Data-driven methods for Fluid Dynamics
I Forward Modelling

Claire Heaney

Applied Modelling and Computation Group (Department of Earth Science and Engineering) and Imperial-X, Imperial College London, UK

Forward Modelling

Computational Fluid Dynamics codes:

single CPU $\rightarrow$ workstation $\rightarrow$ CPU cluster $\rightarrow$ GPUs $\rightarrow$ latest AI processors

(MPI, OpenMP) (CUDA, OpenCL) ?

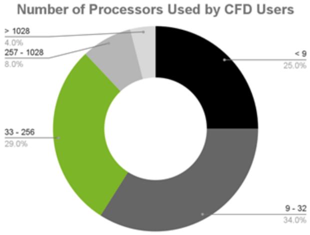


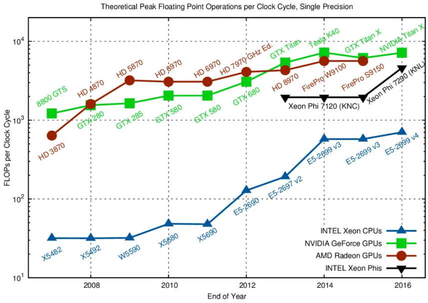
Figure 1. Resolved Analytics survey of CFD users

# processors	% of CFD users
<9	25%
9–32	34%
33–256	29%
257–1024	8%
>1024	4%

Navigating the CFD Software Landscape (a survey by Resolved Analytics)
The Computational Fluid Dynamics Revolution Driven by GPU Acceleration (a technical blog by NVIDIA)

Forward Modelling

CPUs vs GPUs: hard to compare, but code running on GPUs can show speed ups of 1–2 orders of magnitude.



Source: fast.ai

CPU ≈ Maserati, GPU ≈ a truck

The CPU (Maserati) can fetch small amounts of packages (3–4 passengers) in the RAM quickly whereas a GPU (the truck) is slower but can fetch large amounts of memory (∼20 passengers) in one turn.

Do you need a GPU in Deep Learning? (from Towards Data Science on Medium)

Forward Modelling

Machine Learning codes:

CPUs $\longleftrightarrow$ GPUs $\longleftrightarrow$ latest AI processors

```
import torch

# are GPUs available?
torch.cuda.is_available()

# a commonly used variable for handling the device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

model = MyModel(args)

# send the model to the device
model.to(device)
```

Forward Modelling

CFD models:	Machine Learning models:
adhere to governing equations	can lack explainability and interpretability
generalise well	can struggle to generalise
some rewriting involved when running on different platforms	platform agnostic (thanks to well-written libraries)
adjoints can be difficult to formulate and to code	differentiable models

Forward Modelling

CFD models:	Machine Learning models:
adhere to governing equations	can lack explainability and interpretability
generalise well	can struggle to generalise
some rewriting involved when running on different platforms	platform agnostic (thanks to well-written libraries)
adjoints can be difficult to formulate and to code	differentiable models

Forward Modelling

A couple of approaches that use functionality from AI libraries for CFD programming:

JAX-Fluids A fully-differentiable CFD solver for compressible two-phase flows (Bezgin et al (2023) 10.1016/j.cpc.2022.108527).
Finite volume method and level sets with `numpy` and `JAX`.

AI4PDEs or NN4PDEs Fully-differentiable CFD solver for incompressible flows (Chen et al (2024) 10.48550/arXiv.2402.17913).
Finite differences and finite elements with `pytorch`.

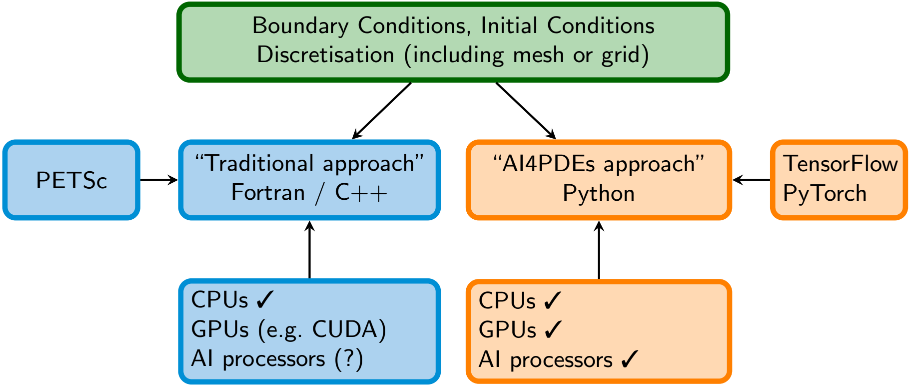
Forward Modelling

Aim: to implement numerical discretisations using AI libraries rather than standard approaches in Fortran / C++.

How: by defining the weights of convolutional neural networks according to a discretisation instead of calculating them during the process of "training". The discretisation coded in this way is exactly identical to the discretisation coded using standard approaches.

Why: (1) benefit from flexible deployment of code on CPUs, GPUs and new AI processors; (2) potential speed-up from using the latest powerful energy-efficient machines; (3) model differentiability for data assimilation, and uncertainty quantification; (4) elegant combination with surrogate models.

AI4PDEs: realising that many numerical discretisations can be written as discrete convolutions leads to the fact that a discretised system can be exactly written as and solved by a convolutional neural network with predefined weights (i.e. there is no need to train to find the weights).



Example: The Laplacian $\nabla^2 C(x, y)$

Control volume / finite difference approximation (based on Taylor series expansions):

$$\left.\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2}\right|_{i,j} \approx \frac{C_{i+1,j} - 2C_{i,j} + C_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2C_{i,j} + C_{i,j-1}}{\Delta y^2} \quad (1)$$

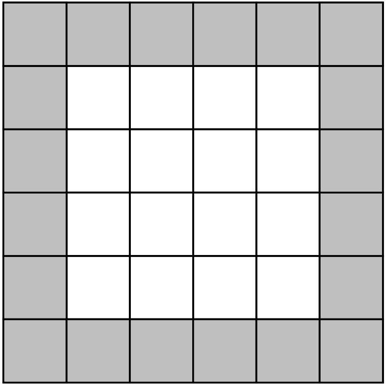
For $\Delta x = 1 = \Delta y$:

$$\left.\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2}\right|_{i,j} \approx (C_{i+1,j} + C_{i-1,j} + C_{i,j+1} + C_{i,j-1} - 4C_{i,j}) \quad (2)$$

Forward Modelling

$$\left.\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2}\right|_{i,j} \approx \sum_{\substack{\text{array}\\ \text{entries}}} \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \odot \begin{bmatrix} C_{i-1,j+1} & C_{i,j+1} & C_{i+1,j+1} \\ C_{i-1,j} & C_{i,j} & C_{i+1,j} \\ C_{i-1,j-1} & C_{i,j-1} & C_{i+1,j-1} \end{bmatrix} \quad (3)$$

