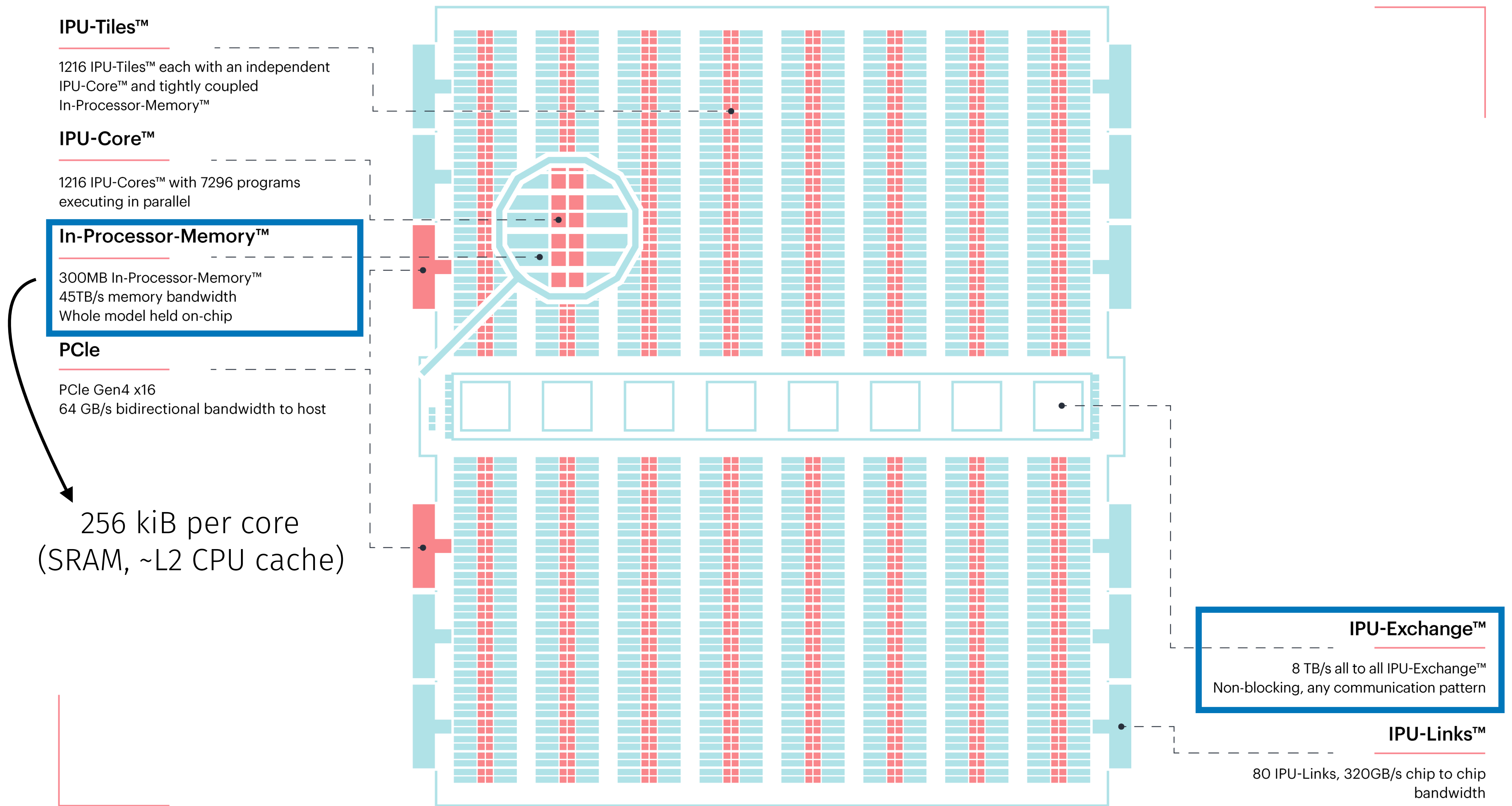
Progress with IPUs

University of BRISTOL

Lakshan Ram Madhan Mohan, Alexander Marshall,
Samuel Maddrell-Mander, **Daniel O'Hanlon**,
Konstantinos Petridis, Jonas Rademacker

What?

'I think we need to move toward a different kind of computer. Fortunately I have one here…' - Geoff Hinton

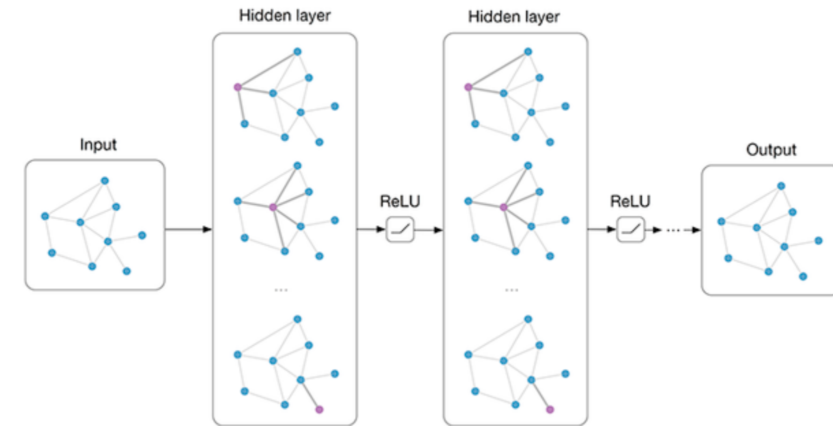


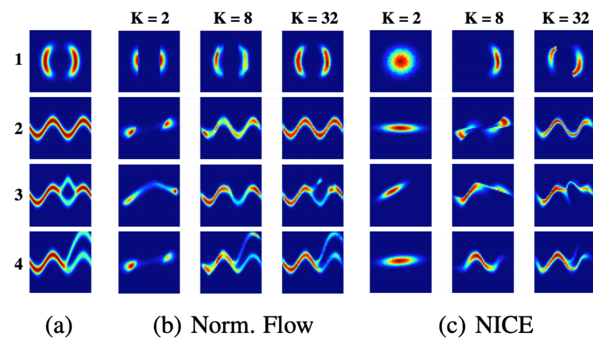
32 TFLOPS/s single precision (125 TFLOPS/s mixed single/half precision)

Why?

