Computing and Software for Big Science (2021) 5:8
https://doi.org/10.1007/s41781-021-00057-z

ORIGINAL ARTICLE

Studying the Potential of Graphcore® IPUs for Applications in Particle Physics

Samuel Maddrell-Mander[1,2] · Lakshan Ram Madhan Mohan[1] · Alexander Marshall[1] · Daniel O'Hanlon[1] · Konstantinos Petridis[1] · Jonas Rademacker[1] · Victoria Rege[2] · Alexander Titterton[2]

Received: 24 August 2020 / Accepted: 2 March 2021

Abstract
This paper presents the first study of Graphcore's Intelligence Processing Unit (IPU) in the context of particle physics applications. The IPU is a new type of processor optimised for machine learning. Comparisons are made for neural-network-based event simulation, multiple-scattering correction, and flavour tagging, implemented on IPUs, GPUs and CPUs, using a variety of neural network architectures and hyperparameters. Additionally, a Kálmán filter for track reconstruction is implemented on IPUs and GPUs. The results indicate that IPUs hold considerable promise in addressing the rapidly increasing compute needs in particle physics.

Keywords IPU · Hardware accelerators · Particle physics · Event generation · Tagging · Track reconstruction · Kálmán filter

Introduction

To perform high-precision measurements of rare processes, particle physics experiments require large data rates. At the Large Hadron Collider (LHC), for example, proton–proton bunch crossing rates of 40MHz result in a typical data rate of $\mathcal{O}(1)$ TB/s, which must be processed in near real time, and is expected to exceed $\mathcal{O}(10)$ TB/s at the high-luminosity LHC [1]. The future Deep Underground Neutrino Experiment is also expected to operate its data acquisition system with a throughput of $\mathcal{O}(1)$ TB/s [2]. Such applications currently require a large number of CPUs on site with considerable ($\mathcal{O}(1)$ PB) disk buffers. In cases where each of these events must be studied in some depth before deciding whether to save the event for offline processing, the overall signal rate is determined by the time taken to make this decision. Furthermore, these high-precision measurements require simulated data, produced 'offline', that mimic the real data as closely as possible, whilst also minimising the computational burden.

As a consequence of these constraints, many organisations within particle physics are investigating heterogeneous computing architectures as part of a strategy to cope with the vast data volumes expected in the next generation of experiments. Such architectures replace CPU-only configurations with combinations of CPUs and graphics processing units (GPUs), and sometimes additionally field-programmable gate arrays (FPGAs); see, for example, studies by ATLAS, COMET and LHCb [3–6]. Most notably, the first level of the software trigger of the upgraded LHCb experiment will run on GPUs [7], and is scheduled to begin operation in 2021.

✉ Daniel O'Hanlon
daniel.ohanlon@bristol.ac.uk

Samuel Maddrell-Mander
sam.maddrell-mander@bristol.ac.uk; samuelm@graphcore.ai

Lakshan Ram Madhan Mohan
lakshan.madhan@bristol.ac.uk

Alexander Marshall
alex.marshall@bristol.ac.uk

Konstantinos Petridis
konstantinos.petridis@bristol.ac.uk

Jonas Rademacker
jonas.rademacker@bristol.ac.uk

Victoria Rege
victoriar@graphcore.ai

Alexander Titterton
alexandert@graphcore.ai

[1] H H Wills Physics Laboratory, University of Bristol, Bristol, UK

[2] Graphcore, Bristol, UK

Published online: 17 March 2021

Increasingly, GPUs are also used for offline data analysis such as fitting complex theoretical distributions with many free parameters to large data samples, for example, using Nvidia's CUDA API [8], or with TensorFlow-based frameworks [9, 10]. As dataset sizes in particle physics are expected to increase exponentially in the coming years, while CPU clock speeds plateau, hardware accelerators are expected become increasingly important in online and offline computing.

Over time, graphics processing units have been modified for general purpose computing workloads, and have become the dominant form of single instruction, multiple data (SIMD), accelerator hardware available to consumers. However, with the renewed interest in large-scale machine-learning (ML) algorithms, numerous machine-learning specific hardware accelerators have been developed. Recently launched by Graphcore is the intelligence Processing Unit (IPU), a new type of hardware accelerator based on a bulk synchronous parallel multiple instruction, multiple data (MIMD) architecture, and designed for machine-learning applications.