C_{05}	C_{15}	C_{25}	C_{35}	C_{45}	C_{55}
C_{04}	C_{14}	C_{24}	C_{34}	C_{44}	C_{54}
C_{03}	C_{13}	C_{23}	C_{33}	C_{43}	C_{53}
C_{02}	C_{12}	C_{22}	C_{32}	C_{42}	C_{52}
C_{01}	C_{11}	C_{21}	C_{31}	C_{41}	C_{51}
C_{00}	C_{10}	C_{20}	C_{30}	C_{40}	C_{50}



Forward Modelling

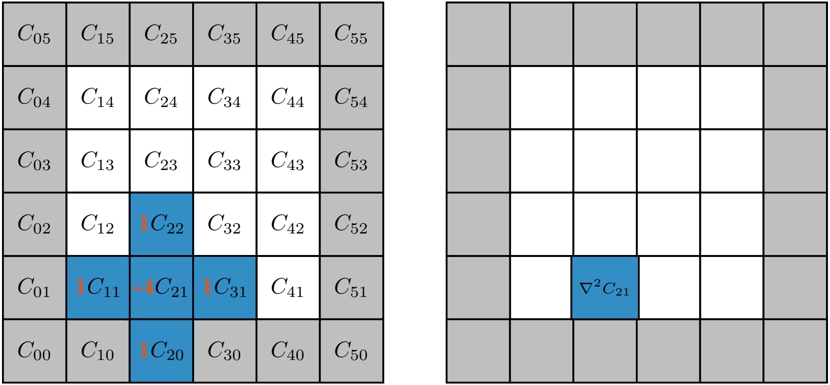
$$\left.\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2}\right|_{i,j} \approx \sum_{\substack{\text{array}\\\text{entries}}} \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \odot \begin{bmatrix} C_{i-1,j+1} & C_{i,j+1} & C_{i+1,j+1} \\ C_{i-1,j} & C_{i,j} & C_{i+1,j} \\ C_{i-1,j-1} & C_{i,j-1} & C_{i+1,j-1} \end{bmatrix} \quad (3)$$

C_{05}	C_{15}	C_{25}	C_{35}	C_{45}	C_{55}
C_{04}	C_{14}	C_{24}	C_{34}	C_{44}	C_{54}
C_{03}	C_{13}	C_{23}	C_{33}	C_{43}	C_{53}
C_{02}	$1C_{12}$	C_{22}	C_{32}	C_{42}	C_{52}
$1C_{01}$	$-4C_{11}$	$1C_{21}$	C_{31}	C_{41}	C_{51}
C_{00}	$1C_{10}$	C_{20}	C_{30}	C_{40}	C_{50}

	$\nabla^2 C_{11}$				

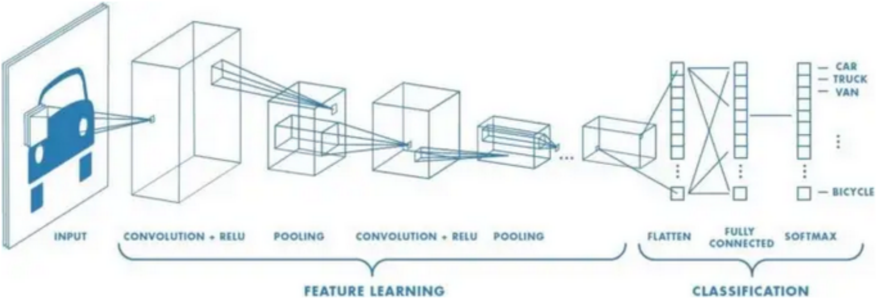
Forward Modelling

$$\left.\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2}\right|_{i,j} \approx \sum_{\substack{\text{array}\\\text{entries}}} \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \odot \begin{bmatrix} C_{i-1,j+1} & C_{i,j+1} & C_{i+1,j+1} \\ C_{i-1,j} & C_{i,j} & C_{i+1,j} \\ C_{i-1,j-1} & C_{i,j-1} & C_{i+1,j-1} \end{bmatrix} \quad (3)$$

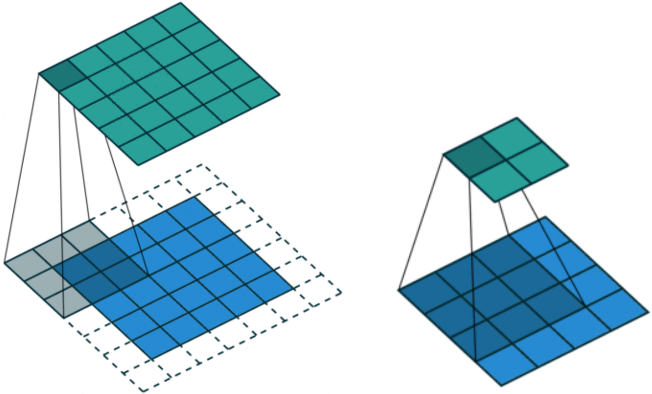


Forward Modelling

Convolutional neural network



https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53



from Dumoulin and Visin (2016) https://github.com/vdumoulin/conv_arithmetic, 10.48550/arXiv.1603.07285

Forward Modelling

Convolutional neural network

w_{02}	w_{12}	w_{22}
w_{01}	w_{11}	w_{21}
w_{00}	w_{10}	w_{20}

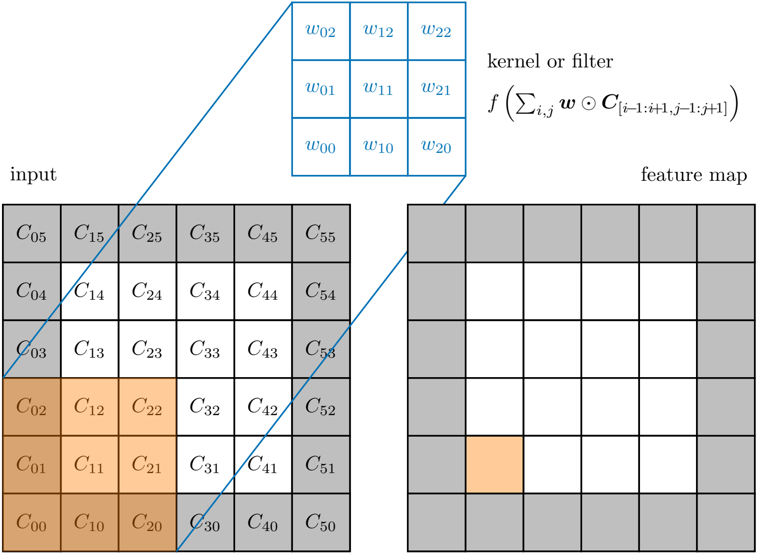
kernel or filter

$$f\left(\sum_{i,j} \boldsymbol{w} \odot \boldsymbol{C}_{[i-1:i+1,j-1:j+1]}\right)$$

input

C_{05}	C_{15}	C_{25}	C_{35}	C_{45}	C_{55}
C_{04}	C_{14}	C_{24}	C_{34}	C_{44}	C_{54}
C_{03}	C_{13}	C_{23}	C_{33}	C_{43}	C_{53}
C_{02}	C_{12}	C_{22}	C_{32}	C_{42}	C_{52}
C_{01}	C_{11}	C_{21}	C_{31}	C_{41}	C_{51}
C_{00}	C_{10}	C_{20}	C_{30}	C_{40}	C_{50}