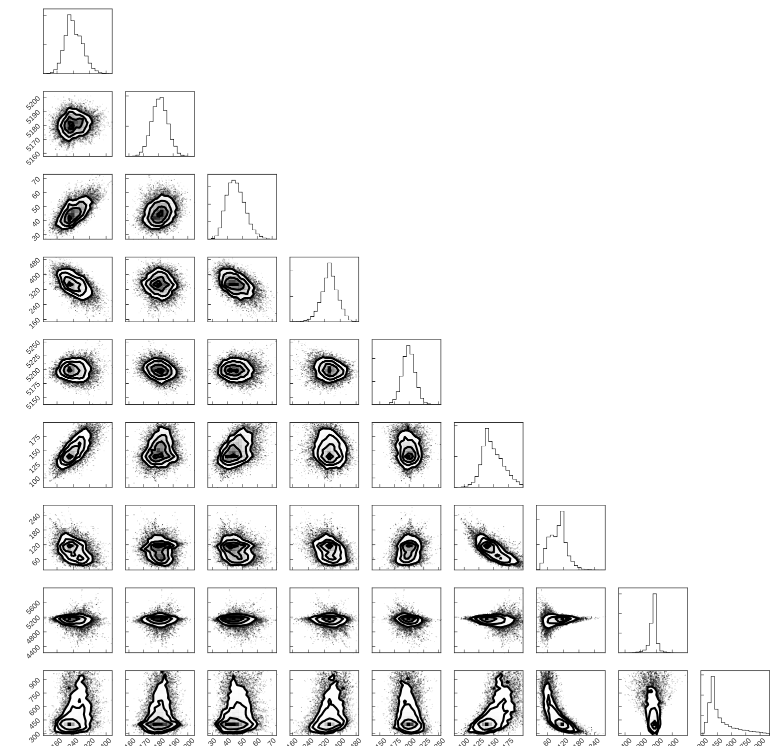
SIMD is great if your problem is pure SIMD!

- A trend in machine learning (the main IPU market) for more ‘awkward’ configurations
  - Graph neural networks
  - Networks with embedded control flow
  - Normalising/autoregressive flows
  - Who knows what in the next few years



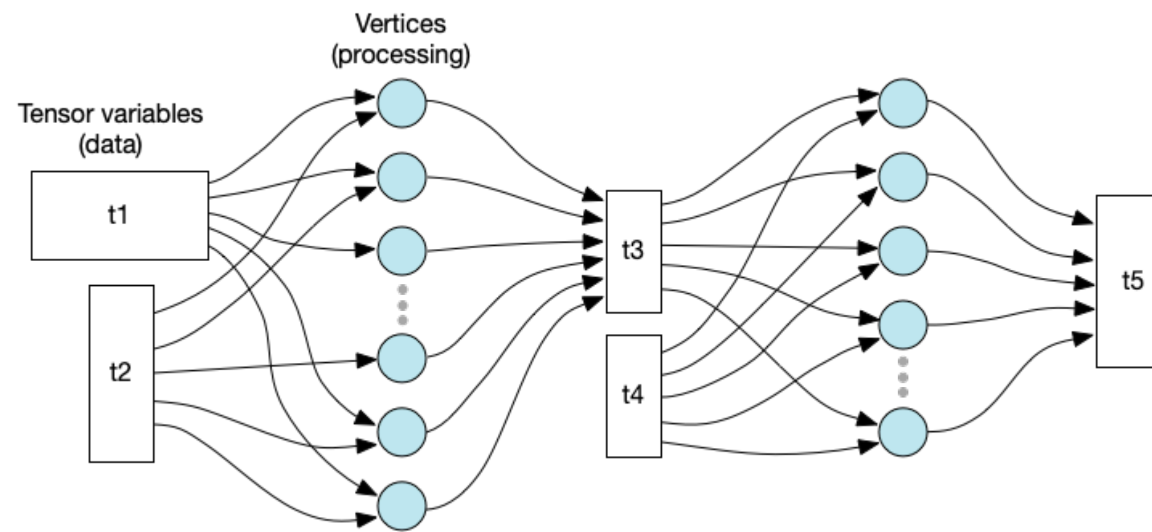


- As well as other applications that have non-trivial program/data flow
  - Markov chain Monte Carlo
  - Monte Carlo simulations (inc. ray tracing)
- Problems like these might benefit from MIMD architecture



How?

- IPU runs a static compute graph, which describes all data flow, conditional execution, synchronisation, etc



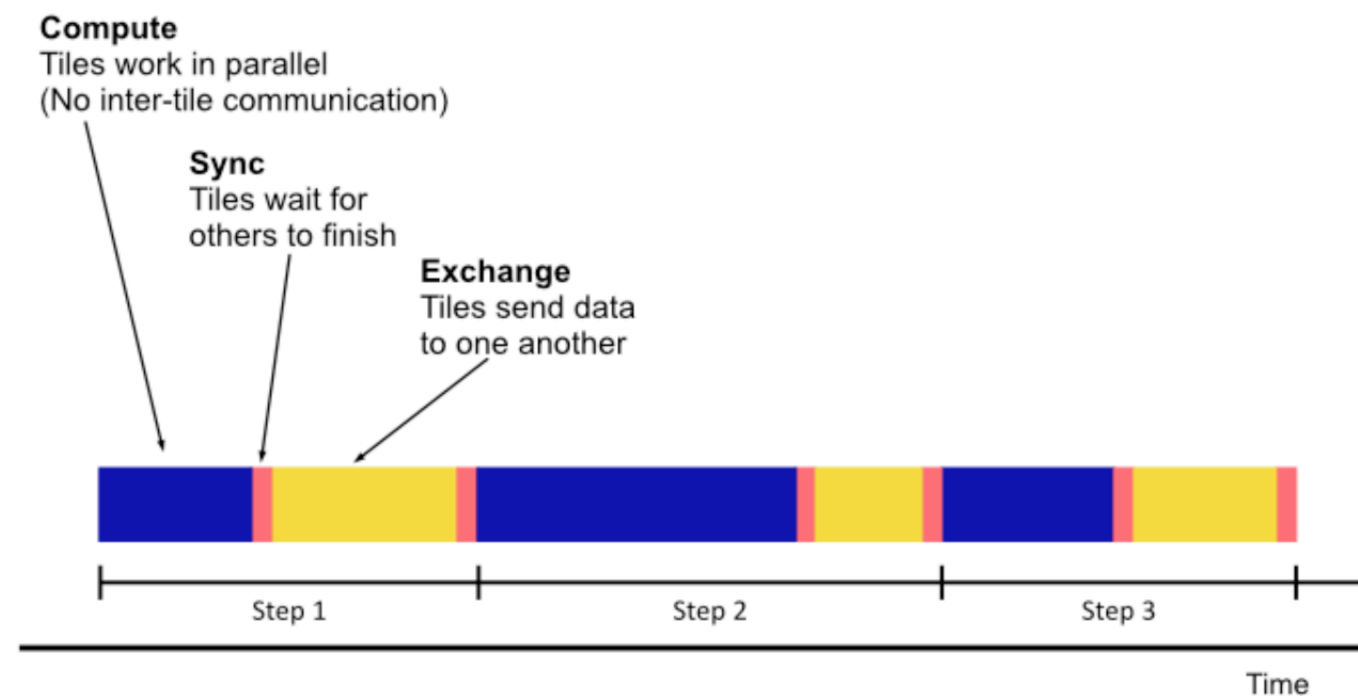
Defined in a declarative way and compiled, much like (pre-eager) TensorFlow

A few ways to build the graph:

- Easiest is with TensorFlow - supports most operations with explicit device preparation or compute ‘strategy’ (some optimised-for-IPU neural network layers)
- C++ ‘Poplar’ interface with full manual control over data placement and execution
- Execution of imported ONNX models in ‘PopART’
- PyTorch (haven’t tried it, became available on *Friday*)
- Assembly, if you’re insane

How?

