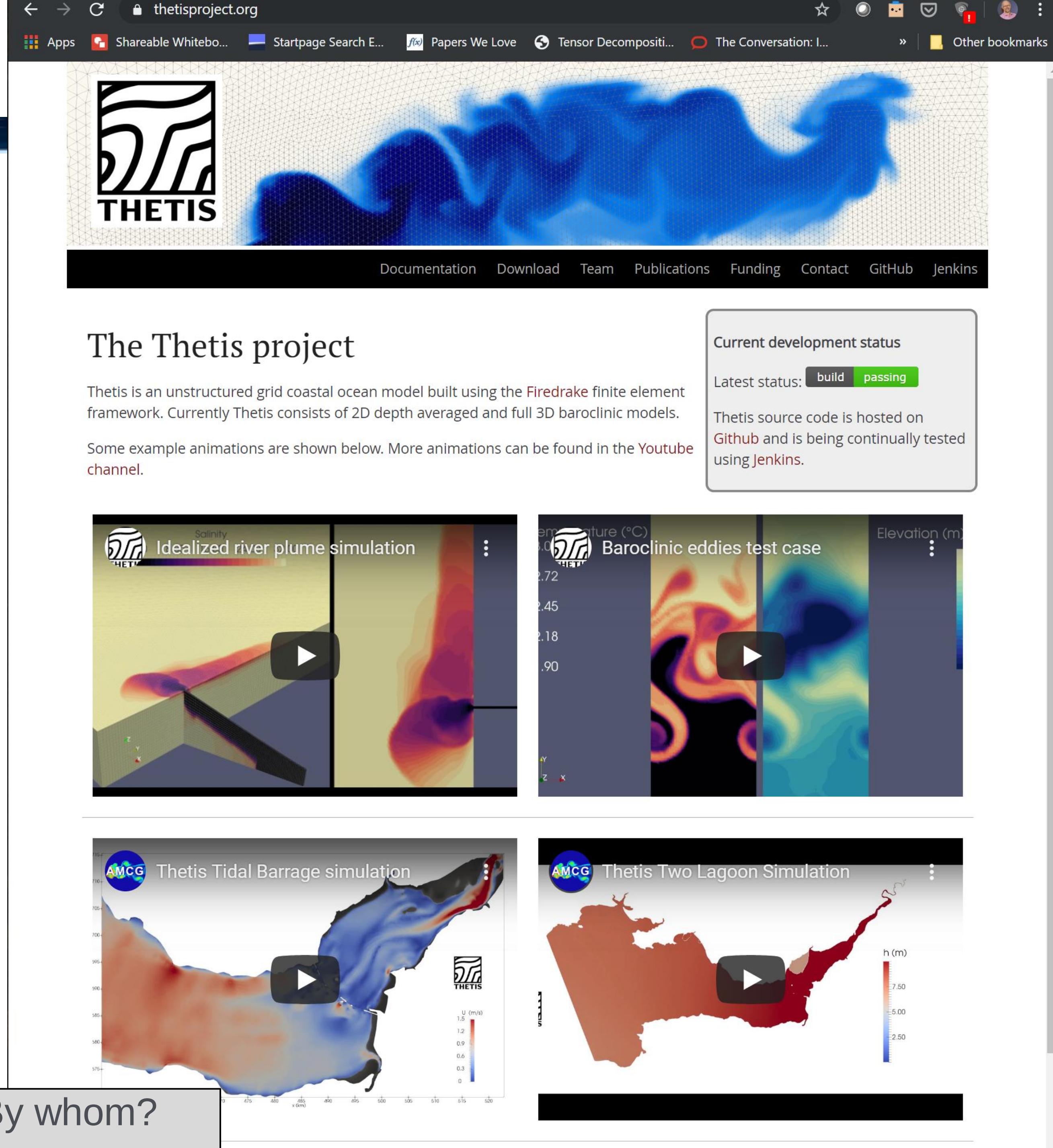
- Expressive specification of any PDE using the Unified Form Languag… Project.
- Sophisticated, programmable solvers through seamless coupling with…
- Triangular, quadrilateral, and tetrahedral unstructured meshes.
- Layered meshes of triangular wedges or hexahedra.
- Vast range of finite element spaces.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



What *is* Firedrake?