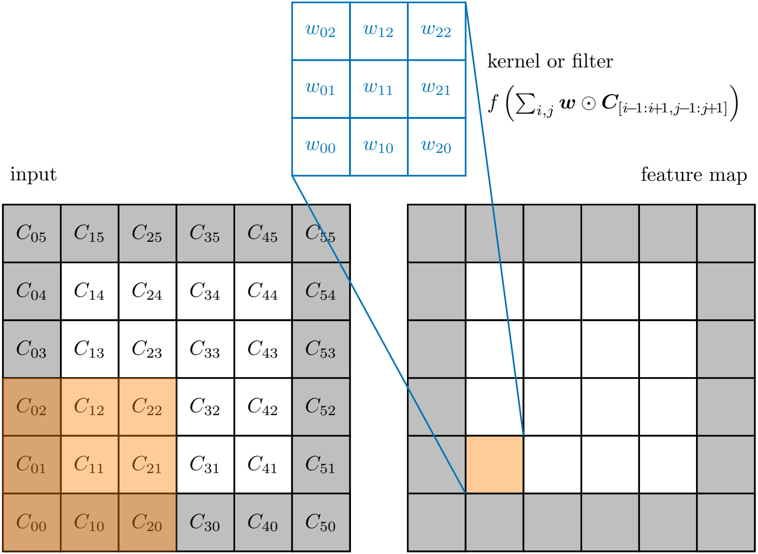
feature map



w_{02} w_{12} w_{22}
w_{01} w_{11} w_{21}
w_{00} w_{10} w_{20}
kernel or filter
$f\left(\sum_{i,j} \boldsymbol{w} \odot \boldsymbol{C}_{[i-1:i+1,j-1:j+1]}\right)$
input
feature map
C_{05} C_{15} C_{25} C_{35} C_{45} C_{55}
C_{04} C_{14} C_{24} C_{34} C_{44} C_{54}
C_{03} C_{13} C_{23} C_{33} C_{43} C_{53}
C_{02} C_{12} C_{22} C_{32} C_{42} C_{52}
C_{01} C_{11} C_{21} C_{31} C_{41} C_{51}
C_{00} C_{10} C_{20} C_{30} C_{40} C_{50}