- Large benefit from explicit mapping of graph to individual hardware cores, and fast all-to-all core connections is that it enables bulk-synchronous parallel execution

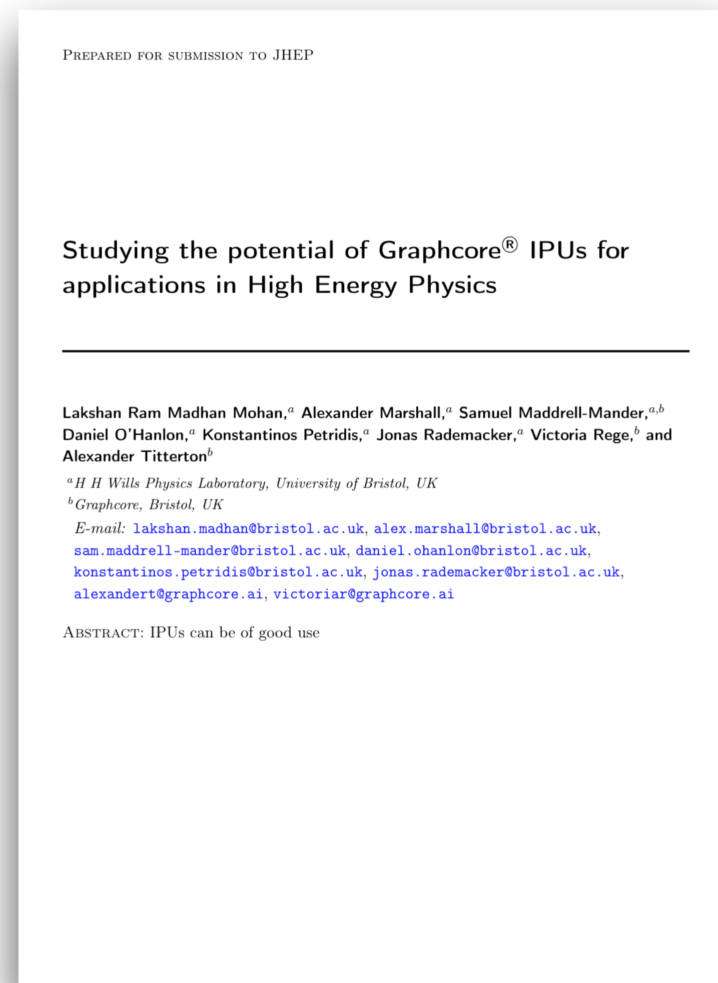


- Upshot: No mutexes, no blocking, no race conditions, no warp divergence, no bank conflicts....
- All you need is the graph, and less understanding of the underlying architecture
- Each core is a real, independent core that can run independent code on independent data (MIMD), modulo the requirement that *all cores* must synchronise at the same time

Benchmarks

- Graphcore have generously loaned us the use of a remote server with a set of IPU boards, to run experiments/benchmarks on, and evaluate the use cases in HEP.

- Clearly these can be used wherever neural networks are used, but also elsewhere.
- Currently have a paper in preparation that explores the potential of various HEP-centric use cases.
- Includes performance of neural networks for tagging, GANs for fast simulation, etc, in addition to 'non-ML' workloads like likelihood fitting and track reconstruction.



PREPARED FOR SUBMISSION TO JHEP

Studying the potential of Graphcore® IPUs for applications in High Energy Physics

Lakshan Ram Madhan Mohan,[a] Alexander Marshall,[a] Samuel Maddrell-Mander,[a,b] Daniel O'Hanlon,[a] Konstantinos Petridis,[a] Jonas Rademacker,[a] Victoria Rege,[b] and Alexander Titterton[b]

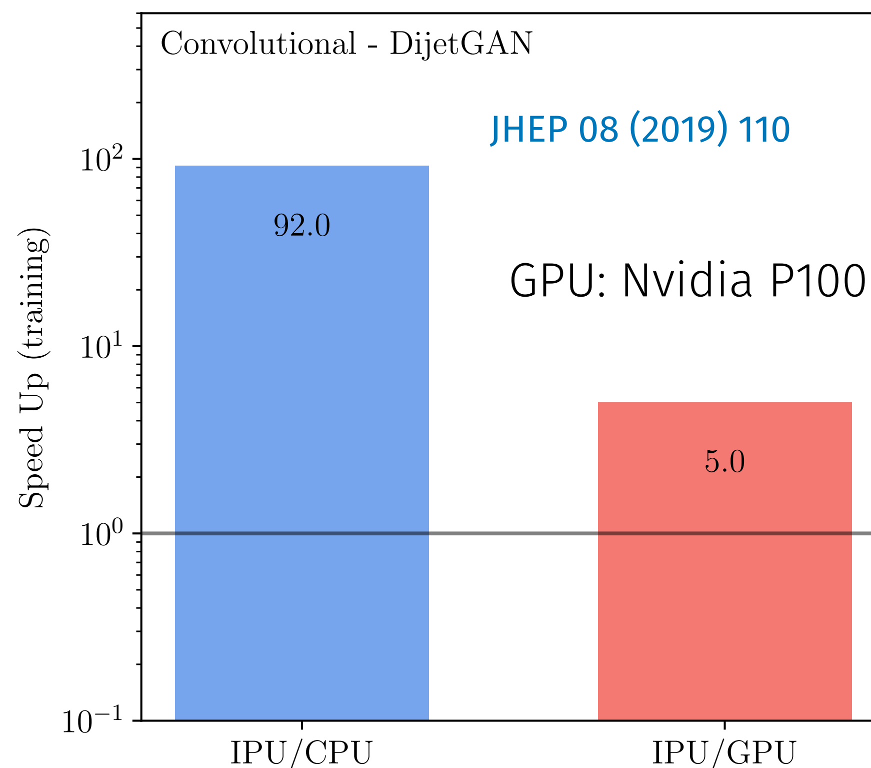
[a] *H H Wills Physics Laboratory, University of Bristol, UK*
[b] *Graphcore, Bristol, UK*
E-mail: lakshan.madhan@bristol.ac.uk, alex.marshall@bristol.ac.uk, sam.maddrell-mander@bristol.ac.uk, daniel.ohanlon@bristol.ac.uk, konstantinos.petridis@bristol.ac.uk, jonas.rademacker@bristol.ac.uk, alexandert@graphcore.ai, victoriar@graphcore.ai