Imperial College London

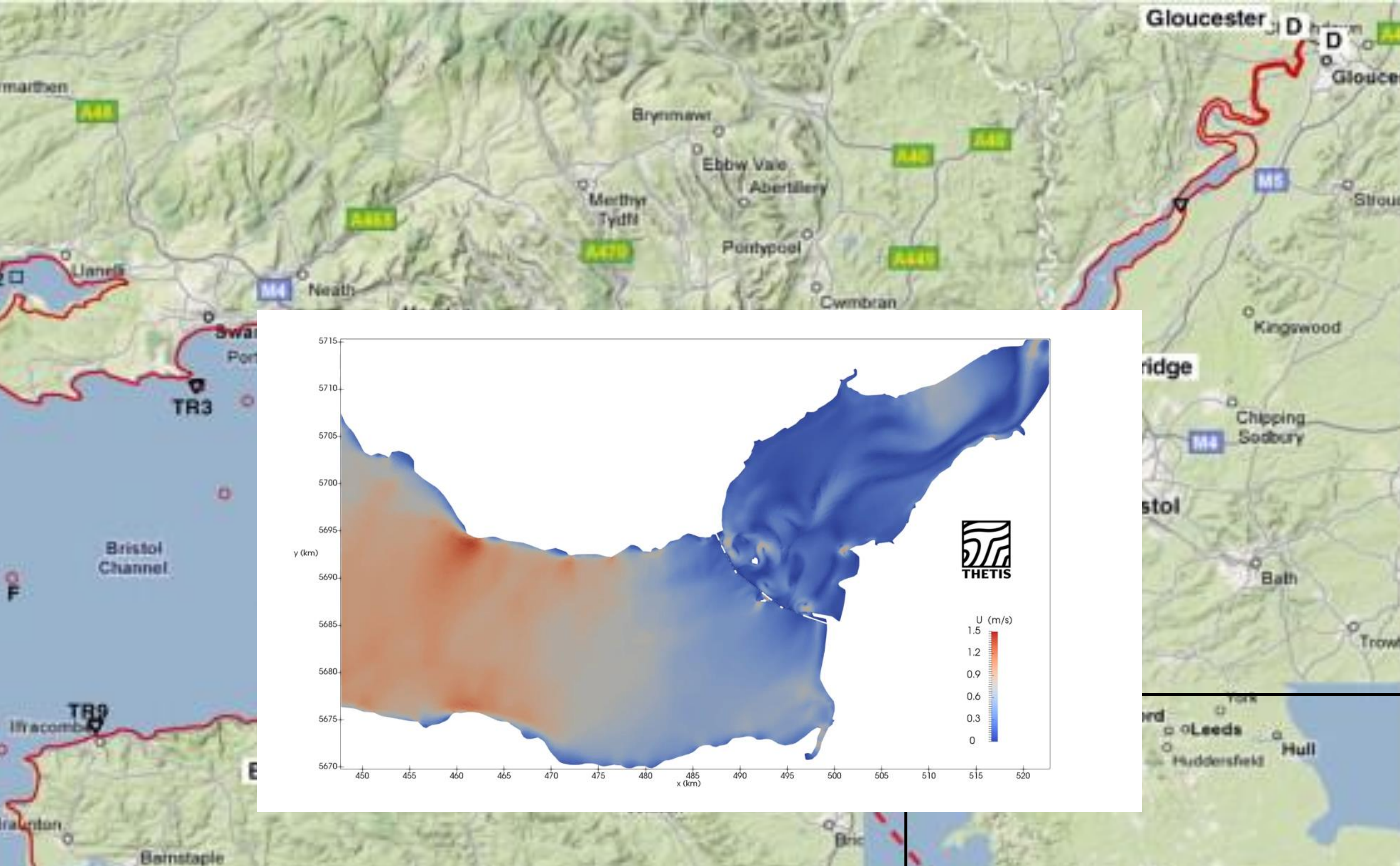
- Firedrake is used in:
  - **Thetis**: unstructured grid coastal modelling framework



- What is it used for? By whom?

- *Estuary of the River Severn: huge tidal energy opportunity*
- *Significant causes for concern over ecological impact*
- *Should we do it? How? Where? How much energy? How much impact?*