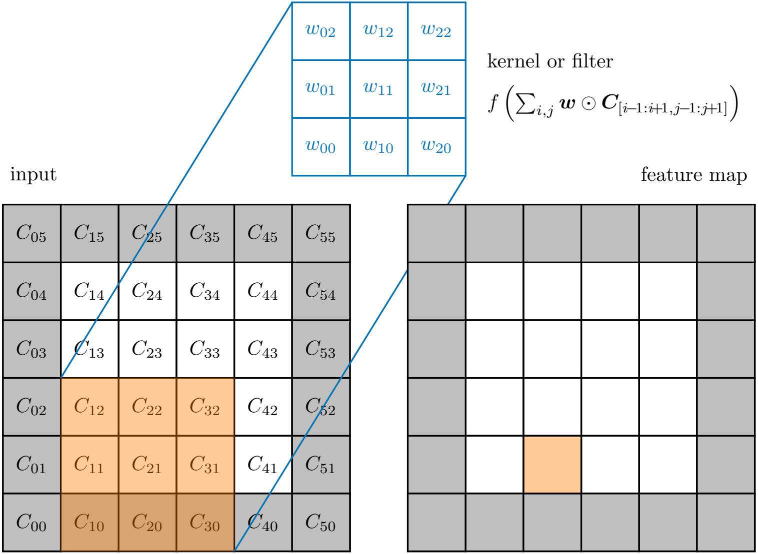
Forward Modelling

Convolutional neural network



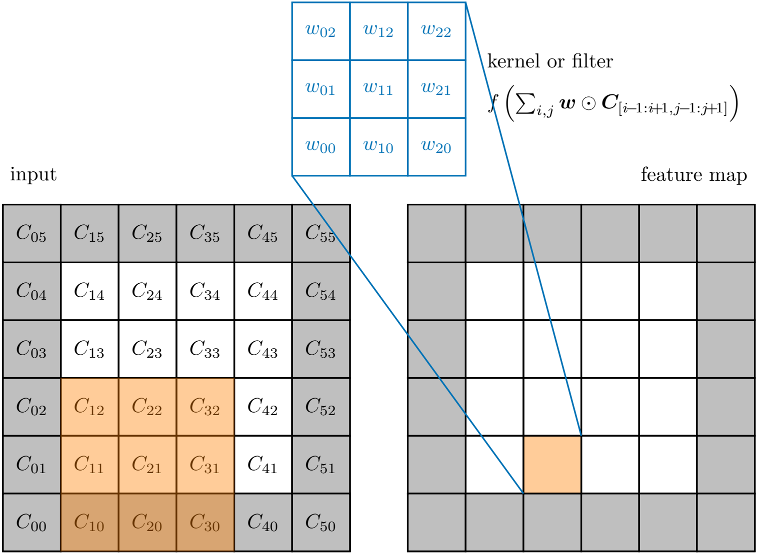
Forward Modelling

Convolutional neural network



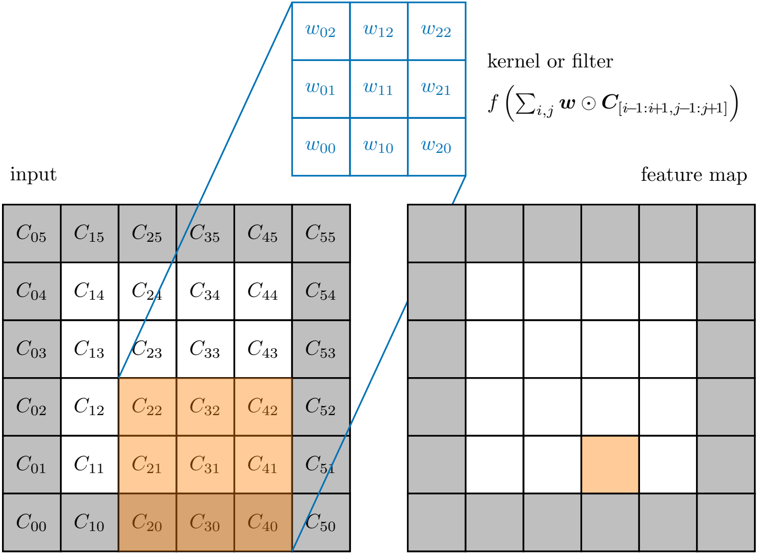
Forward Modelling

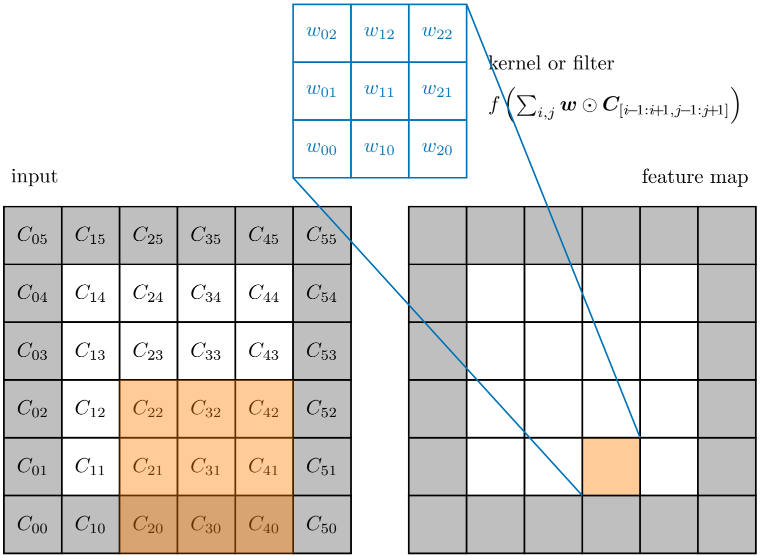
Convolutional neural network

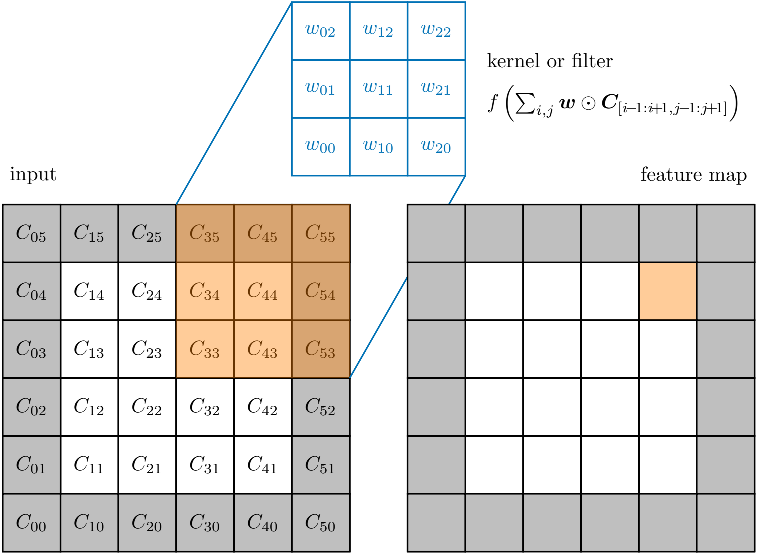


Forward Modelling

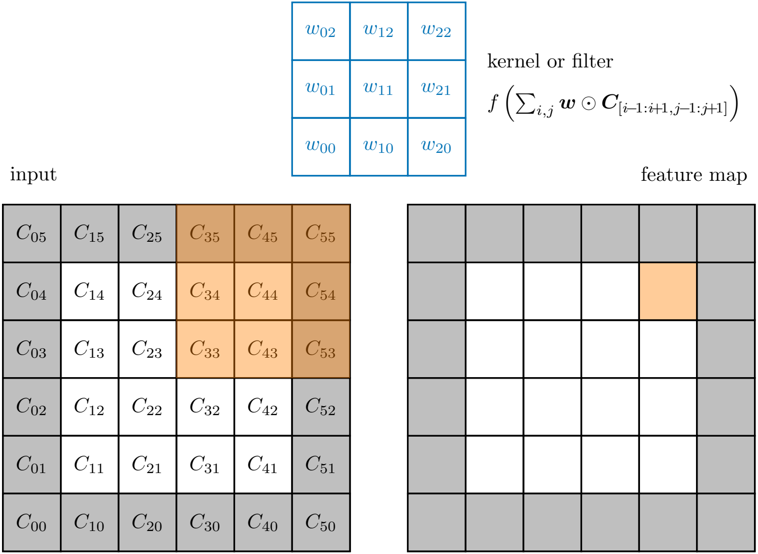
Convolutional neural network




w_{02} w_{12} w_{22}
w_{01} w_{11} w_{21}
w_{00} w_{10} w_{20}
kernel or filter
$f\left(\sum_{i,j} \boldsymbol{w} \odot \boldsymbol{C}_{[i-1:i+1,j-1:j+1]}\right)$
input
feature map
C_{05} C_{15} C_{25} C_{35} C_{45} C_{55}
C_{04} C_{14} C_{24} C_{34} C_{44} C_{54}
C_{03} C_{13} C_{23} C_{33} C_{43} C_{53}
C_{02} C_{12} C_{22} C_{32} C_{42} C_{52}
C_{01} C_{11} C_{21} C_{31} C_{41} C_{51}
C_{00} C_{10} C_{20} C_{30} C_{40} C_{50}



kernel or filter
$f\left(\sum_{i,j} \boldsymbol{w} \odot \boldsymbol{C}_{[i-1:i+1,j-1:j+1]}\right)$
input
feature map



$$f\left(\sum_{i,j} \boldsymbol{w} \odot \boldsymbol{C}_{[i-1:i+1,\,j-1:j+1]}\right)$$

Forward Modelling

Convolutional neural network

0	**1**	**0**
1	**-4**	**1**
0	**1**	**0**

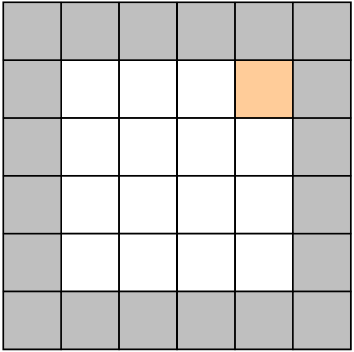
kernel or filter

$$f\left(\sum_{i,j} \boldsymbol{w} \odot \boldsymbol{C}_{[i-1:i+1,j-1:j+1]}\right)$$

input

C_{05}	C_{15}	C_{25}	C_{35}	C_{45}	C_{55}
C_{04}	C_{14}	C_{24}	C_{34}	C_{44}	C_{54}
C_{03}	C_{13}	C_{23}	C_{33}	C_{43}	C_{53}
C_{02}	C_{12}	C_{22}	C_{32}	C_{42}	C_{52}
C_{01}	C_{11}	C_{21}	C_{31}	C_{41}	C_{51}
C_{00}	C_{10}	C_{20}	C_{30}	C_{40}	C_{50}

feature map



$$\text{Solve} \quad \frac{\partial C}{\partial t} - \kappa \nabla^2 C = 0 \tag{4}$$

$$\text{Predictor} \quad \frac{C^{n+1,*} - C^n}{\Delta t} \quad - \quad \kappa \nabla^2 C^n = 0 \tag{5}$$

