Imperial College London

Compiling for data not code

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

This talk includes work done by or influenced by: David Ham (Imperial Maths), Lawrence Mitchell (University of Durham)
Gerard Gorman, Fabio Luporini, (Imperial Earth Science Engineering – Applied Modelling and Computation Group)
Mike Giles, Gihan Mudalige, Istvan Reguly (Mathematical Inst, Oxford)
Spencer Sherwin, Peter Vincent, Chris Cantwell (Aeronautics, Imperial)
Michelle Mills Strout (Univ of Arizona), Chris Krieger, Cathie Olschanowsky (Colorado State University)
Carlo Bertolli, Doru Bercea (IBM Research*), Richard Veras, Ram Ramanujam (Louisiana State University)
Doru Thom Popovici, Franz Franchetti (CMU), Karl Wilkinson (Capetown), Chris–Kriton Skylaris (Southampton)
Sajad Saeedi (Ryerson University*), Luigi Nardi (Stanford/Lund University*), Ridgway Scott (University of Chicago)
Florian Rathgeber (Google*), Michael Lange (ECMWF*), Graham Markall (NVIDIA*), Francis Russell (Hadean*), George Rokos (Intel*), Tianjiao Sun (Cerebras*), Thomas Debrunner (IniVation), Mehedi Paribartan (Imperial), Freddie Witherden (Texas A&M)

* Work done while at Imperial

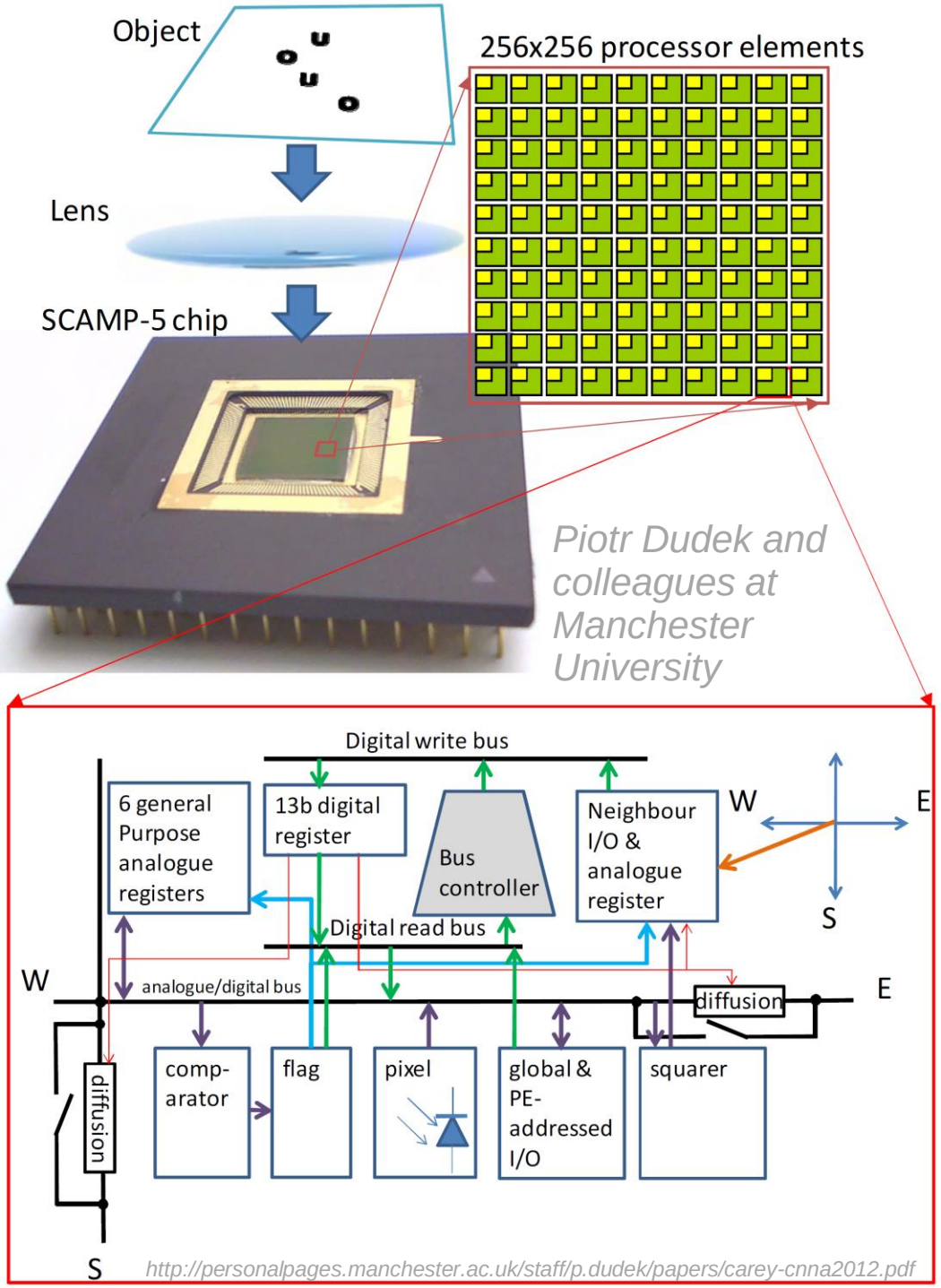
- Compilers usually compile *code*
- This talk is about compiling *data*
- Examples:
  - Convolutions on an analogue SIMD image sensor
  - Block-panel matrix-multiply: GiMMiK and libxsmm
  - The “topological” optimisation
  - Steiner points
  - Keeping a whole matrix in registers
  - Tiling
  - Re-ordering
  - Beyond:
    - Matrix factorisation
    - Matrix approximation
    - Training for convenient values