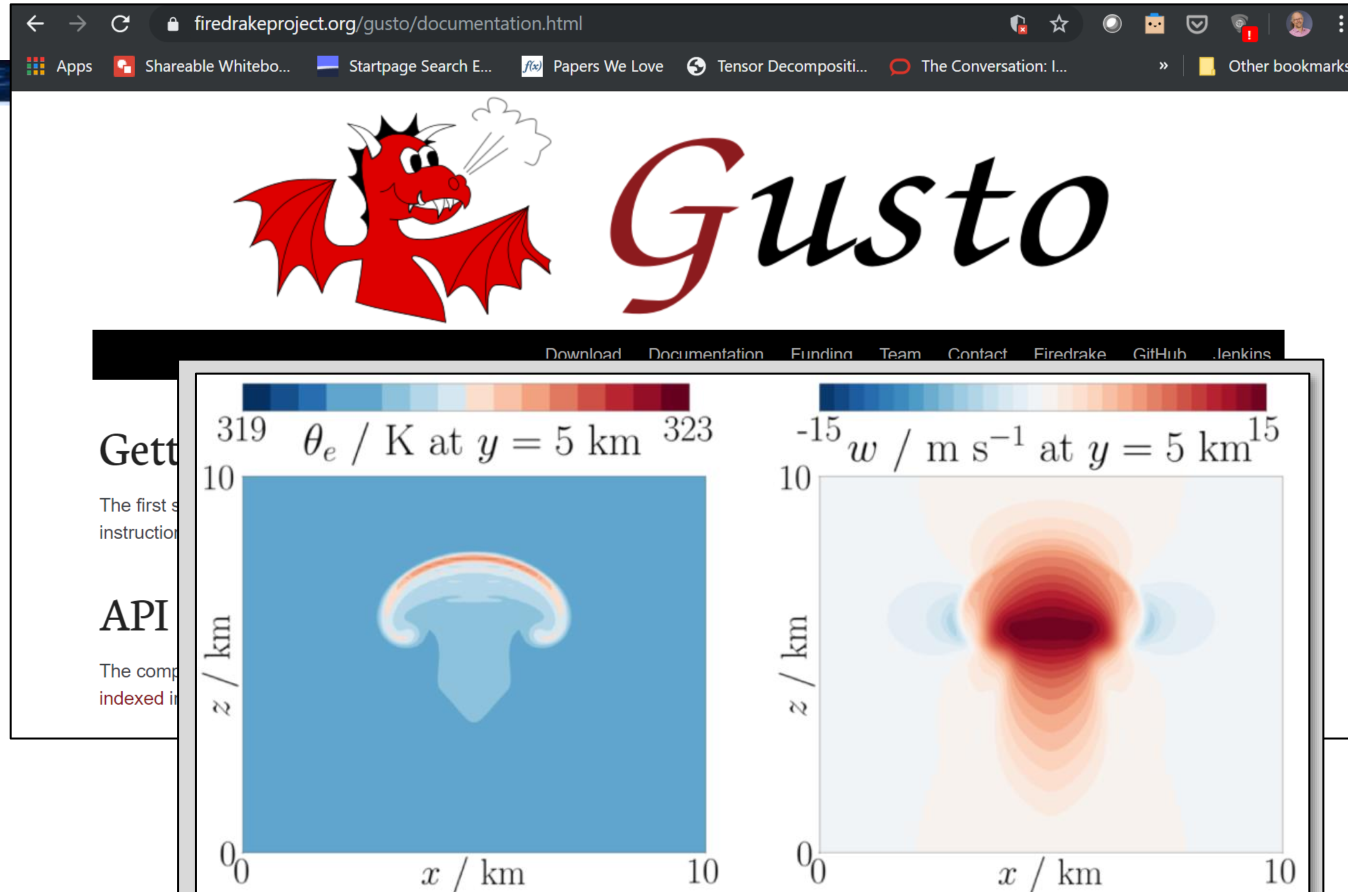
https://doi.org/10.1016/j.apenergy.2009.11.024



- *Estuary of the River Severn: huge tidal energy opportunity*
- *Significant causes for concern over ecological impact*
- *Should we do it? How? Where? How much energy? How much impact?*