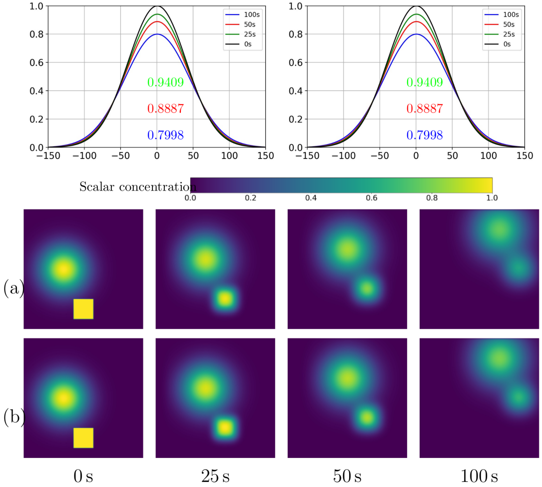
$$\text{Corrector} \quad \frac{C^{n+1} - C^n}{\Delta t} \quad - \quad \frac{\kappa}{2} \nabla^2 \left(C^{n+1,*} + C^n \right) = 0 \tag{6}$$

Predictor:

$$\frac{C_{i,j}^{n+1,*} - C_{i,j}^n}{\Delta t} - \kappa \sum_{\substack{\text{array}\\\text{entries}}} \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \odot \begin{bmatrix} C_{i-1,j+1}^n & C_{i,j+1}^n & C_{i+1,j+1}^n \\ C_{i-1,j}^n & C_{i,j}^n & C_{i+1,j}^n \\ C_{i-1,j-1}^n & C_{i,j-1}^n & C_{i+1,j-1}^n \end{bmatrix} = 0 \tag{7}$$

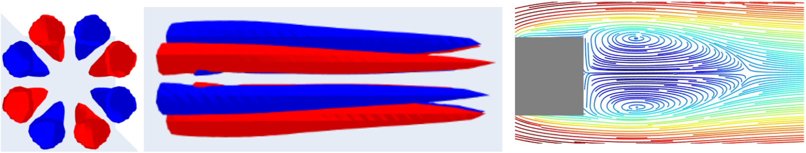
Forward Modelling

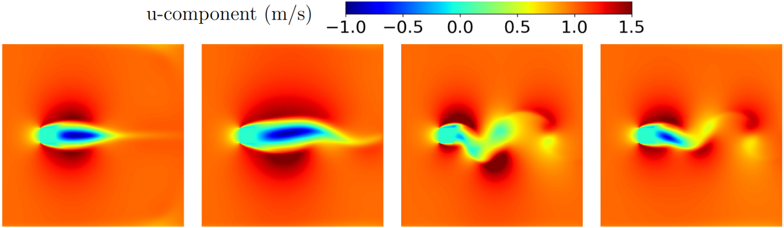
Results - Advection diffusion



Forward Modelling

Results - Flow past a bluff body, $Re = 200$





The Strouhal number is calculated to be 0.142, close to values in the literature of 0.147. The length and diameter of the re-circulation area are also both close to values in the literature.

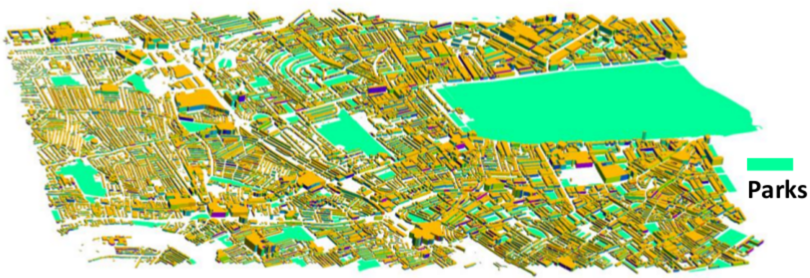
Computational efficiency:

	128^3 nodes	256^3 nodes	512^3 nodes
Intel Xeon 2.3Ghz CPU	165 s	1275 s	14 376 s
NVIDIA Tesla T4 GPU *(2560 CUDA cores)*	3 s	11 s	34 s

Linear finite elements with nodes as quoted in table, each running for five time steps with 20 multigrid iterations per time step.

Forward Modelling

South Kensington area

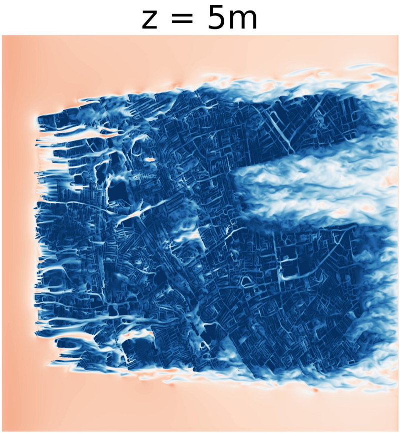


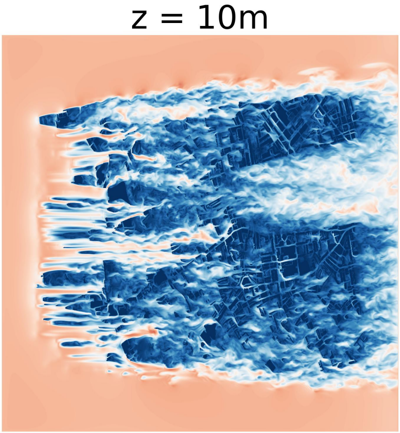
CFD details

Quadratic finite elements
Domain size: 4096m × 5120m × 256m (resolution $\sim 1\,\mathrm{m}^3$)
2 billion nodes
4 GPUs: NVIDIA RTX A100
1 hour of time takes 5 hours computation time