2

Cameras produce images for humans, not machines

3

SCAMP 5 focal-plane sensor processor

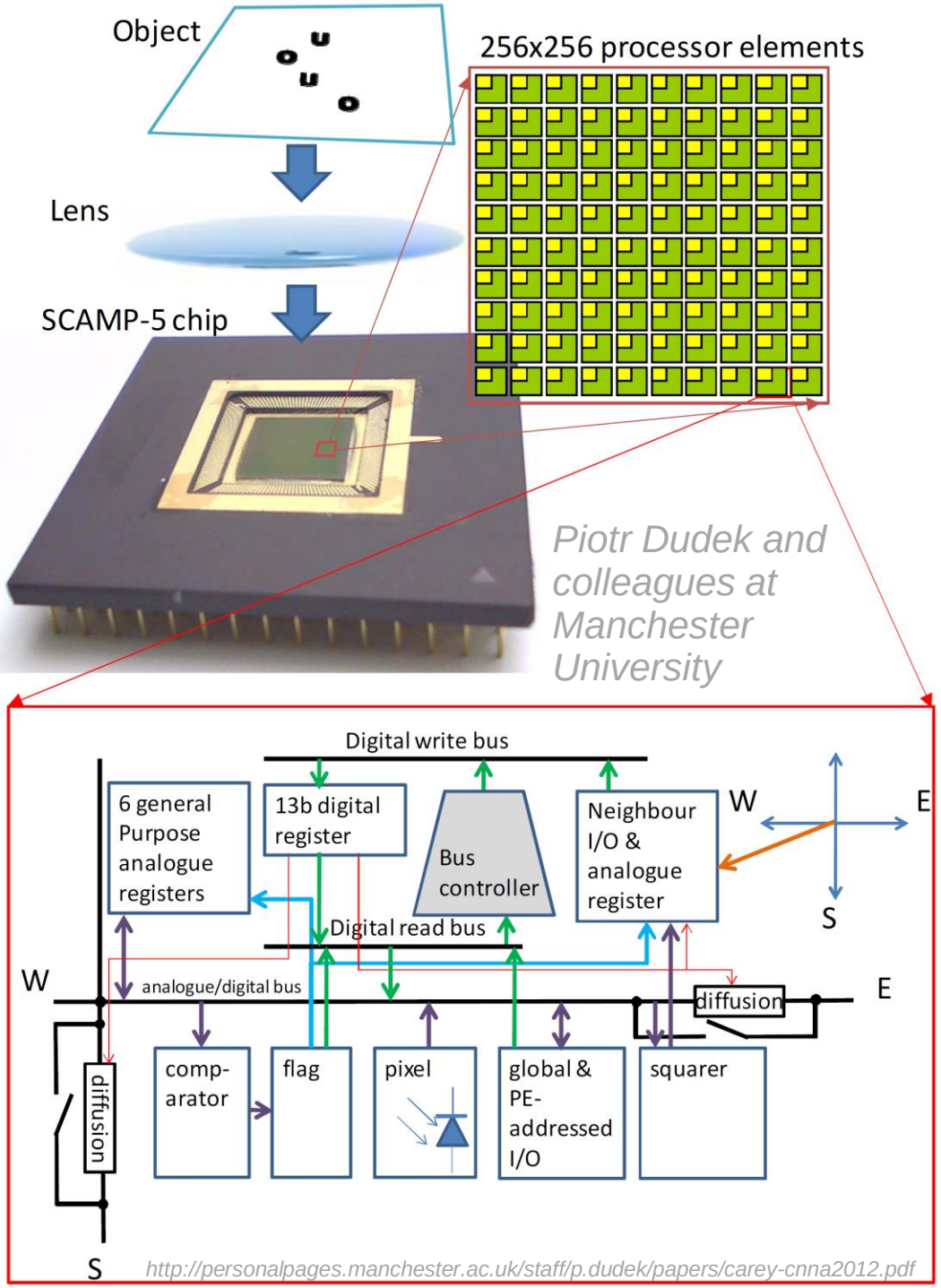


Piotr Dudek and colleagues at Manchester University

http://personalpages.manchester.ac.uk/staff/p.dudek/papers/carey-cnna2012.pdf

- **256x256 SIMD** processor array
- **Light sensor** on every processor
- Ca.**170 transistors** per processor

SCAMP 5 focal-plane sensor processor



- **Seven registers** holding analogue values
- Computation by **moving charge**
- **Addition** is easy
- **No multiply**
- North-east-west-south **data movement**


Motion Estimation and Mapping using SCAMP-5 FPSP.
BIT-VO operates at 300 FPS on a live system.

Basic instruction set *(of interest)*

- **Shift** image x
- **Shift** image y
- **Add** two images
- **Subtract** two images
- **Scale** image by 1/2
- Take **absolute value** of image

How to do convolution filters on SCAMP 5?