https://doi.org/10.1016/j.apenergy.2009.11.024

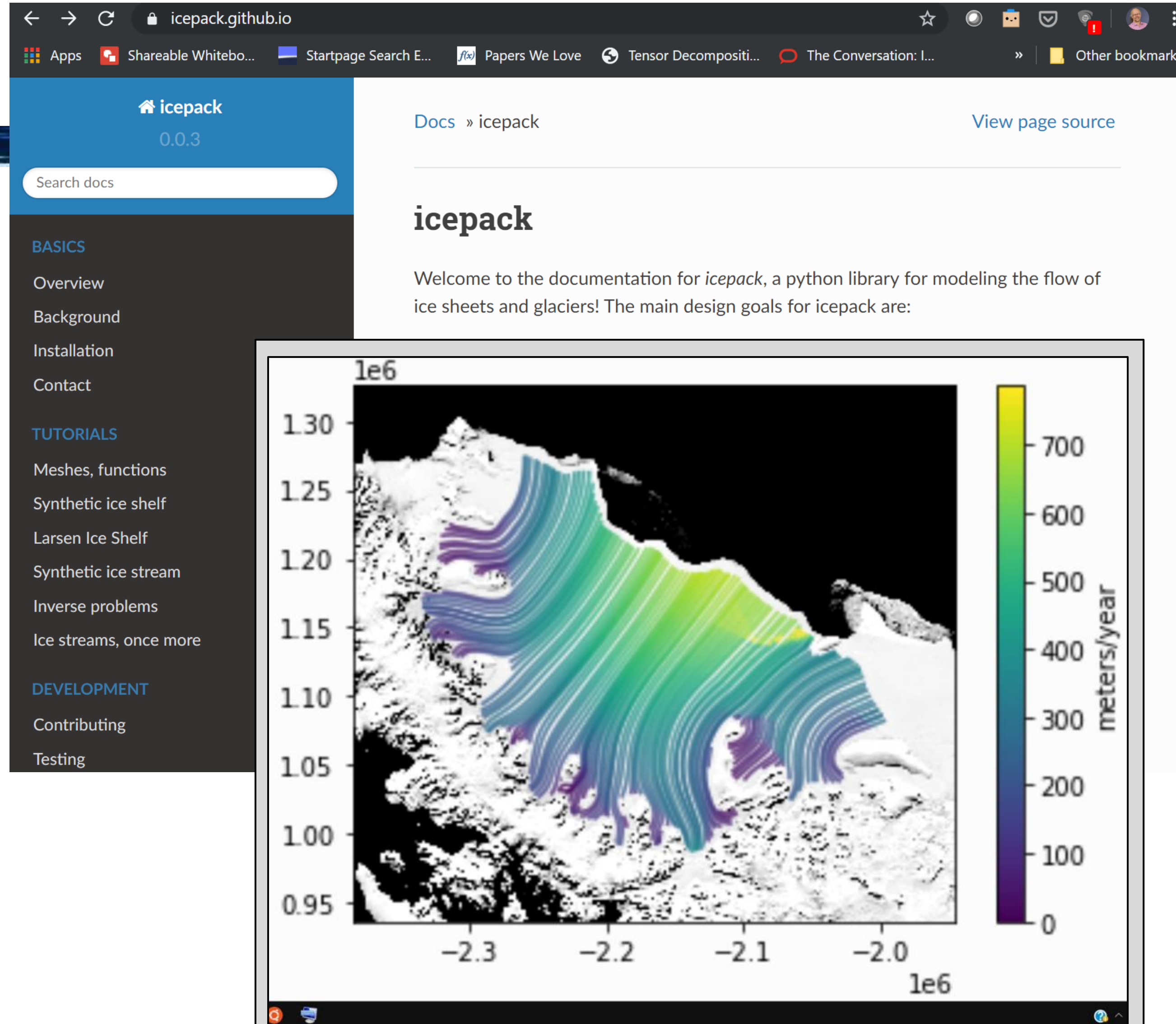
- Firedrake is used in:
  - **Gusto**: atmospheric modelling framework being used to prototype the next generation of weather and climate simulations for the UK Met Office



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?

- 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?

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
```

- Set up the equation and solve for the next timestep u:

```
solve(F == 0, u)
```

- At this point, Firedrake generates code to assemble a linear system, runs it and calls a linear solver (we use PetSC)