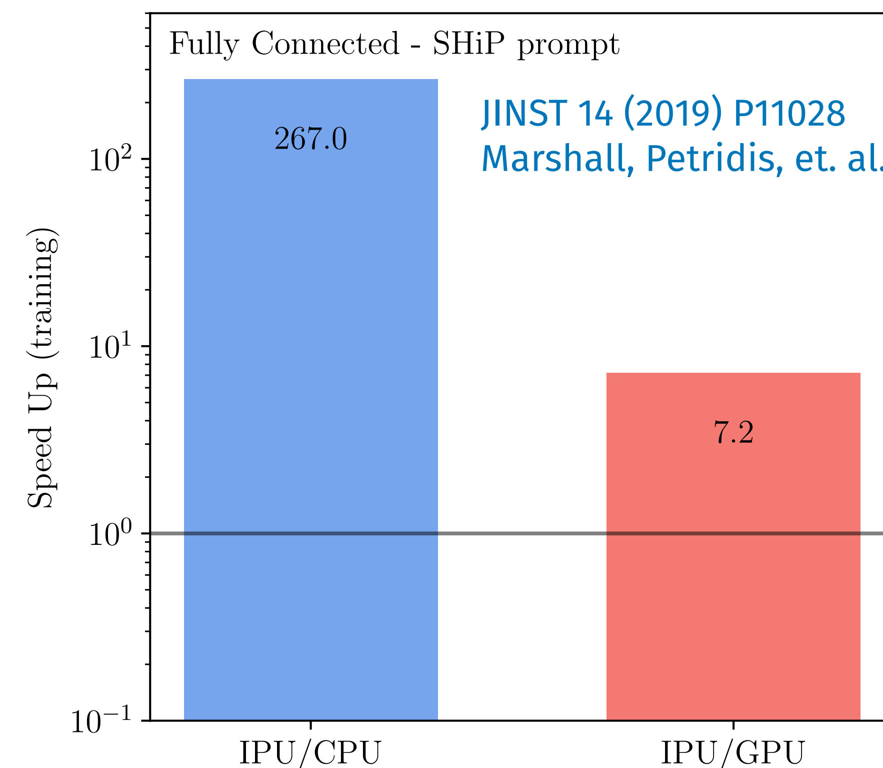
ABSTRACT: IPUs can be of good use

Benchmarks - GANs

Generative adversarial networks: coupled pair generative and discriminative neural networks participate in a two player 'game' - aim for generative to learn to generate synthetic data points to fool the discriminator

Seem useful for fast approximate generation of physics responses, and terrifying pictures of 'cats'





Studied a few examples in the HEP literature - considerable speed up achieved for training...

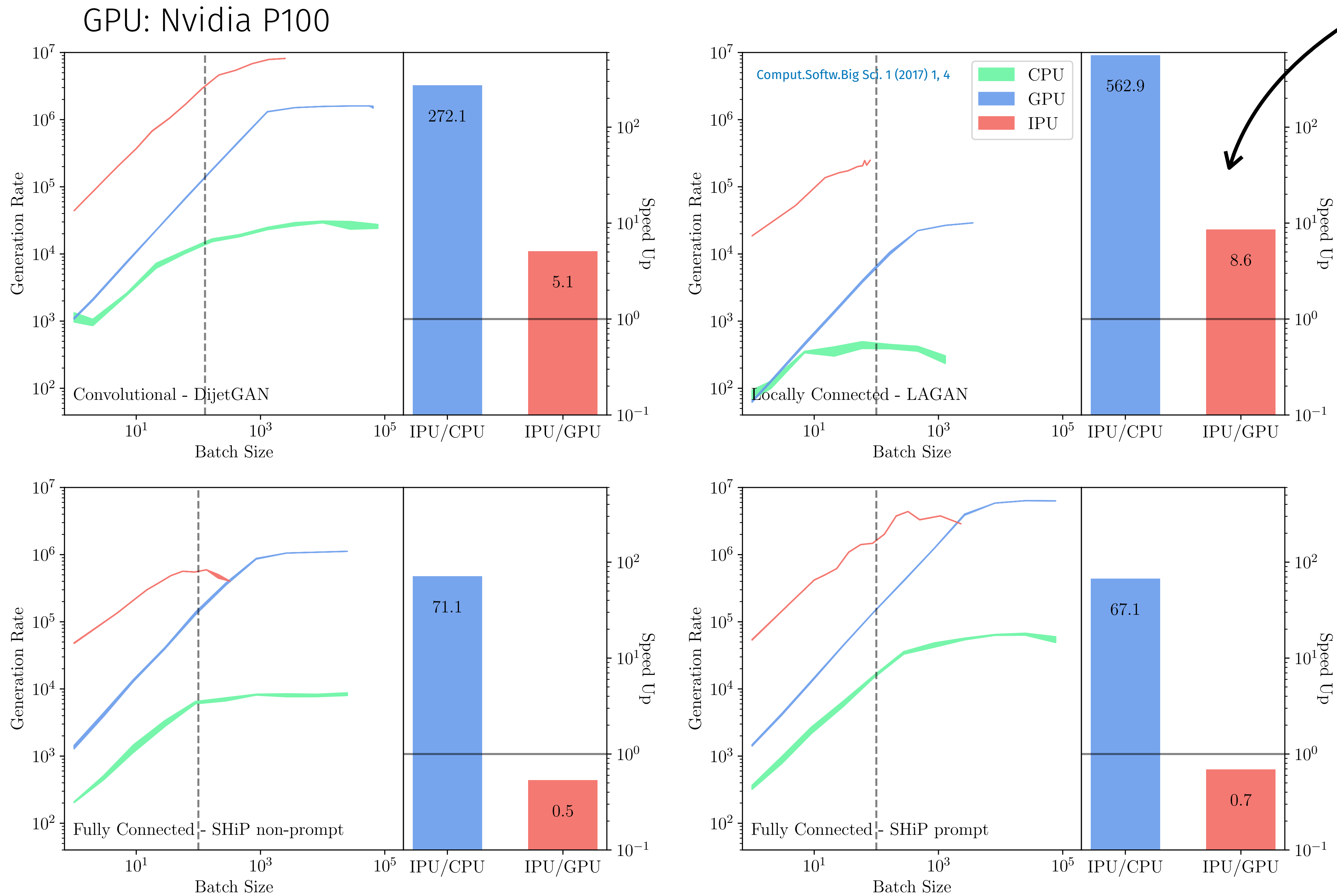
Benchmarks - GANs

For generation, the picture is more mixed, dependent on the GAN architecture.