For image filtering

As a component in image processing algorithms

Notably CNNs

Convolution filters on SCAMP 5

Easy filters

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

We can add/subtract repeatedly – so we can multiply by a constant

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Convolution filters on SCAMP 5

Harder filters

$$\begin{bmatrix} 0.125 & 0.25 & 0.125 \\ 0.25 & 0.5 & 0.25 \\ 0.125 & 0.25 & 0.125 \end{bmatrix}$$

Convolution filters on SCAMP 5

Harder filters – still easy

$$\begin{bmatrix} 0.125 & 0.25 & 0.125 \\ 0.25 & 0.5 & 0.25 \\ 0.125 & 0.25 & 0.125 \end{bmatrix} = \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

We can divide by two repeatedly

Convolution filters on SCAMP 5

Hard filters

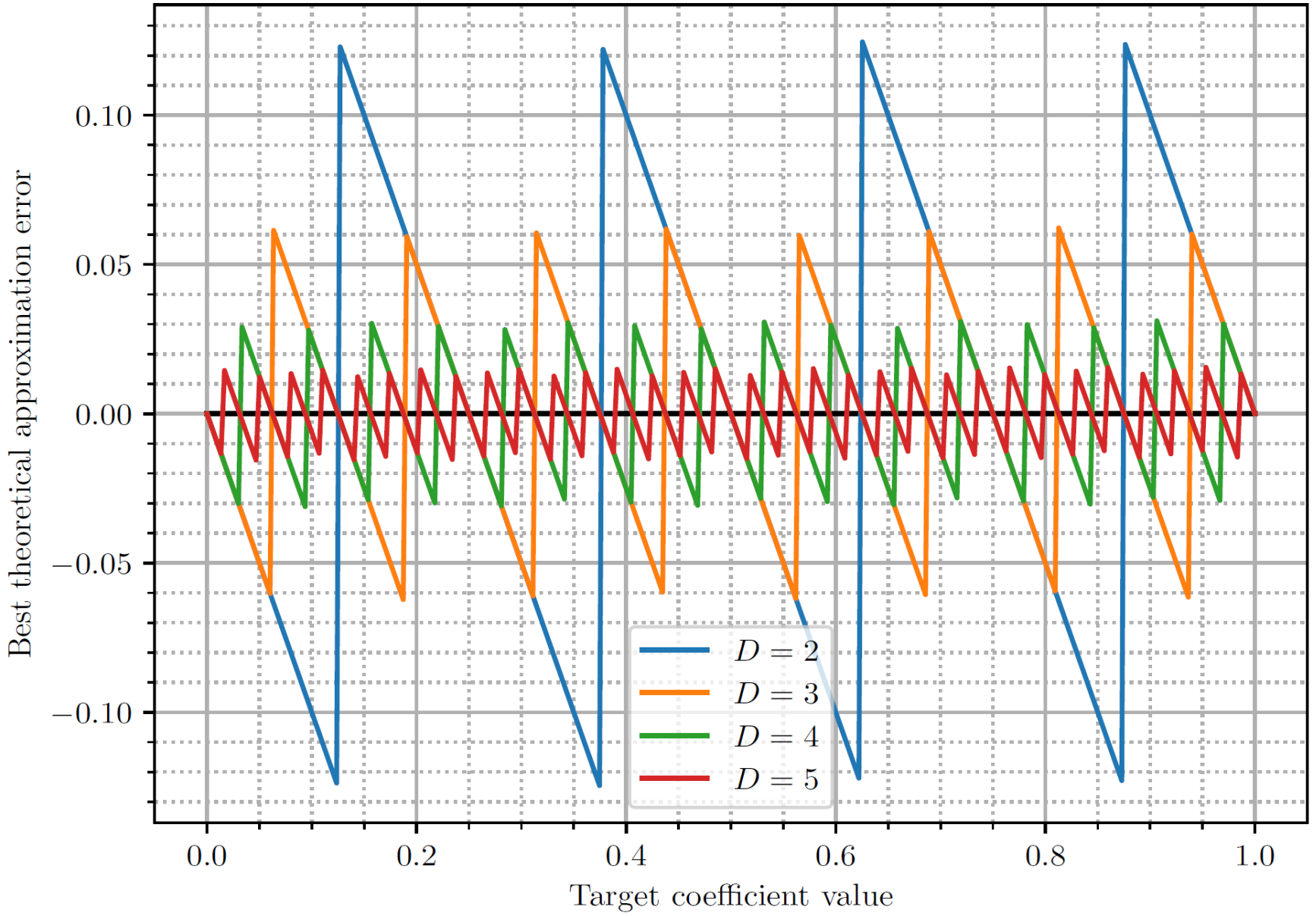
$$\begin{bmatrix} 1.12 & 0.23 & 0.88 \\ 0.36 & 0.51 & 0.89 \\ 0.16 & 0.13 & 0.73 \end{bmatrix}$$

Convolution filters on SCAMP 5

Hard filters – easy again

$$\begin{bmatrix} 1.12 & 0.23 & 0.88 \\ 0.36 & 0.51 & 0.89 \\ 0.16 & 0.13 & 0.73 \end{bmatrix} \approx \begin{bmatrix} 1.125 & 0.25 & 0.875 \\ 0.375 & 0.5 & 0.875 \\ 0.125 & 0.125 & 0.75 \end{bmatrix} = \frac{1}{8}\begin{bmatrix} 9 & 2 & 7 \\ 3 & 4 & 7 \\ 1 & 1 & 6 \end{bmatrix}$$