```
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"))
```

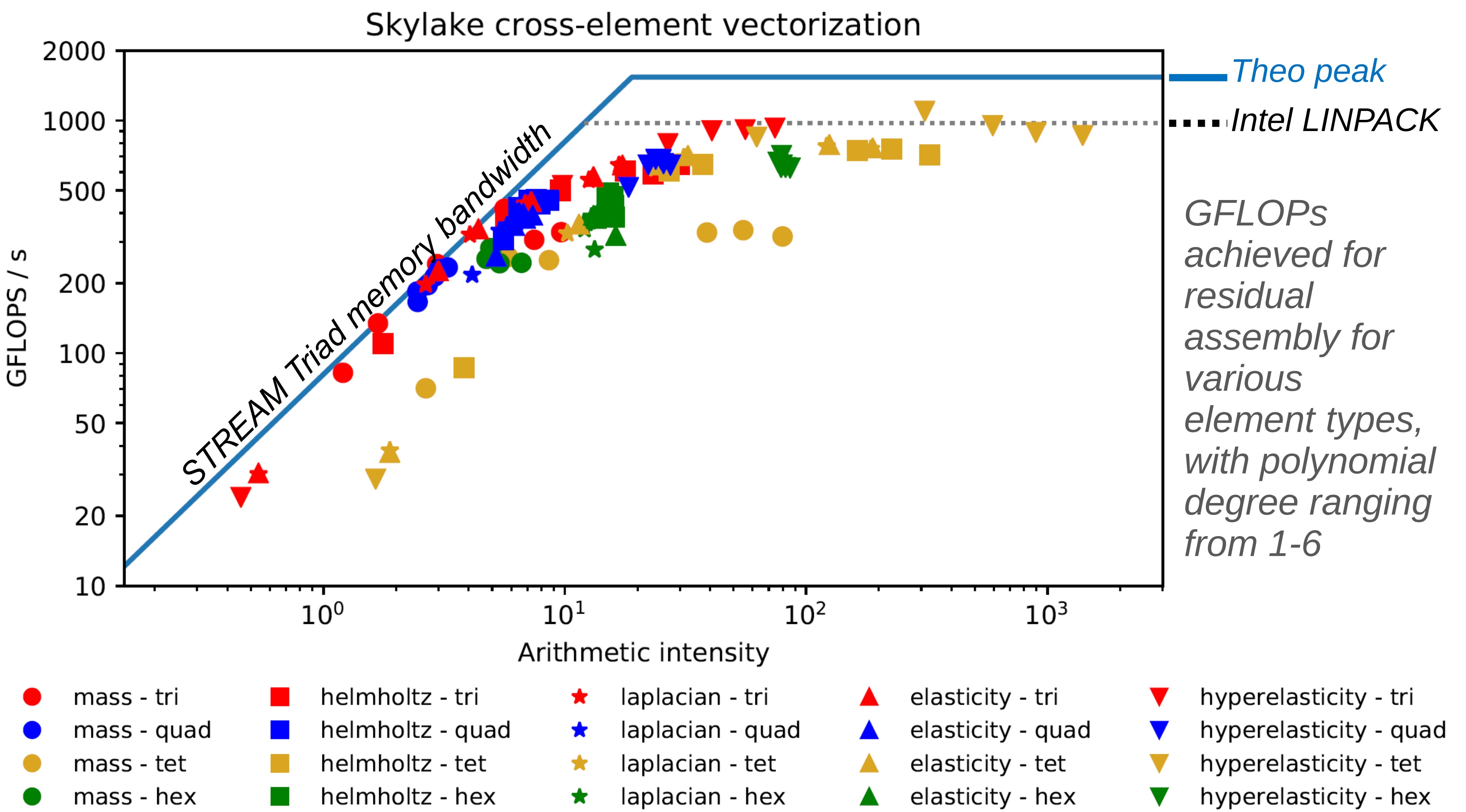
- What does its DSL actually look like?

```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

Firedrake: single-node AVX512 performance

- Does it generate good code?