This paper represents a first investigation of the suitability and performance of IPUs in typical high-energy physics ML applications, and an IPU implementation of a Kálmán filter. It includes benchmark tests relative to GPUs and CPUs. The hardware used for these studies is summarised in Table 1. The code used to produce the results presented here can be found in Ref. [11].

The paper is organised as follows: the next section provides a brief overview of relevant features of Graphcore's IPUs. The subsequent sections present implementations of several particle-physics-related applications, and their performance on IPUs, GPUs and CPUs. Then a study of generative-adversarial neural networks (GANs) for particle physics event generation and reconstruction is presented, and in the following section, neural network implementations for online flavour tagging. The code in these first sections is implemented in TensorFlow or PyTorch, and can easily be executed on IPUs, GPUs and CPUs. Additionally, the differences in performance behaviour between IPUs and GPUs are investigated in some detail for different network types and parameters. The penultimate section explores the IPU beyond neural networks and ML, and presents a Poplar-based implementation of a Kálmán filter, one of the most ubiquitous track reconstruction tools in particle physics. The final section concludes this paper.

Graphcore's IPU

The IPU is a new type of processor designed specifically for ML applications. Its architecture is fundamentally different from that of either CPU or GPU. A detailed review of the architecture and performance of the first-generation IPUs used in this paper can be found in Ref [12].

The IPU processor is optimised to perform highly parallelised fine-grained operations. In contrast to the SIMD architecture of GPUs, which requires contiguous vectorised data for efficient operation, the IPU is highly efficient on applications that require irregular and sparse data access and can run individual processing threads on small data blocks while exploiting its MIMD architecture.

This study makes use of Graphcore's first-generation Colossus™ MK1 GC2 IPU (see Fig. 1). This IPU comprises 1216 processing elements, called tiles, each of which consists of a computing core with 256 KiB of local memory. In total 7296 threads can be executed in parallel in a single IPU. The tiles are linked through an on-chip interconnect, the IPU exchange™, allowing for a low-latency and high-bandwidth communication up to 7.7 Tb/s. Each IPU card consists of two such IPUs. The IPUs are connected to each other via 80 IPU links™ reaching a total chip-to-chip bandwidth of 2.5 Tb/s, and are connected to the host via 16 PCIe Gen4 links (8 per IPU).

The IPUs used here are integrated into a DELL DSS8440 IPU server containing eight dual IPU cards. This server includes two Xeon Platinum 8168 CPUs with 24× 32 GB 2.4 GHz DDR4 DIMM Modules. Graphcore also provides drivers along with its Poplar Software Development Kit (SDK). Updates to both the drivers and SDK can

Table 1 Key specifications of the processors used in this paper as provided on manufacturer websites [13–16], and in [12, 17]. Many features are not represented in this table; key differences in performance arise from the very different memory architectures and technologies. Performance in terms of floating point operations per second (FLOPS) is given for 32 bit single-precision operations. Thermal design power (TDP) is given for each processor, where for the IPU this is half of the total board TDP.

	Name	Cores	Memory	Clock speed	TDP
CPU 1	Intel Xeon Platinum 8168	24	732 GiB	2.7 – 3.7 GHz	205 W
CPU 2	Intel Xeon E5-2680 v4	14	128 GiB	2.4 – 3.3 GHz	120 W
	Name	Cores	Memory	32 bit FLOPS	TDP
GPU	Nvidia TESLA P100	3584	16,000 MiB	9.3 TFLOPS	250 W
IPU	Graphcore Colossus™ GC2	1216	286 MiB	31.1 TFLOPS	*120 W