We can approximate



We can approximate – D-digit binary weights

With CNNs, we can train for representable weights

Filters often have repeated terms

$$\begin{bmatrix} 1.125 & 0.25 & 0.875 \\ 0.375 & 0.5 & 0.875 \\ 0.125 & 0.125 & 0.75 \end{bmatrix} = \frac{1}{8}\begin{bmatrix} 9 & 2 & 7 \\ 3 & 4 & 7 \\ 1 & 1 & 6 \end{bmatrix}$$

We implement multiplication using summations – so there are *lots* of common subterms

We can shift intermediate values to save redundant computation

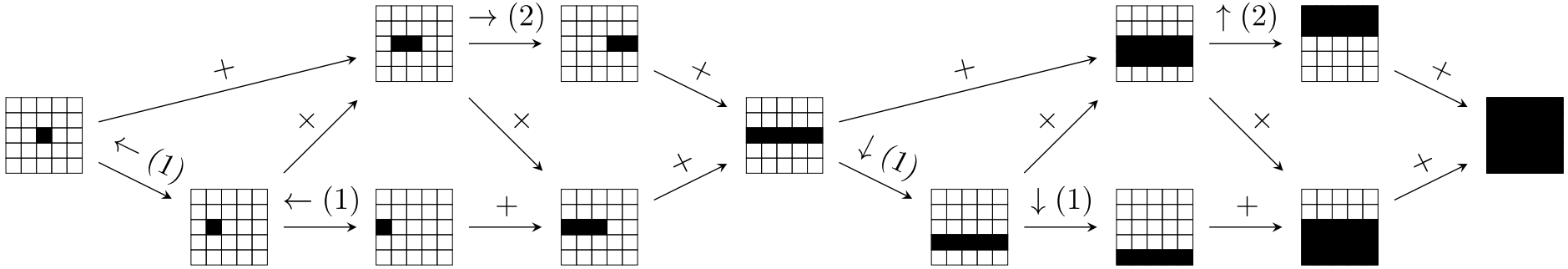
Our compiler takes a convolution as input, and generates optimised code

Simple motivating (extreme) example

5x5 Box:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Naively: 25 additions




```
1  B = east(A)
2  A = add(A, B)
3  B = east(B)
4  B = add(B, A)
```

```
5  A = west(A)
6  A = west(A)
7  A = add(B, A)
8  B = north(A)
9  A = add(A, B)
```

```
10 B = north(B)
11 B = add(A, B)
12 A = south(A)
13 A = south(A)
14 A = add(B, A)
```

6 additions

Thomas Debrunner, *Sajad Saeedi, and Paul H. J. Kelly. 2019. AUKE: Automatic Kernel Code Generation for an Analogue SIMD Focal-Plane Sensor-Processor Array. ACM TACO 15, 4, Article 59 (January 2019),*

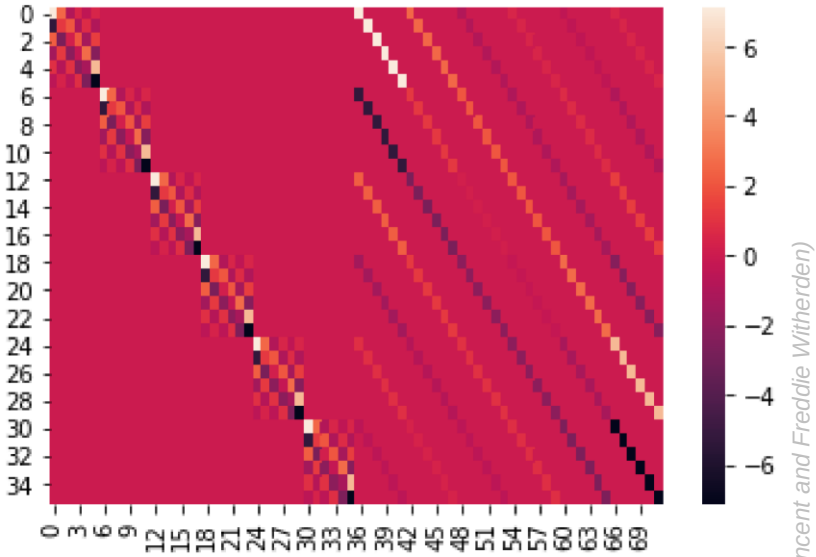
GiMMiK and libxsmm

- Our motivation: CFD using PyFR
- Flux reconstruction – roughly, high-order discontinuous-Galerkin finite element