The IPU prefers smaller batch sizes, and runs out of memory before the GPU

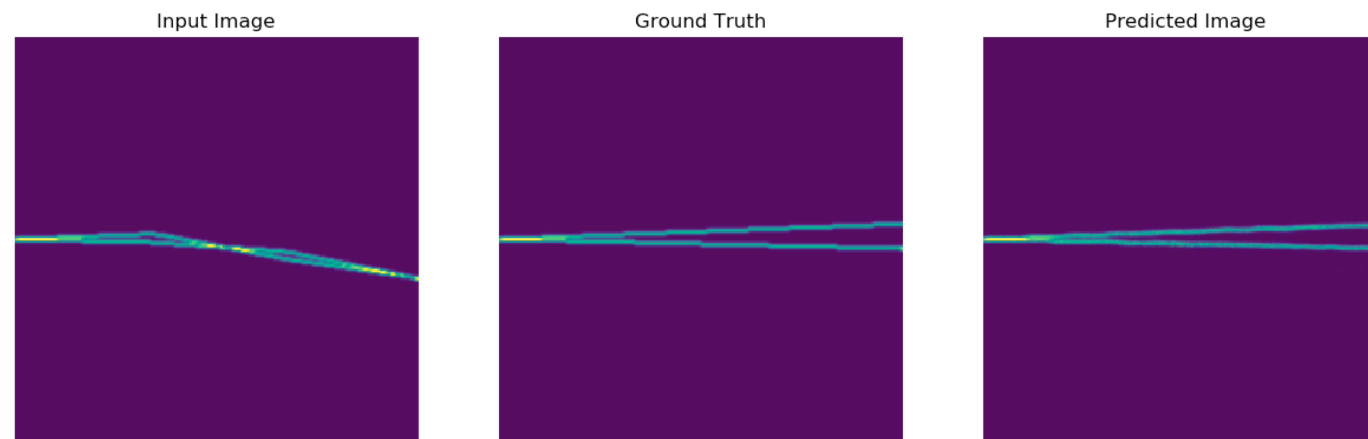
Value at best working point for each

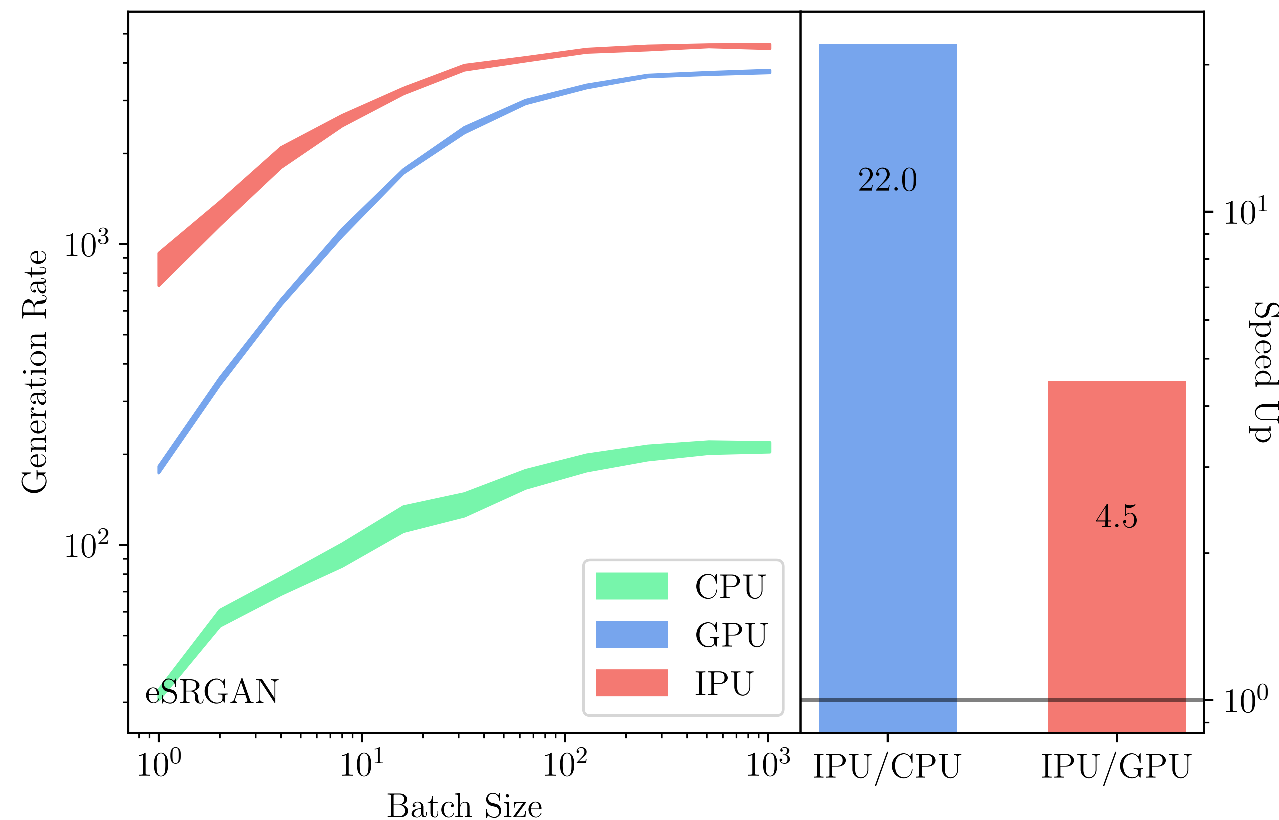
GPU: Nvidia P100



Benchmarks - GANs

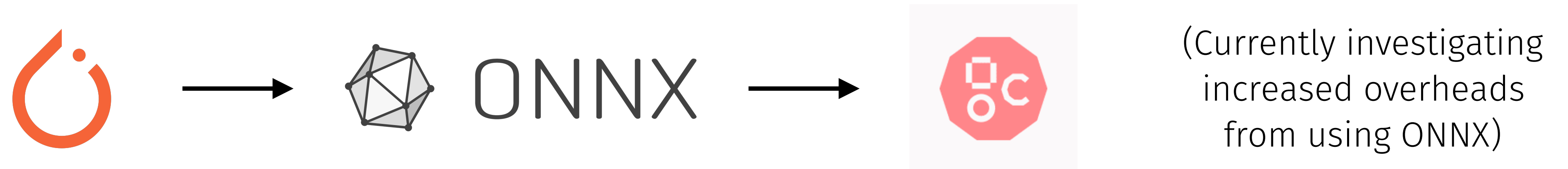
Can also be used for conditional generation, for example to generate multiple-scattering corrected tracks (with super resolution GAN)