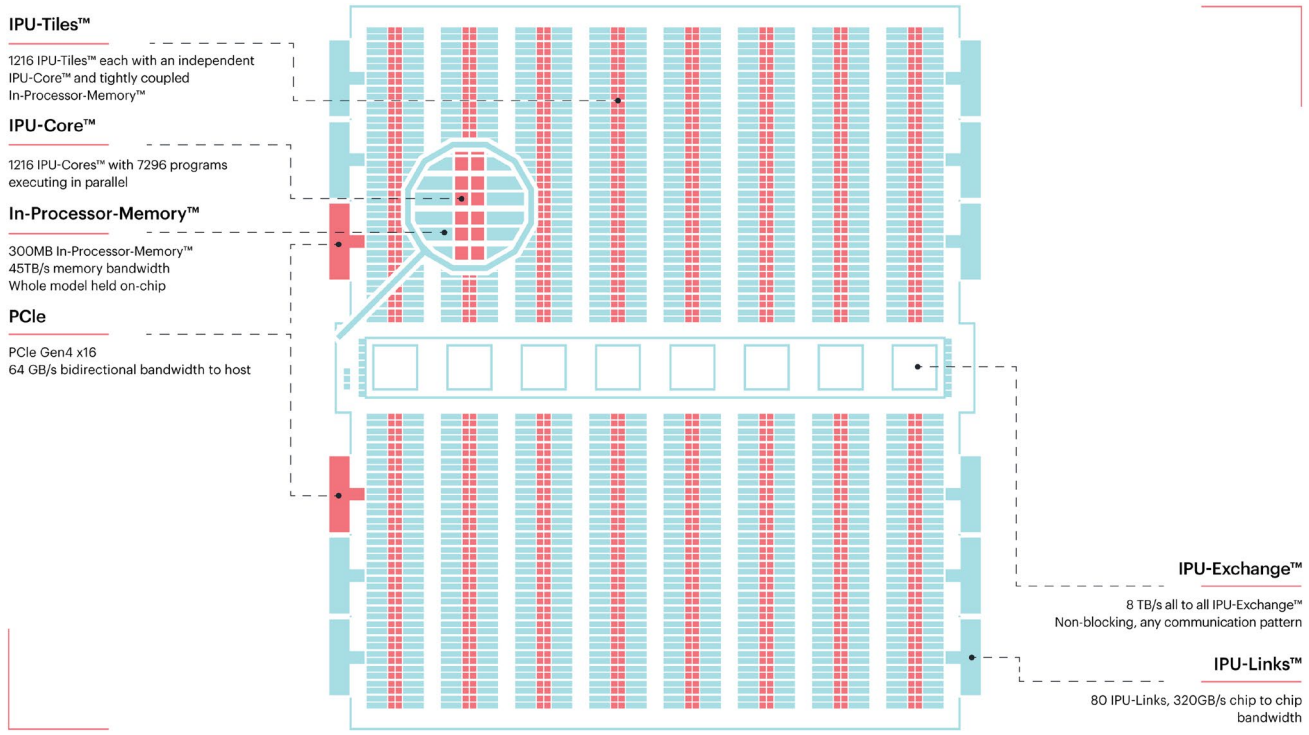

Fig. 1 The Graphcore Colossus™ MK1 GC2 IPU [13]

result in improvements to the IPU performance. This paper relies on SDK version v1.2.0.

During the preparation of this paper, Graphcore released its second-generation IPU, the Colossus™ MK2 C200 with 20% more tiles and triple the local memory per tile [13].

In this paper, the performance of a single first-generation IPU is tested against a Nvidia TESLA P100 GPU and two types of CPUs, depending on the particular form of the test. The power consumption of the single IPU is approximately half that of the GPU. Key technical specifications of the IPUs, GPUs and CPUs used are given in Table 1.

IPUs out-perform GPUs in many machine-learning applications such as computer vision, natural language processing and probabilistic modelling [18–20]. Machine learning has been used in particle physics for decades, initially referred to as 'multivariate analysis' and typically carried out with tools developed by and for particle physicists, such as the widely used TMVA package [21]. Increasingly, though, industry-standard tools and environments are being used, such as CUDA [22] TensorFlow [23] and PyTorch [24]. While ML algorithms are most frequently applied in the final stage of event selection, they are also used for particle identification [25], flavour tagging [26] and triggering [27, 28]. Neural networks have been studied for use in track reconstruction [29], motivated by their high performance on hardware accelerators like GPUs and FPGAs.

The increased use of GPUs in particle physics offline data analysis coincided with the advent of increasingly user-friendly programming environments (such as CUDA and TensorFlow) that allow programmers without special training to easily exploit GPU resources. Such environments exist for IPUs already, including TensorFlow, PyTorch, and Graphcore's C++-based API, Poplar. Ease of programming is a substantial advantage over FPGAs, and is, apart from performance, a key reason that motivates our study of potential use of IPUs in particle physics.

In the same way, as GPUs outperform CPUs not only in the rendering applications they were originally designed for, but also other applications such as ML, it is reasonable to expect IPUs to excel in applications beyond ML. Particularly promising are applications that benefit from the IPU's flexible MIMD architecture that contrasts with the GPUs SIMD design, which may result in more optimal parallel software.