Forward Modelling

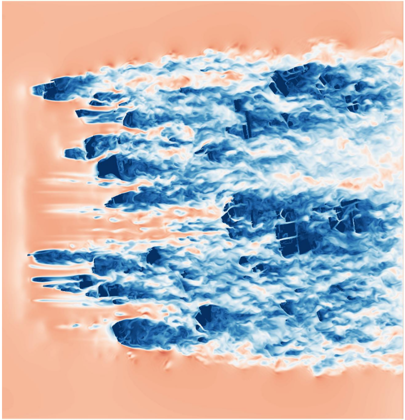
Horizontal cross-sections



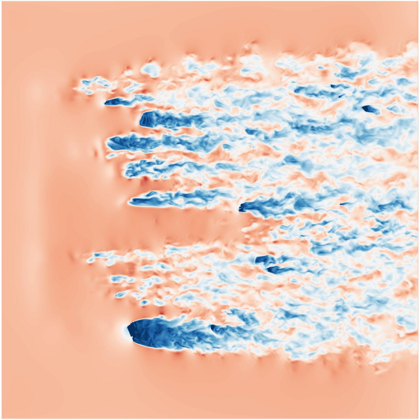


Horizontal cross-sections

z = 15m

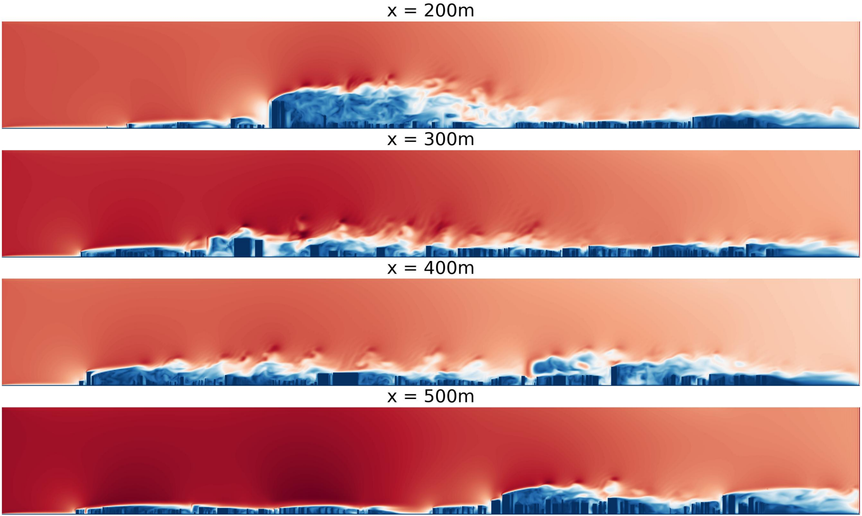


z = 30m



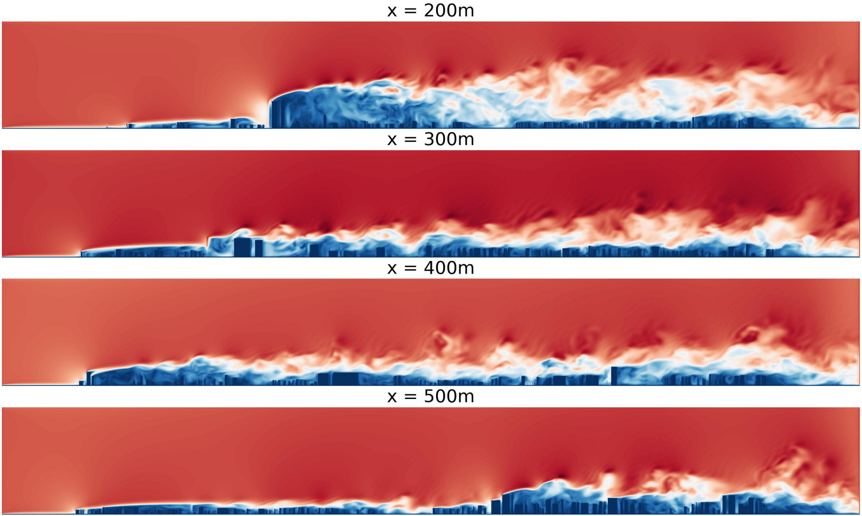
Forward Modelling

Vertical cross-sections



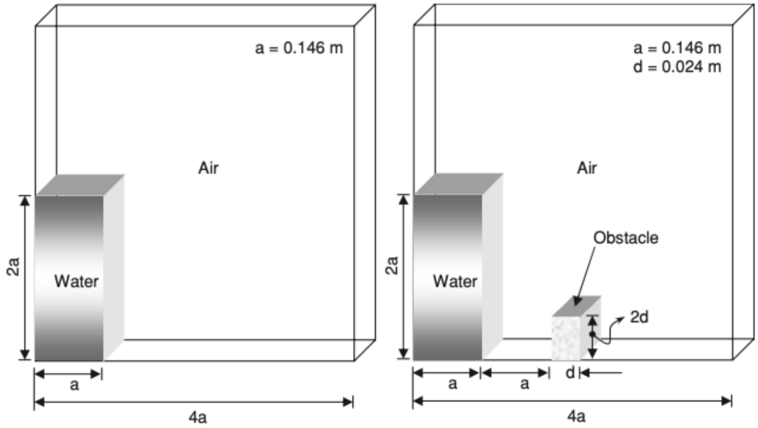
Forward Modelling

Vertical cross-sections

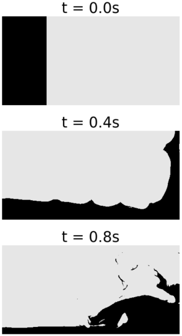


Forward Modelling

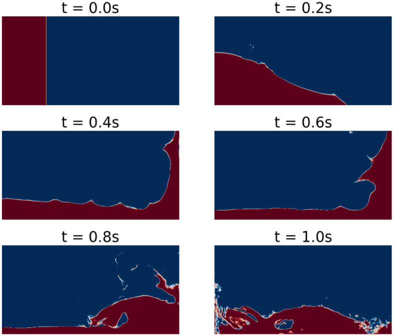
Results - Collapsing water column



Results - Collapsing water column

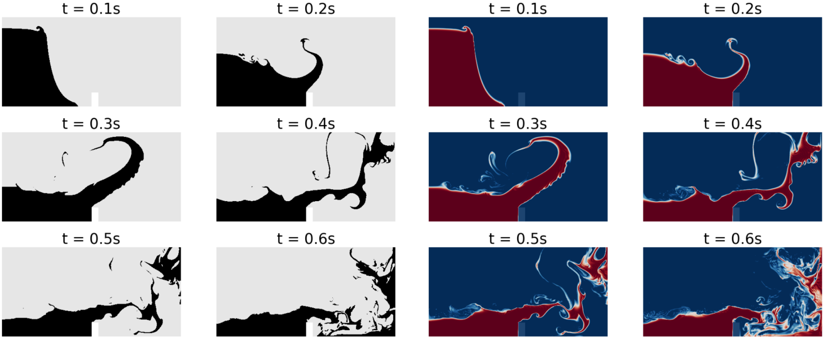


(a)



(b)

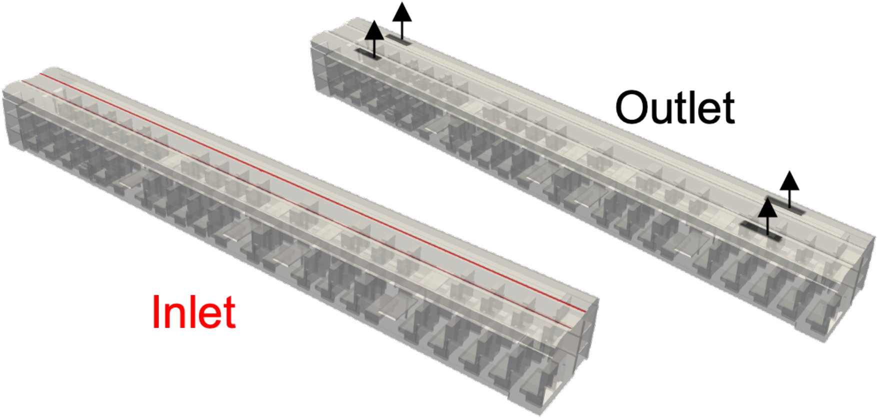
Results - Collapsing water column



(a) (b)