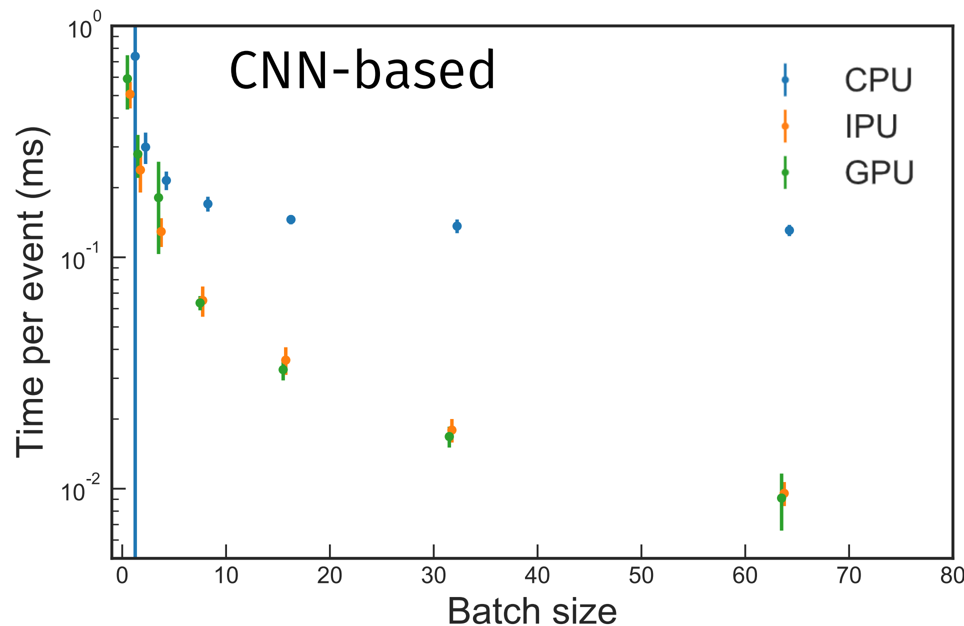


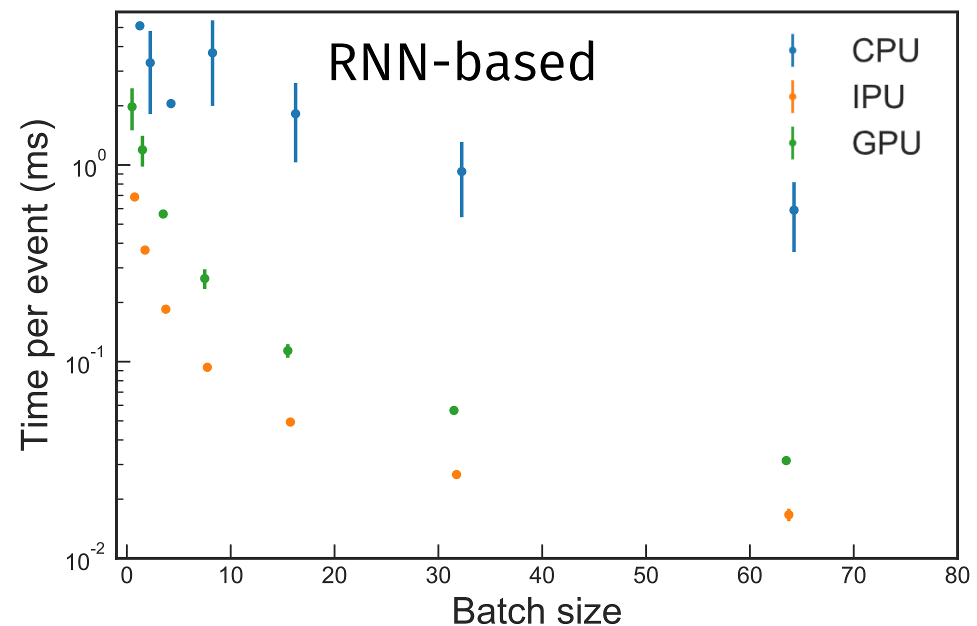
Benchmarks - tagging

- Also look into smaller (~few thousand parameter) networks for online B/jet tagging, where execution speed is important



(Currently investigating increased overheads from using ONNX)





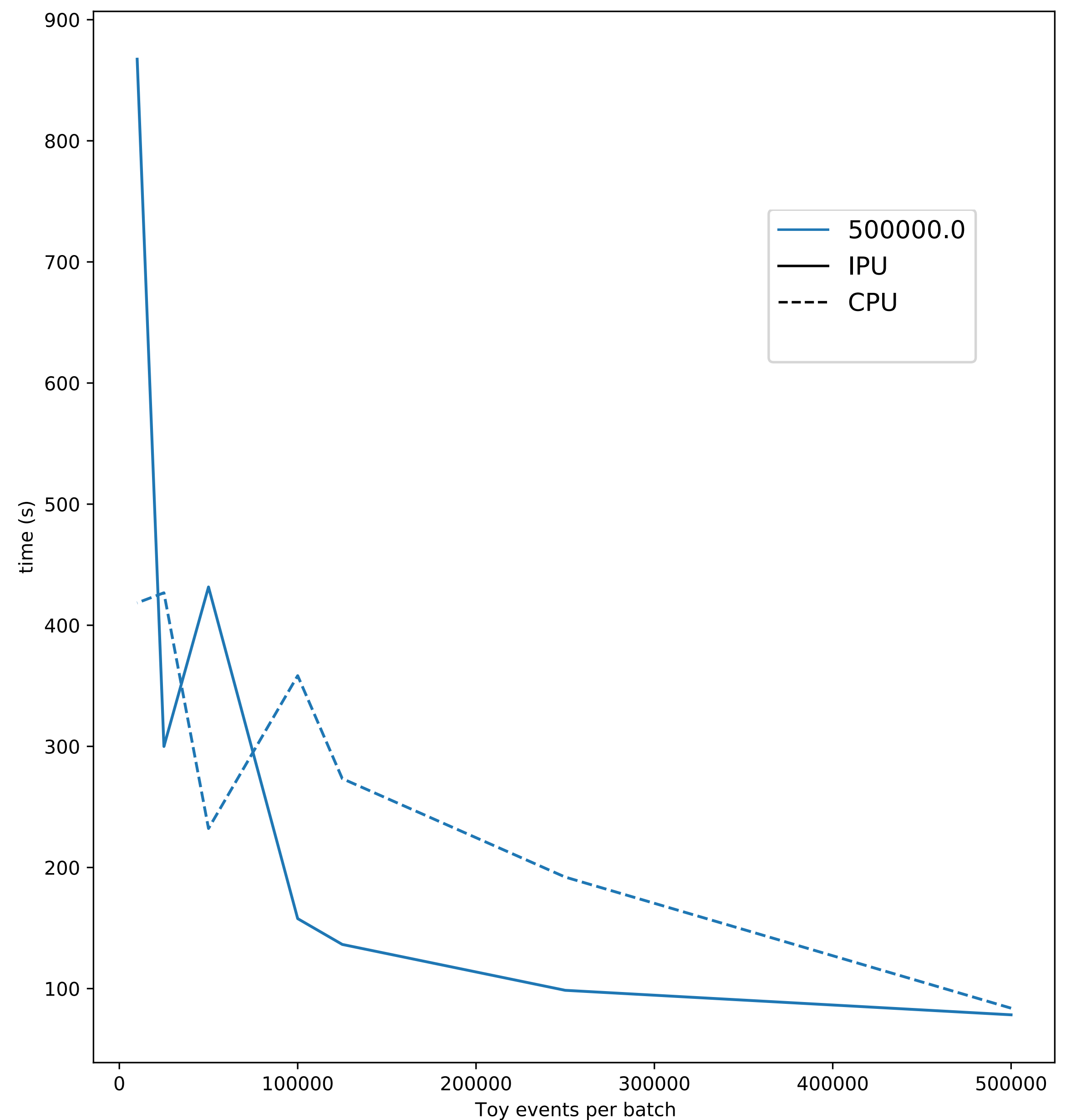
- Similar performance for the CNN but ~5x faster for the RNN on the IPU
- Perhaps not too surprising - even with fused operations, matrix sizes are smaller for the RNN