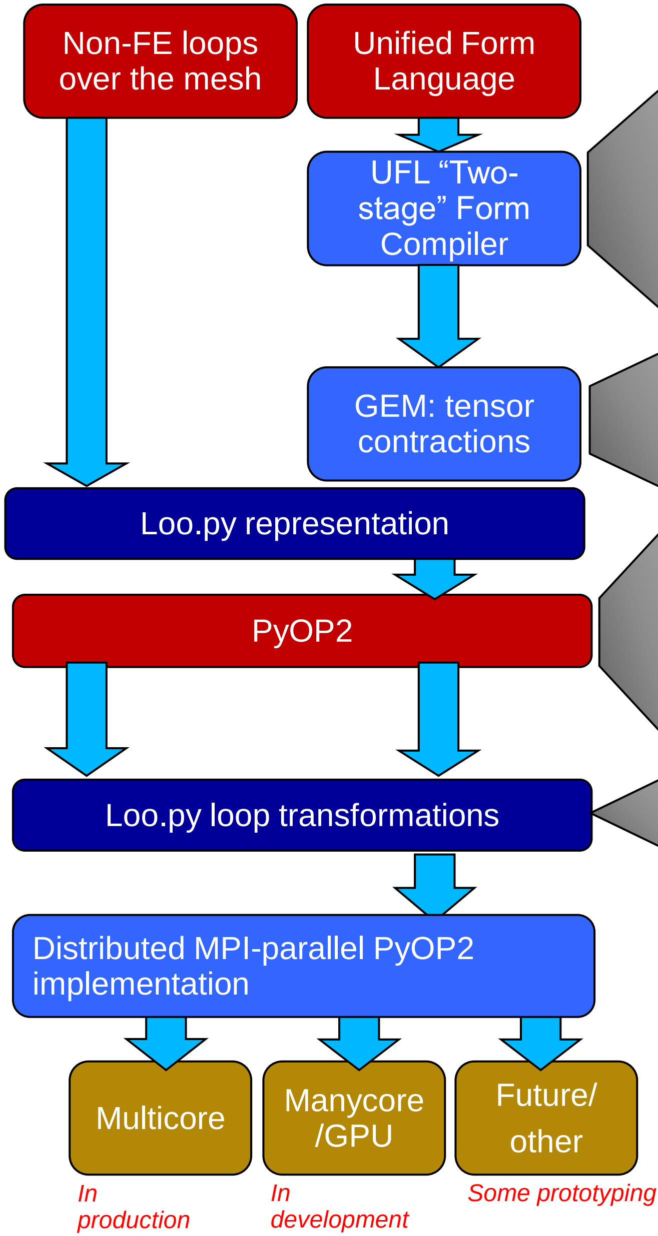


GFLOPs achieved for residual assembly for various element types, with polynomial degree ranging from 1-6

[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 IJHPCA 2020 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

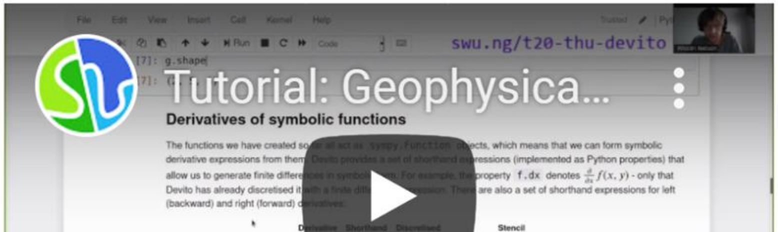
Another example DSL:

Devito: Symbolic Finite Difference Computation

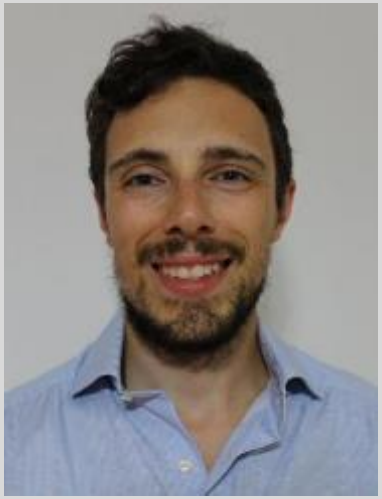
Devito is a domain-specific Language (DSL) and code generation framework for the design of highly optimised finite difference kernels for use in inversion methods. Devito utilises SymPy to allow the definition of operators from high-level symbolic equations and generates optimised and automatically tuned code specific to a given target architecture.

Symbolic computation is a powerful tool that allows users to:

- Build complex solvers from only a few lines of high-level code
- Use automated performance optimisation for generated code
- Adjust stencil discretisation at runtime as required
- (Re-)development of solver code in hours rather than months



Gerard Gorman



Fabio Luporini

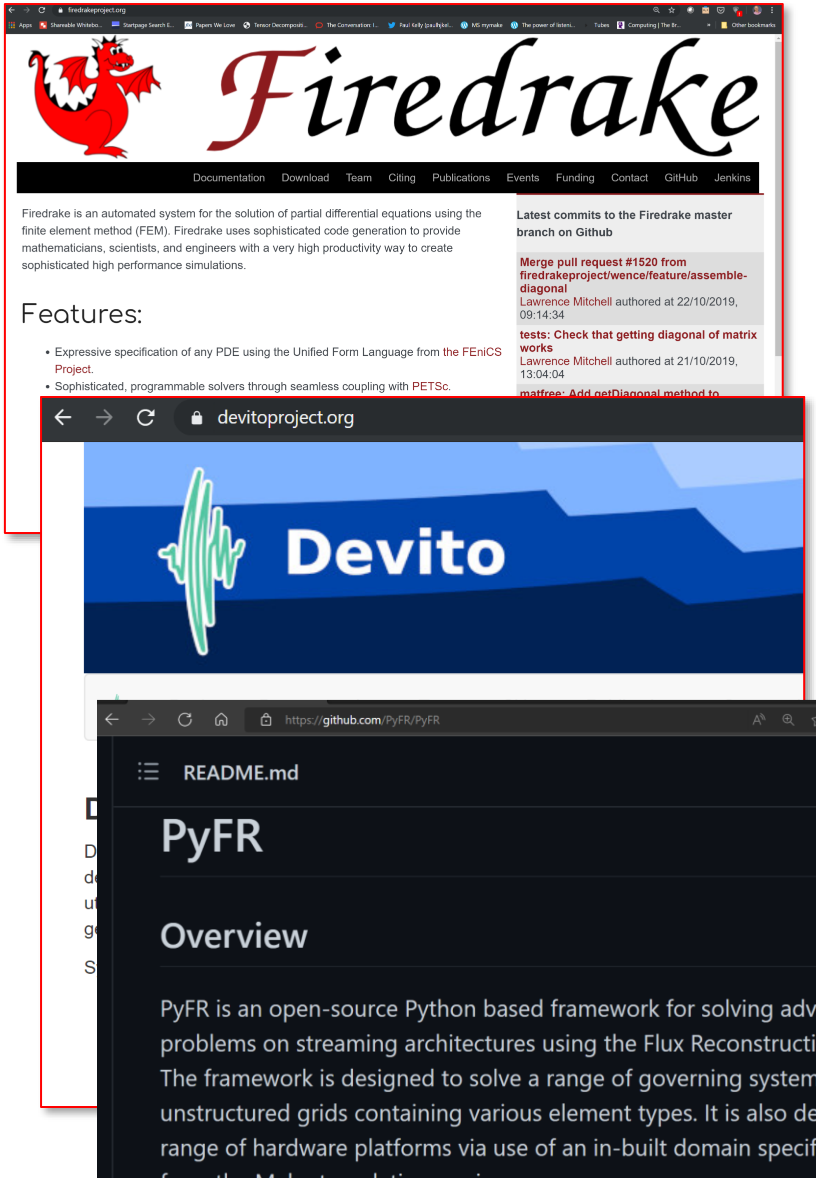
And many many more!