Forward Modelling

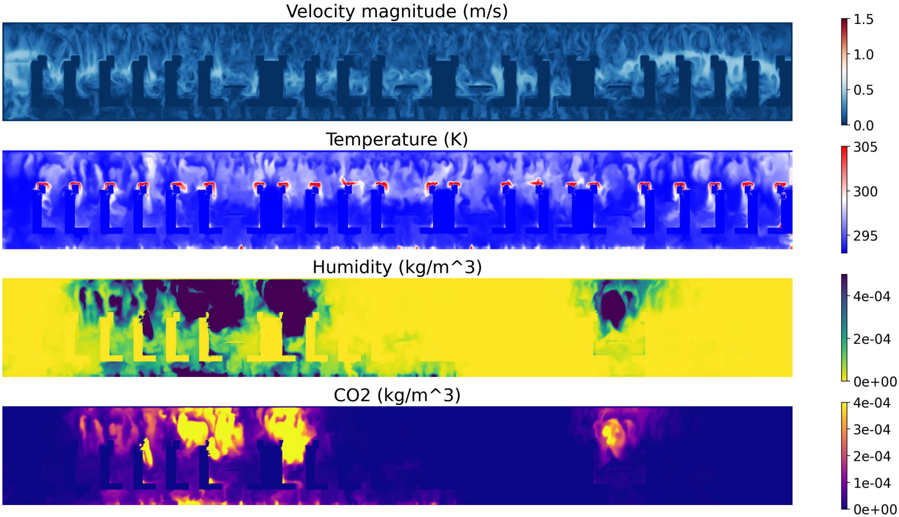
Results - Train carriage



128 million nodes
Quadratic finite elements
22.5 m by 3 m by 3 m (1 cm resolution)

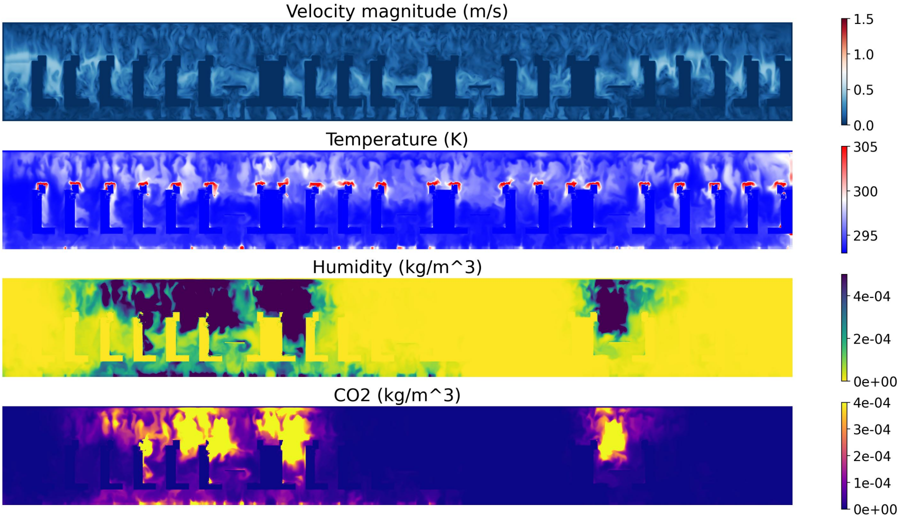
Forward Modelling

Results - Train carriage



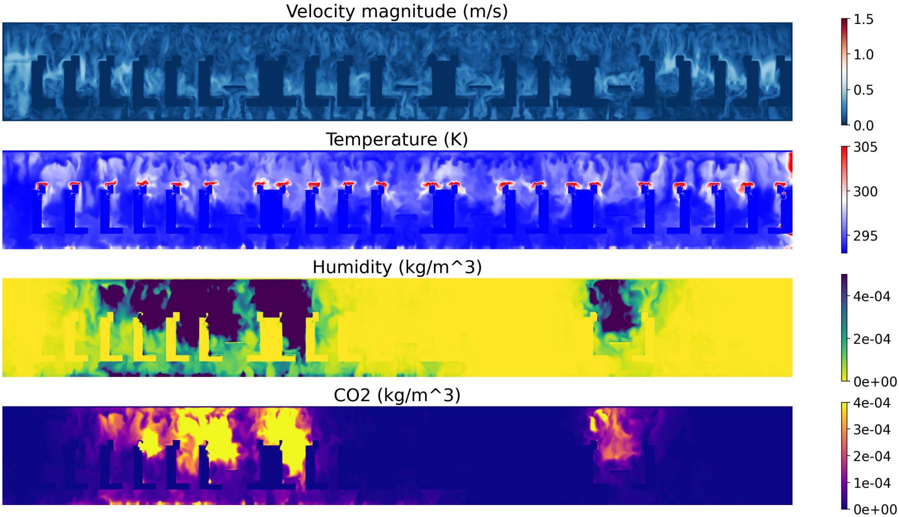
Forward Modelling

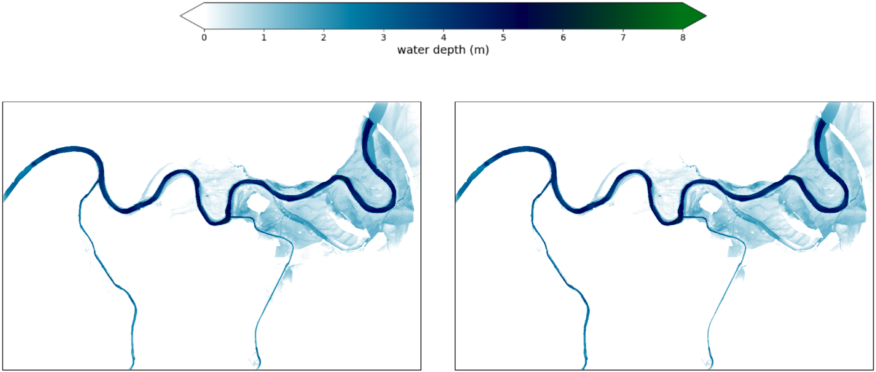
Results - Train carriage



Forward Modelling

Results - Train carriage





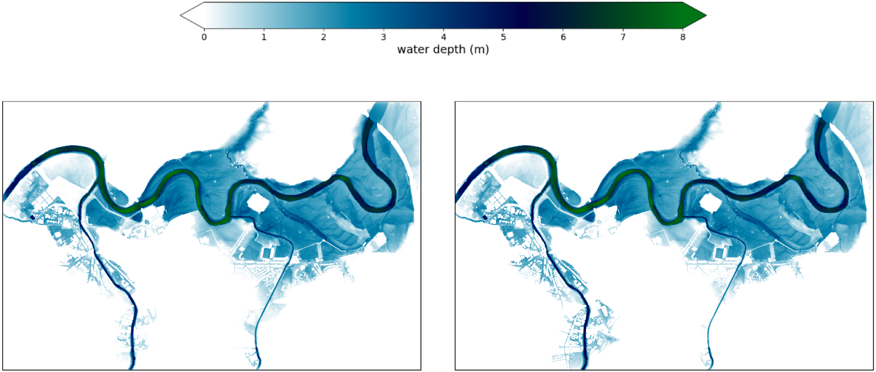
951 by 611 nodes

Domain of 4.75 km by 3.05 km $\Delta x = 5\,\mathrm{m}$, $\Delta t = 0.5\,\mathrm{s}$

Snapshots taken at 14, 28, 42 and 56 hours (linear and quadratic elements).