Benchmarks - maximum likelihood fitting

- Simple max likelihood fit of a double Crystal-Ball function
- Calculate likelihood on the IPU and do minimisation in Minuit on the CPU

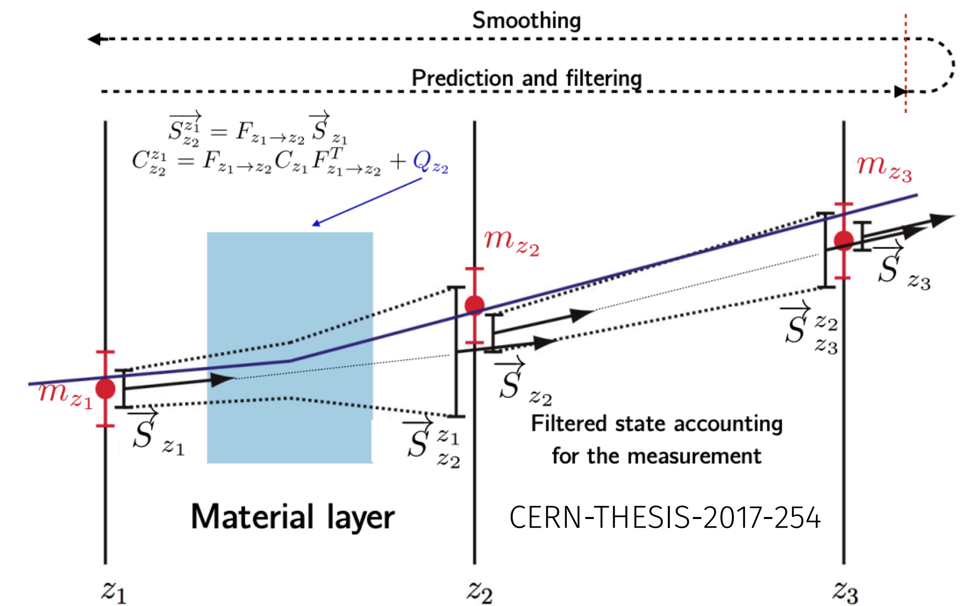
- May end up being too simple - not enough work done on the IPU such that the overhead from transferring log-likelihood to host RAM and updating parameters is small

- Will investigate more complex models, and hopefully optimise minimisation 'loop'

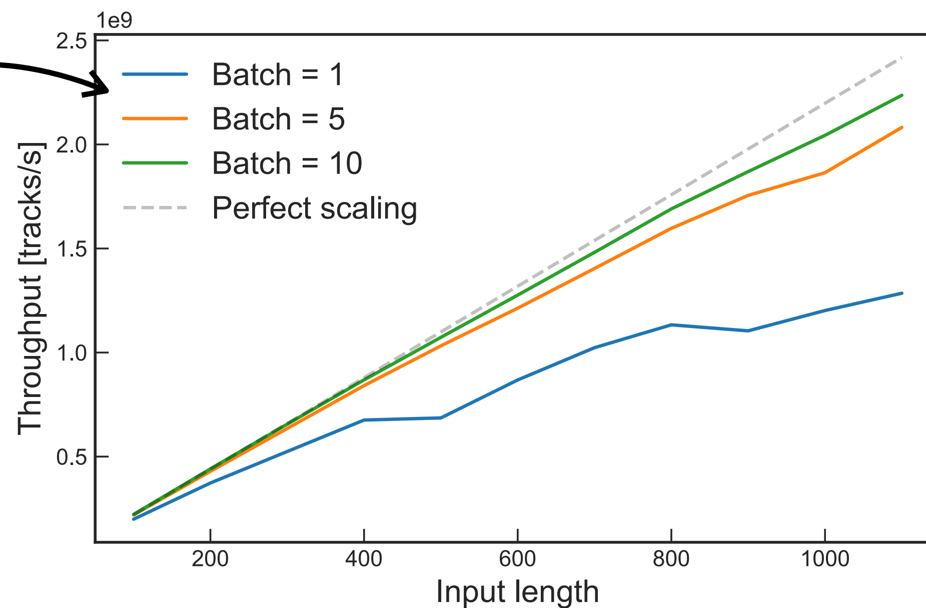


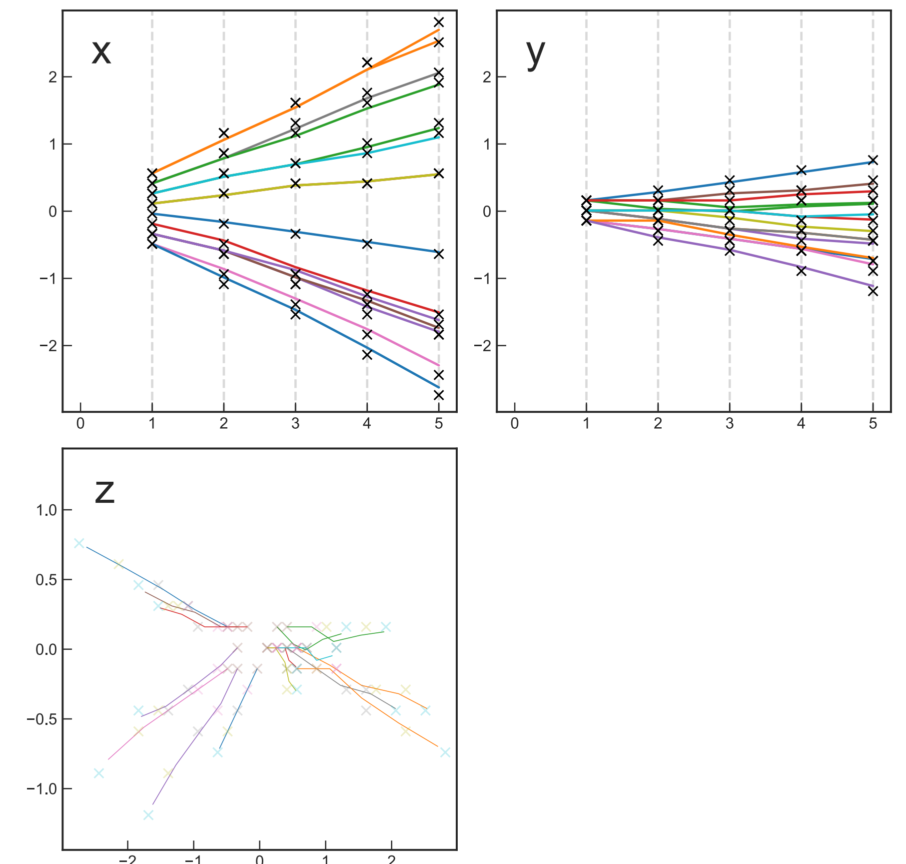
Benchmarks - Kalman filtering

- Simulate simple set of tracking stations, no magnetic field, parameterised homogeneous multiple scattering
- Kalman filtering: Progressive fit of track state to hits at each tracker plane, refined using detector uncertainty and kinematics projection
- Inherently parallel between tracks
- Assign each track to a tile, constrain operations to the respective tile - fit ~1200 tracks in parallel
- Poplar IPU implementation:



Reduce overhead by copying n batches each time





Benchmarks - Kalman filtering

- Would be nice to use this information for other things - rejecting ‘bad’ hits, exchanging hits between nearby tracks, etc

- Can exploit the IPU architecture for this, e.g:
  - Keep running track of residuals and calculate chi-squared
  - If a chi-squared is above threshold, re-fit without that hit
  - Synchronise
  - If the total track chi-squared is above threshold, exchange bad hits with other tracks
  - Re-fit
  - Synchronise again

- Might also be possible to use similar tile exchange for efficient primary vertex refinement

Funding

The Jean Golding* Institute

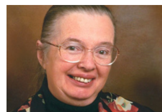
link/support/funding ←

The Alan Turing Institute

School of Physics

- Graphcore gives us cloud access to their IPUs
- We received 18k STFC IAA funding
- We are in the process of recruiting an RSE funded via the JGI (2 years) to work mainly on this project. Ad will be out, soon!
- Graphcore offered to gift us IPUs if we can find the money for the server… we are in the process of scraping that together.

*



Professor, Jean Golding, mathematician, epidemiologist and founder of the Children of the 90s cohort study.

Summary

- One IPU seems to usually exceed the performance to a last-gen datacentre GPU (Nvidia P100), where comparisons are available.
- At 150W per chip (2 per board), they are also likely a bit more efficient (price is TBD).
- *In principle* they are easier to program at a 'lower' level than GPUs.
- *In principle* they should have considerably better performance for tasks with more complex control flow or that require interprocess communication.
- We hope to demonstrate whether this is the case in future, with more studies on (e.g.) graph neural networks, more complex reconstruction and Kalman filter use cases, Bayesian MCMC fits, etc.
- The software seems to be improving very rapidly, so it should be increasingly easy to squeeze out all of the FLOPs

Some more architectural details

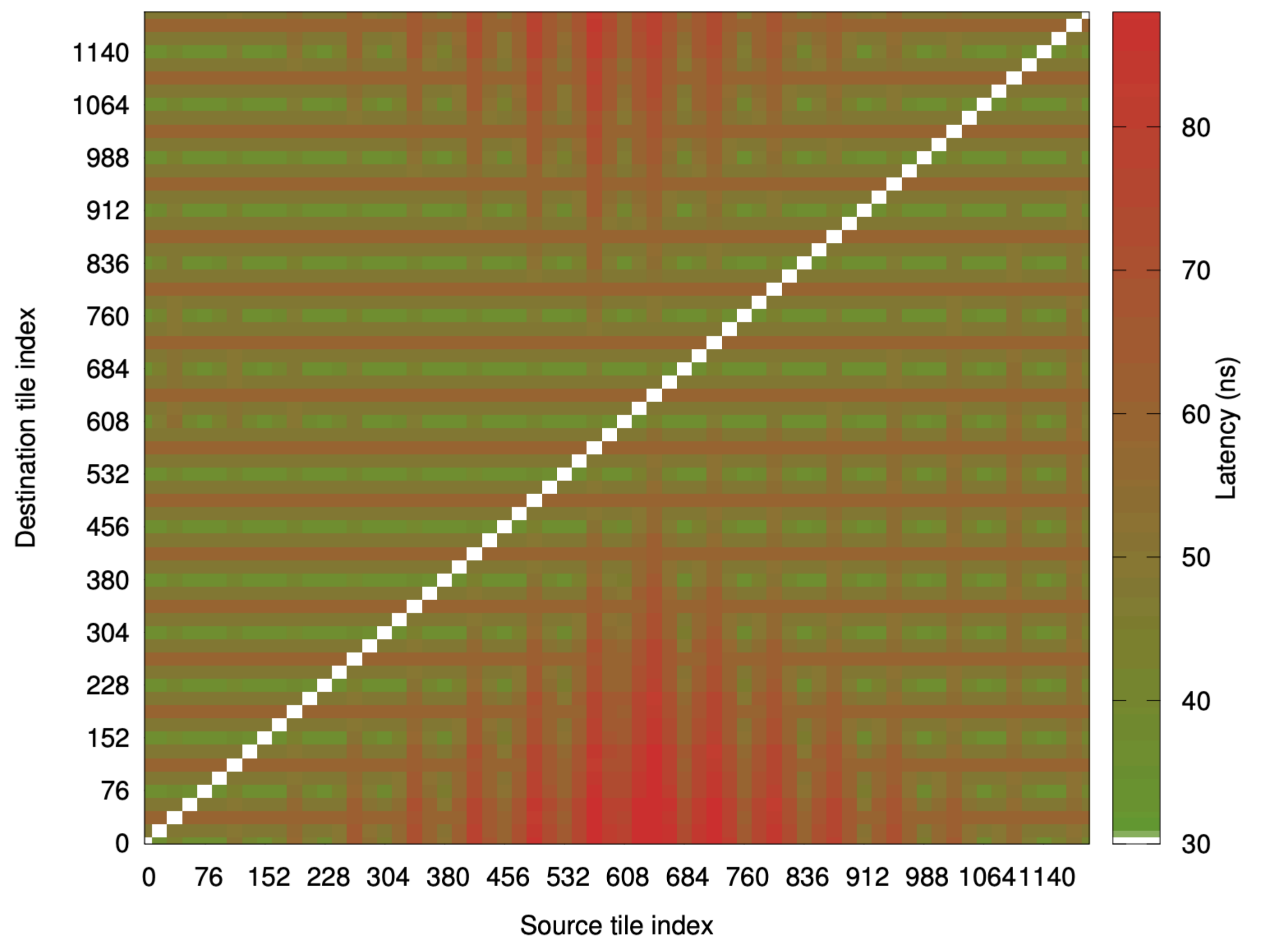
Report from a hedge fund with more

architecture information and benchmarks:

https://arxiv.org/abs/1912.03413

Architecture	Memory	Per-chip Capacity (MiB)	Latency (ns)	Latency (cycles)	Clock Frequency (GHz)
Graphcore IPU	Tile-local	304	3.75	6	1.60
NVidia T4 GPU	Shared L1	1.25 ... 2.5 1.25 ... 2.5	11.94 20.13	19 32	1.59
Intel *-Lake CPU	L1D L2	0.25 ... 0.875 4 ... 28	0.93 ... 1.92 2.79 ... 4.62	4 ... 5 12	2.60 ... 4.30

Table 1.1: Size and latency comparison between IPU memories and similar SRAM-based memory hierarchy levels on contemporary GPUs and CPUs. The IPU's local memory has lower latency than the fastest memories on the Turing GPUs and is on par with the L2 cache in modern Intel CPUs. However, the IPU's local memory is vastly larger than those memories in a per-chip comparison. (Intel data: from public sources; intervals range over the product offering at the time of the writing. GPU data: from our prior work [2].)



Tagging RNN partial dependence