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

How can we change the world?

- **The real value of Firedrake and Devito is in supporting the applications users in exploring *their* design space**
- **We enable them to navigate rapidly through alternative solutions to their problem**
- **In the future, we will have automated pathways from maths to code for many classes of problem, and many alternative solution techniques**



The actual message of this talk….

- **Domain-specific code generation tools really work**

- **But we can’t afford to continue like this!**

Imperial College London

DSLs are a dysfunctional ecosystem

- Users have really good reasons not to adopt DSLs:
  - **Longevity**: will the tool support outlive my project?
  - **Walled garden**: if later I discover I need something the DSL can't do, I'm in trouble
  - **Interoperability**: how do I plug DSL stuff together with other parts of my code?
- Devito shares no code with Firedrake above the C compiler
- For example: dozens of stencil DSLs
  - Almost all are dead academic projects
  - None share any code

Stencil DSLs:

- Halide
- Gridtools/Stella
- Open Earth Compiler
- Psyclone
- YASK
- OPS
- Artemis
- StencilGen
- Lift
- SDSLc
- ExaStencils
- Patus
- Physis
- Mint
- Pochoir
- ShiftCalculus
- HPF
- MSF
- ….

We have to work together…

- What do you need to do to get leading-edge performance with stencils?
  - Spatial tiling (hierarchical)
  - Temporal tiling (wavefront, diamond….)
  - Data layout transformations (“dimension lifting”)
  - Arrow scheduling (use associativity to control register pressure)
  - Cross-iteration redundancy elimination (CIRE)
  - Shuffling (“Software Systolic Array”)
  - Matrix ISA optimisations, tensor cores
  - Loop fusion/fission/distribution
  - Compute-communicate overlap
  - Wavefield compression
  - Precision management
  - And many many more!!!

We have to work together…

- What do you need to do to get leading-edge performance with stencils?
  - Spatial tiling (hierarchical)
  - Temporal tiling (wavefront, diamond….)
  - Data layout transformations (“dimension lifting”)
  - Arrow scheduling (use associativity to control register pressure)
  - Cross-iteration redundancy elimination (CIRE)
  - Shuffling (“Software Systolic Array”)
  - Matrix ISA optimisations, tensor core
  - Loop fusion/fission/distribution
  - Compute-communicate overlap
  - Wavefield compression
  - Precision management
  - And many many more!!!

No one team can expect to integrate all the known tehniques into a single silo of code