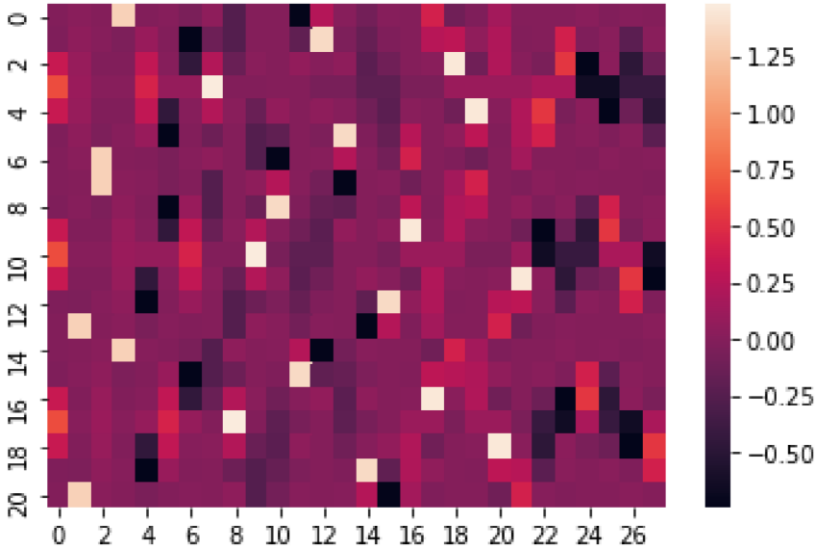
- 18,000 K20X GPUs on Titan. 195 billion DOFs, achieved 13.7 DP-PFLOP/s (58.0% peak accelerator DP-FLOP/s).

- *Peter Vincent, Freddie Witherden, Brian Vermeire, Jin Seok Park, and Arvind Iyer. 2016. Towards green aviation with python at petascale. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '16). IEEE Press, Article 1, 1–11.*
- Shortlisted for Gordon Bell Prize

- Flux reconstruction is dominated by block-panel GEMM:
  - C = A * B
  - Where A is small (<100x100) and compile-time constant
  - And sometimes sparse
  - And highly structured
  - Precise structure depends on PDE and discretisation
  - Full unrolling works really well
  - All the zeroes disappear
  - GiMMiK generates CUDA code for the matrix multiply
  - We evaluate using a large suite of matrices found in PyFR applications



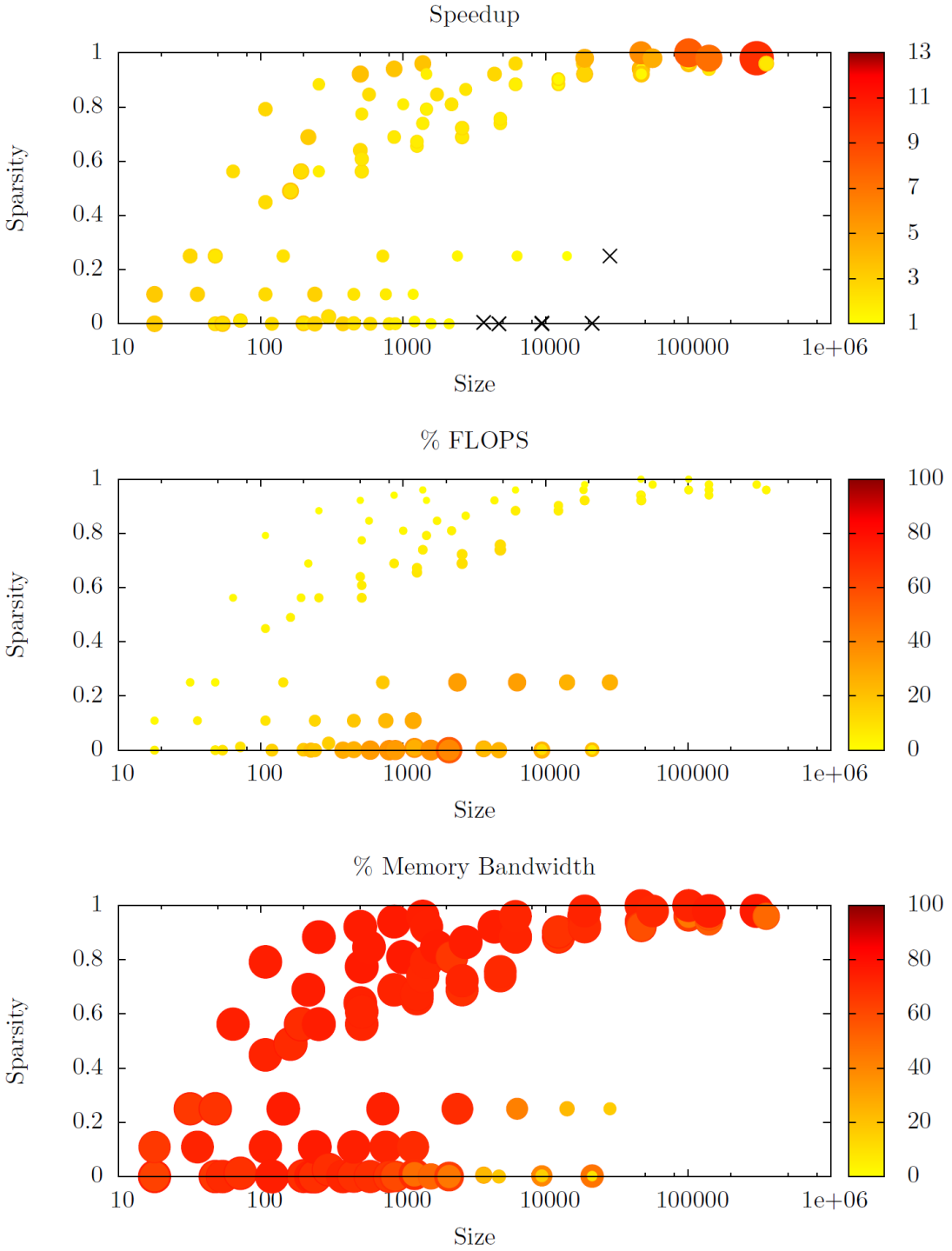
36x36: 2592 elements, 384 non-zeros
32 distinct constants