Event Generation and Tracking Corrections Using GANs

Generative Adversarial Networks (GANs) are a class of flexible neural network architectures characterised by a two-player adversarial training environment where the response of a classification discriminator network informs the updates to a generator network [30]. The discriminator is trained to

distinguish between generated samples and samples from a training set. The generator network transforms a vector of random noise into a fabricated sample. GANs are trained with an iterative approach, this allows the generator and discriminator networks to improve together in parallel. The goal of GAN training is to create a generator that is able to emulate the characteristics of a training data set with high fidelity.

In the ML community, GANs have been shown to work well across a spectrum of tasks. The most common task is the generation of data in the form of images [31–34]. Increased functionality in the GAN comes with the introduction of conditional inputs into the generator, where the conditional arguments represent characteristics of the generated sample. The conditional input could be an input image to which a style transfer can be applied [35], or the resolution upscaled to reconstruct sub-pixel information [36]. The flexibility of neural networks enable the creation of a wide range of architectures. These recent developments in the ML community, catalysed by hardware improvements, have improved generative neural networks to the point that they can feature as viable tools within particle physics computation. GANs are capable of modelling high-dimensional distributions or transformations and are able to generate samples with high fidelity to training information. Conditional architectures can be designed to enable the networks to understand physical processes.

Applications of GANs within particle physics are constantly appearing. GANs have been applied in both event generation [37–43] and detector modelling [44–52]. In this section the inference and training speeds of some of these particle physics based GANs are assessed on the IPU hardware and compared to results on the GPU and CPU described in Table 1.

Event Generation

Accurate event generation is a crucial component of modern particle physics experiments. Large samples of simulated particle physics processes, including the detector response, are required to optimise the design of the detectors, develop reconstruction algorithms, understand the efficiency subsystems and model the impacts of various physics based selection criteria. Experiments at the LHC simulate billions of events every year, each event taking $\mathcal{O}$(minutes) to simulate [37]. This results in simulation campaigns consuming up to 70% of experiment computing resources [44, 53].

Newly proposed experiments will continue to demand a rapid increase in the number simulated events [54, 55]. The ongoing optimisation and parallelisation of traditional event generation software will at best result in an order of magnitude reduction of resources [56, 57]. This reduction is not sufficient to meet ever increasing simulation demand. Estimates forecast a fourfold shortfall of computing power within the next 10 years without significant new investment [58, 59]. This has catalysed efforts to develop faster simulation and event generation technologies of which GANs are currently a front runner. GANs or other generative network architectures are likely to become an integral part of a future fast simulation tool kit.

GANs are, of course, unable to completely replace traditional simulation methods as they rely on training data produced with the slower full physics simulation, this fact makes the optimisation of traditional methods no less valuable. GANs learn by example and are largely limited to modelling the exact process that they were trained on. In comparing a GAN to the full simulation care needs to be taken to assign a systematic uncertainty related to the residual mismodelling. The GAN event generation is particularly helpful when the systematic uncertainty due to its mismodelling is smaller than other errors associated with other parts of the analysis procedure [38]. A limitation of the GAN-based event-generation stems from the fact that the range of the feature space that the GAN can accurately model is defined by that of the full-simulation training sample. However, GANs are able to accurately interpolate between points in the feature space of the training sample, acting as a powerful data augmentation tool.

Using GPUs to generate events using a GAN-based approach offers large increases in event-generation rate over traditional simulation approaches [37, 38, 47]. However further increases in the rate would be valuable. This section investigates if IPUs can provide any additional increase in the inference speed of a GAN for event generation.

Examples of GAN architectures are taken from the literature and event-generation rates are compared across a range of batch sizes and different hardware options. Currently, convolutional networks are the most commonly used in the particle physics community. Two such networks are investigated here, the small convolutional DijetGAN from Ref. [39] and the larger locally connected LAGAN from Ref. [37]. Additionally, two fully connected networks are investigated. These are the prompt and non-prompt muon kinematic generators developed for the SHiP experiment in Ref. [38]. Both fully connected networks are of similar architecture; however, the prompt network is significantly smaller. As the network weights are not publicly available for all the network architectures under study, random values are assigned to the network weights without affecting the speed of the event generation.