Forward Modelling

Results - Shallow water equations



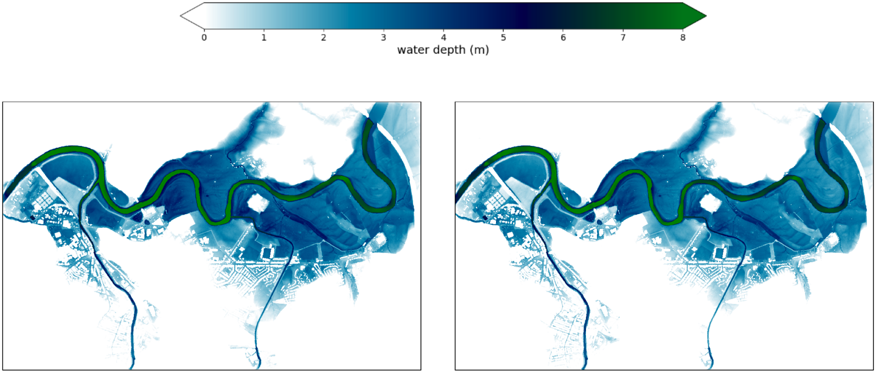
951 by 611 nodes

Domain of 4.75 km by 3.05 km $\Delta x = 5\,\mathrm{m}$, $\Delta t = 0.5\,\mathrm{s}$

Snapshots taken at 14, 28, 42 and 56 hours (linear and quadratic elements).

Forward Modelling

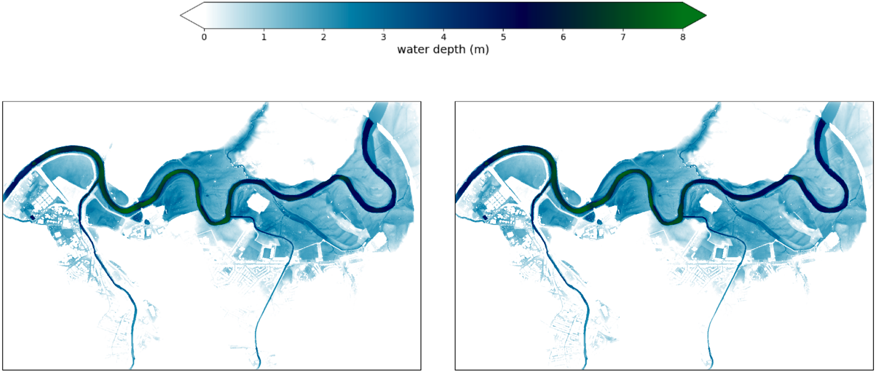
Results - Shallow water equations



951 by 611 nodes
Domain of 4.75 km by 3.05 km $\Delta x = 5\,\text{m}$, $\Delta t = 0.5\,\text{s}$
Snapshots taken at 14, 28, 42 and 56 hours (linear and quadratic elements).

Forward Modelling

Results - Shallow water equations

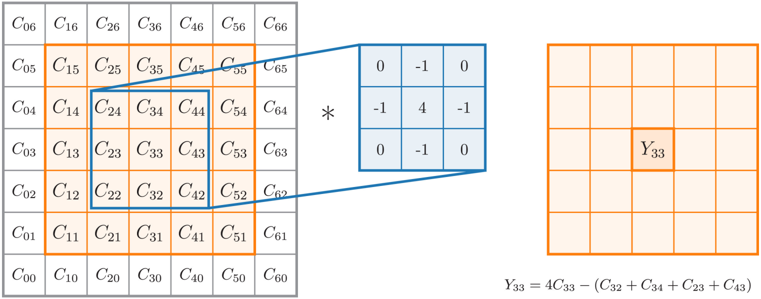


951 by 611 nodes

Domain of 4.75 km by 3.05 km $\Delta x = 5\,\mathrm{m}$, $\Delta t = 0.5\,\mathrm{s}$

Snapshots taken at 14, 28, 42 and 56 hours (linear and quadratic elements).

Solution field on a grid | Central difference stencil | Diffusion discretisation



$$Y_{33} = 4C_{33} - (C_{32} + C_{34} + C_{23} + C_{43})$$

Pixel values of a 2D image | Filter | Feature map

Advantages:

- flexibility — the same code runs on CPUs, GPUs, AI processors
- speed — potentially extremely fast
- concise and readible — well-written libraries with abstraction
- usability — code more easier to modify for students and collaborators
- combination with (trained) surrogate models
- digital twins

Disadvantages:

- certain operations may not be easily written as (fixed) stencils
- certain operations might not be efficient

Progress includes

- benchmarks for single-phase incompressible flow [3]
- urban flow demonstration (5km by 4km area in South Kensington)
- multiphase flow equations [4]
- shallow water equations
- neutron diffusion equation [5]
- Boltzmann transport equation [6]
- variable resolution
- unstructured meshes (with graph neural network and space-filling curves) [7]
- inverse problems — electrical resistivity inversion
- inverse problems — seismic full waveform inversion

Forward Modelling

Previous work [1,2] and references for previous slide

[1] Zhao et al., (2020) A TensorFlow-based new high-performance computational framework for CFD, *Journal of Hydrodynamics* 32(4):735–746, 10.1007/s42241-020-0050-0.

[2] Wang et al., (2022) A TensorFlow simulation framework for scientific computing of fluid flows on tensor processing units, *Computer Physics Communications*, 274:108292, 10.1016/j.cpc.2022.108292.

[3] Chen, Heaney and Pain (2023) Using AI libraries for Incompressible Computational Fluid Dynamics *arXiv preprints* 10.48550/arXiv.2402.17913

[4] Chen, Heaney, Gomes, Matar, Pain (2024) Solving the discretised multiphase flow equations with interface capturing on structured grids using machine learning libraries, *CMAME* 426:116974 10.1016/j.cma.2024.116974.

[5] Phillips et al. (2023) Solving the Discretised Neutron Diffusion Equations using Neural Networks, *IJNME*, doi: 10.48550/arXiv.2301.09939.

[6] Phillips et al. (2023) Solving the Discretised Boltzmann Transport Equations using Neural Networks: Applications in Neutron Transport, *arXiv preprint*, 10.48550/arXiv.2301.09991.

[7] Li et al. (2024) Implementing the Discontinuous-Galerkin Finite Element Method Using Graph Neural Networks, *SSRN preprint* 10.2139/ssrn.4698813.

Christopher Pain, Boyang Chen, and others in Department of Earth Science and Engineering, and Imperial-X.

The AI4PDEs team at the Schmidt Sciences hackathon (Oxford, June 2024).