p5-williams-shunn-m0:
21x18: 378 elements, all non-zero
63 distinct constants

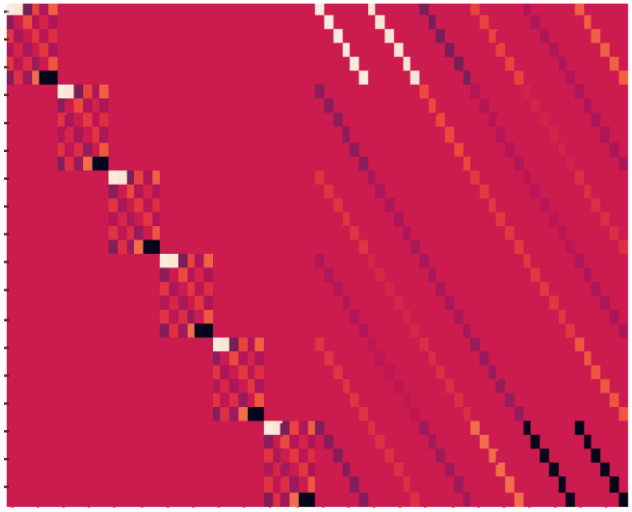
rom Peter Vincent and Freddie Witherden)

- Speedup of GiMMiK's kernels over cuBLAS,
- Achieved percentage of the peak floating-point rate
- Achieved percentage of the peak memory bandwidth
- The metric of interest is represented through the size and colour intensity of the data points. Speedups smaller than 1 are denoted with crosses.
- Double precision on Tesla K40c.

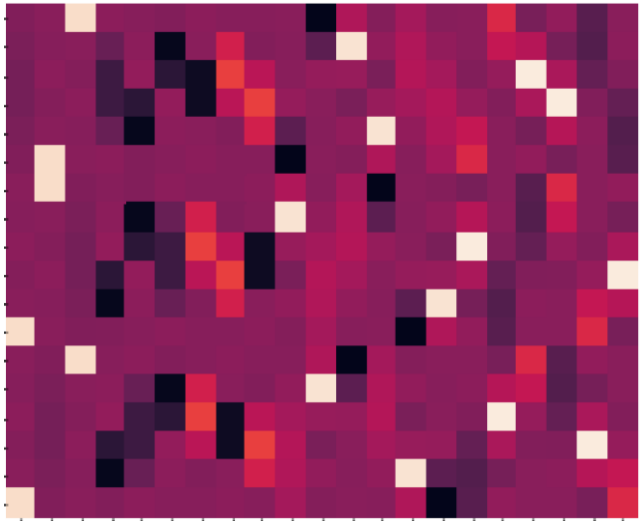


GiMMiK—Generating bespoke matrix multiplication kernels for accelerators: Application to high-order Computational Fluid Dynamics. **BD Wozniak**, FD Witherden, FP Russell, PE Vincent, PHJ Kelly. Computer Physics Communications 202, 12-22

- This idea was re-implemented in Intel's open-source libxsmm library
- Libxsmm is a library for specialized dense and sparse matrix operations as well as for deep learning primitives such as small convolutions
- Libxsmm includes a specialised JIT compiler to generate highly-optimised, vectorised, specialised code for each matrix/convolution (really fast – low 100s of microseconds)
- https://github.com/hfp/libxsmm
  - *Alexander Heinecke, Greg Henry, Maxwell Hutchinson, and Hans Pabst. 2016. LIBXSMM: accelerating small matrix multiplications by runtime code generation. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '16). IEEE Press, Article 84, 1–11.*



p5-gauss-legendre-lobatto-m132:
36x36: 2592 elements, 384 non-zeros
32 distinct constants



p5-williams-shunn-m0:
21x18: 378 elements, all non-zero
63 distinct constants

(analysis courtesy Mehedi Paribartan, data from Peter Vincent and Freddie Witherden)

More small-matrix optimisations

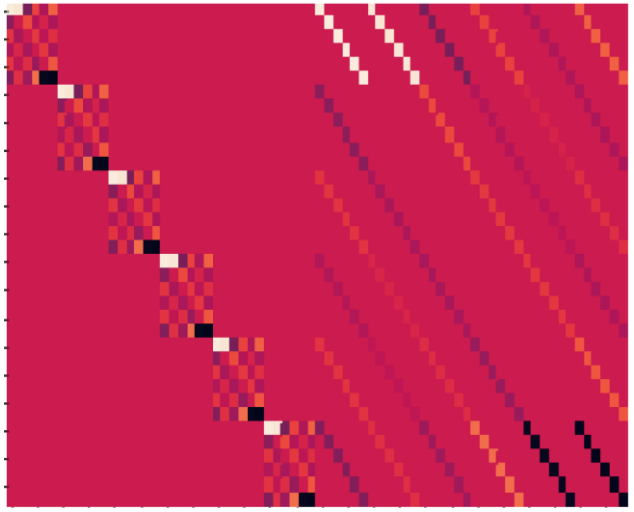
Registerise the A matrix:

- In common PyFR cases, the number of *distinct* non-zeroes is small
- Small enough to keep in registers
  - Prototype implementation in libxsmm
- Especially if you use all the lanes of the vector registers
  - Prototype implementation, no results yet
- So the A matrix incurs no memory accesses at all

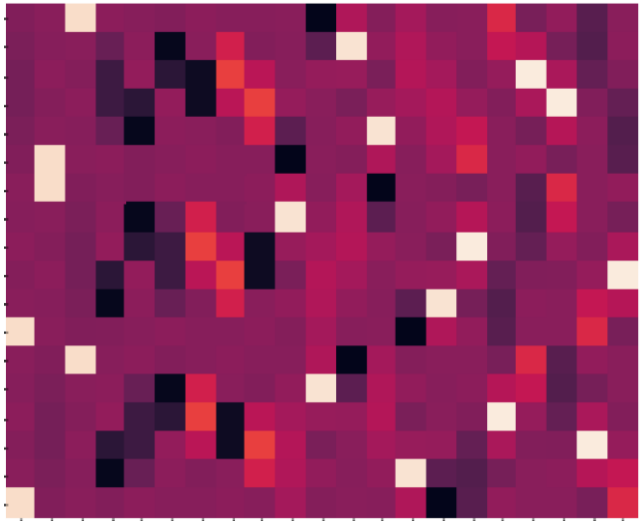
More small-matrix optimisations

Common subexpressions:

- In common PyFR cases, the number of *distinct* non-zeroes is small
- And they recur within the same column
- These result in redundant multiplies
- But we have FMA instructions so eliminating multiplies doesn't help
- Until we see more of them?



p5-gauss-legendre-lobatto-m132:
36x36: 2592 elements, 384 non-zeros
32 distinct constants

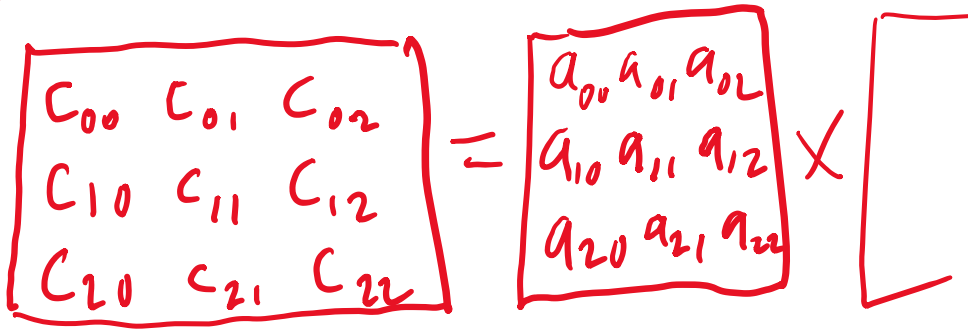


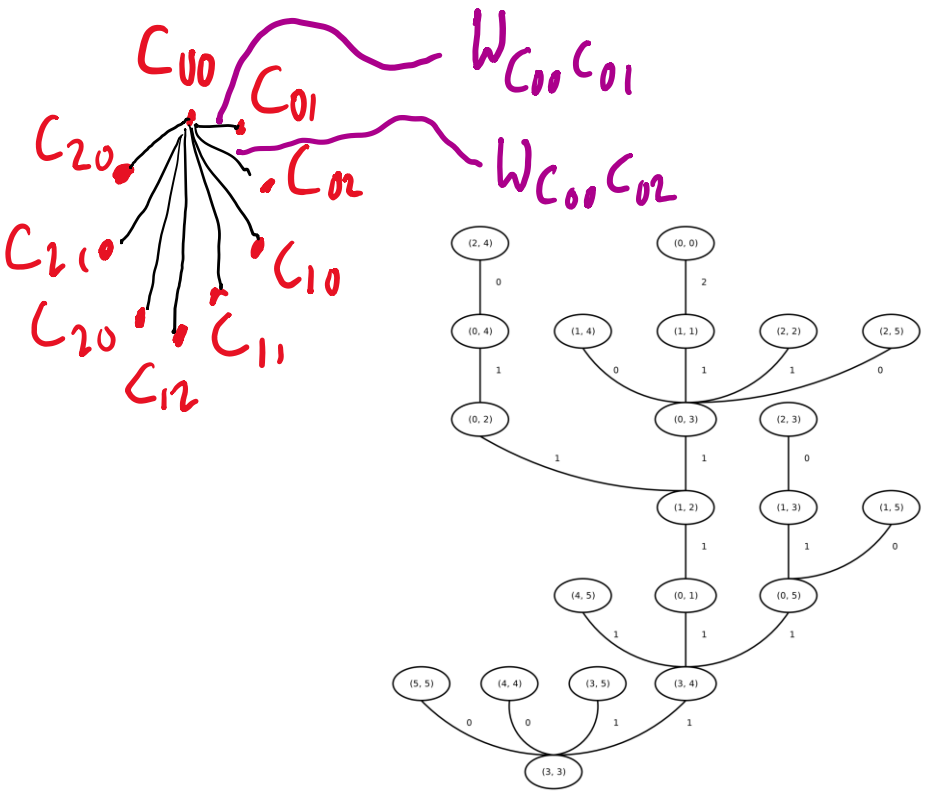
p5-williams-shunn-m0:
21x18: 378 elements, all non-zero
63 distinct constants