We have to build code that works together…

- **What does it mean to *compose* DSLs?**

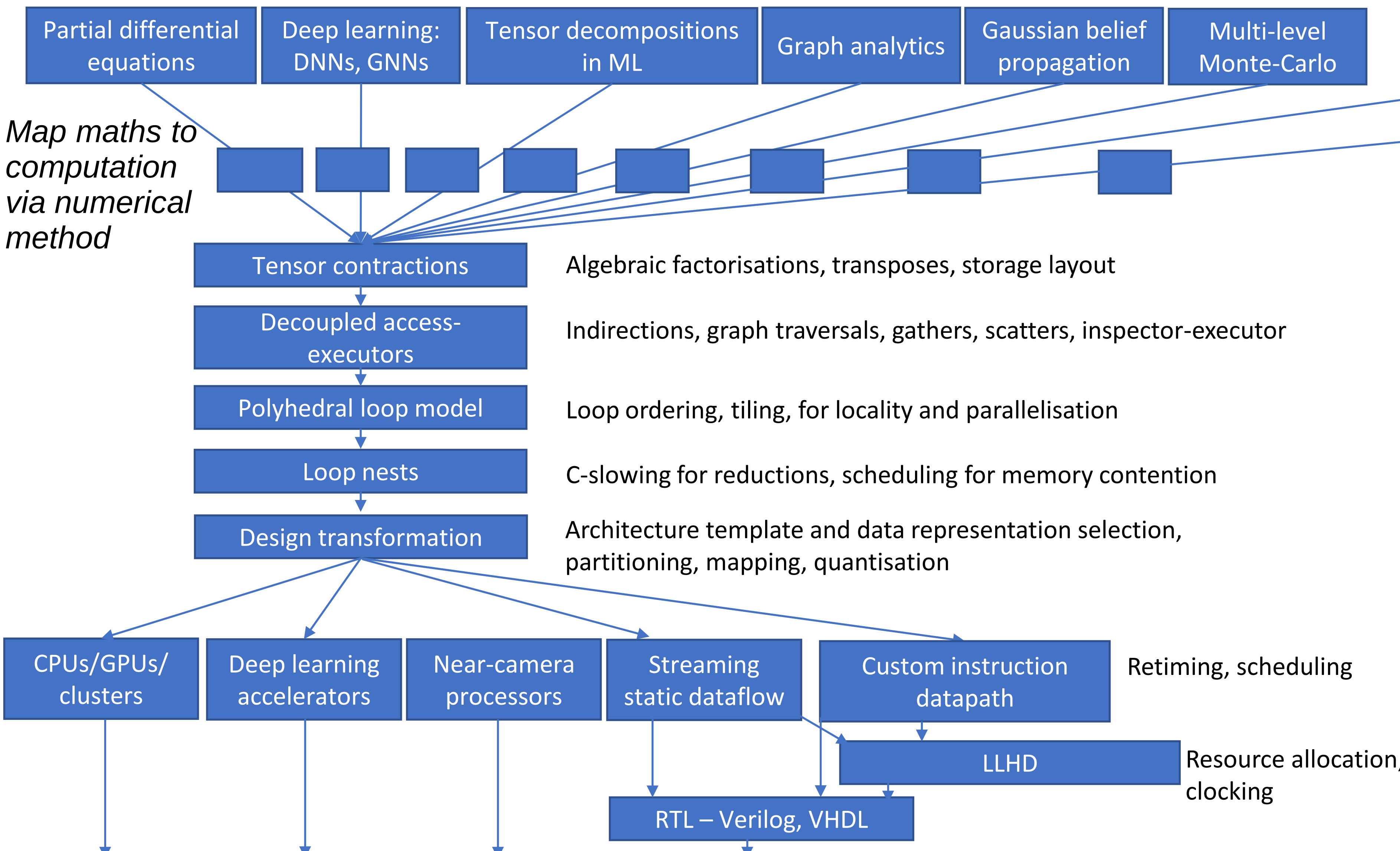
- Target-specific DSLs
  - **Eg**: CUDA, HLS, Graphcore’s Poplar, Maxeler’s MaxJ
  - **Composition**: eg MPI+CUDA, MPI+X, multi-target OpenMP

- Data-structure-specific DSLs
  - **Eg** OP2/PyOP2, OPS, Psyclone, SAMR DSLs like Chombo, BVH DSLs like Optix, Octtree DSLs like SuperEight, Tensorflow
  - **Composition:** Particle-in-cell, Particle mesh Ewald, Multiphysics frameworks like Uintah, visualisation tools like VTK

- DSLs that automate numerical methods
  - **Eg:** FeNICS, Firedrake, Devito, PyFR, OpenSBLI, SPIRAL, GPFlow
  - Composition:
    - Coupled problems: fluid-structure, PME, Model Coupling Toolkit
    - Outer-loop: PDE-constrained optimisation (FWI), assimilation (4DVAR), uncertainty quantification (MLMC), parameterisation with deep learning
    - At this level we have access to the *maths*!

Imperial College London

Vision for the future



Imperial College London

Acknowledgements

Thank you to our many many collaborators!

Partly funded/supported by

- SysGenX: Composable software generation for system-level simulation at Exascale (EP/W026066/1)
- XDSL: Efficient Cross-Domain DSL Development for Exascale (EP/W007789/1)
- 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)
- Schloss Dagstuhl

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