(analysis courtesy Mehedi Paribartan, data from Peter Vincent and Freddie Witherden)

The “topological” optimisation idea

- Build a graph with a vertex for each inner product in the GEMM
- Fully-connected
  - Edge a-b weighted with the estimated cost of computing b having just computed a
  - Cost may be reduced if *some* redundancy of *some* kind can be exploited
- Construct a minimum spanning tree of this graph to find an optimal execution strategy
  - Steiner variant: add vertices if they reduce the total





Robert C. Kirby, Anders Logg, L. Ridgway Scott, and Andy R. Terrel. 2006. Topological Optimization of the Evaluation of Finite Element Matrices. SIAM J. Sci. Comput. 28, 1 (January 2006), 224–240. DOI:https://doi.org/10.1137/050635547

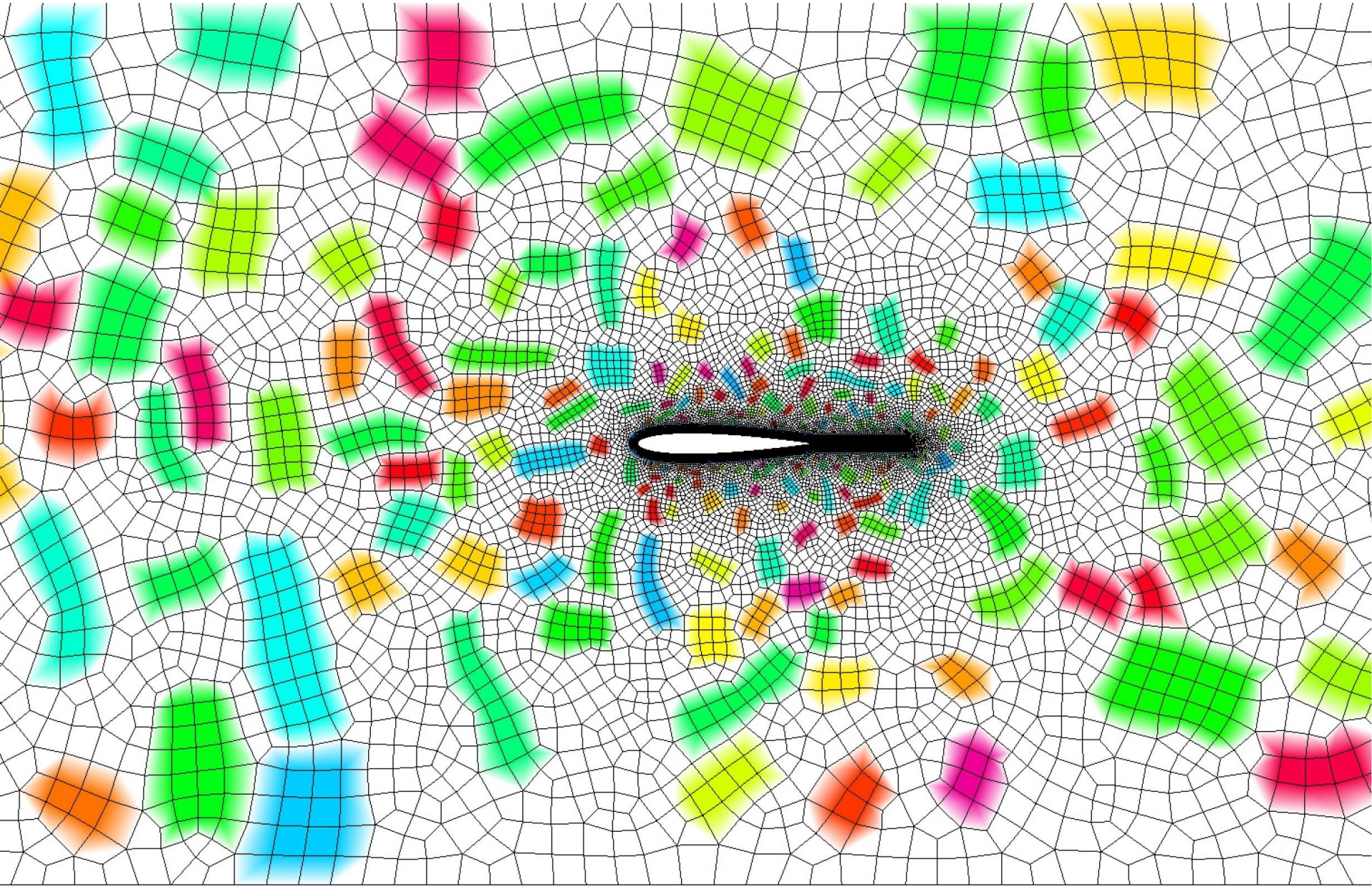
Conclusion

- We usually think of compilers as operating on code
- We have seen a couple of examples where it's profitable to build a compiler whose only input is data
- This idea applies not just to specific data values, but to any exploitable structure in the data

- For this we need a *language* for describing the exploitable structure

- Structured sparsity
- Symmetries
- Meshes
- Matrix approximation

Structure in unstructured meshes



Wael Al-Jishi, Crystal: Identifying and leveraging structure in quad meshes. *MEng thesis, Imperial College 2014*