Figure 2 presents the event-generation rate for CPU, GPU and IPU as a function of the batch size for each network studied. The relationship between rate and batch size is shown to be consistent across network and hardware configurations, with larger batch sizes giving larger generation rates. However, there is a limit to the maximum batch size

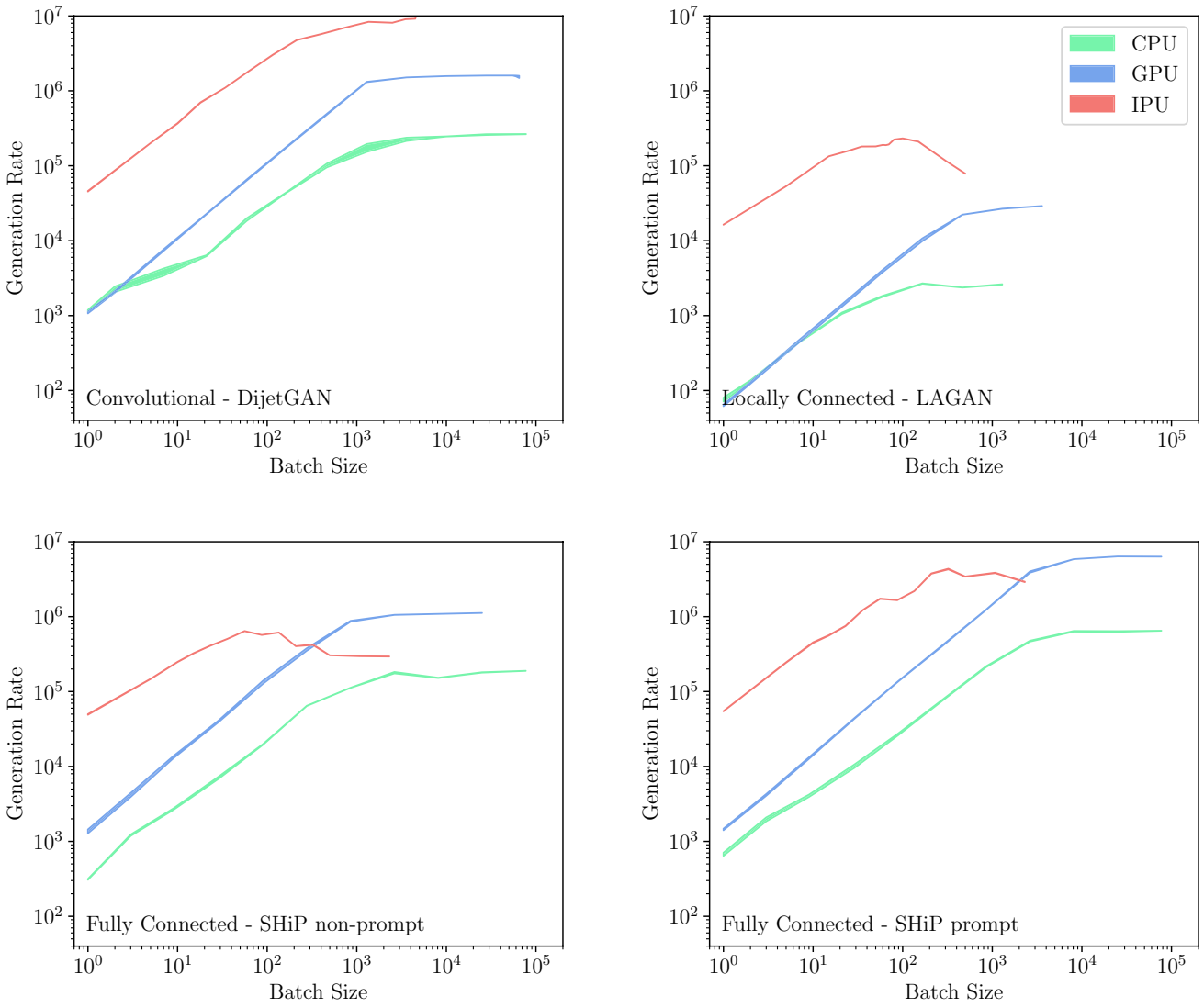

Fig. 2 Benchmarking results of the event-generation rate as a function of the batch size of the network. Results are presented for IPU, GPU and CPU hardware options outlined in Table 1

accessible by each hardware option due to memory constraints. This limitation results in a plateau of the event-generation rate.

For each network architecture and hardware option, the batch size that gives the largest event-generation rate is chosen. The CPU and GPU results are obtained with TensorFlow `2.1.0` and the IPU results are obtained using TensorFlow `1.15.0` as Graphcore's SDK version 1.2.0 offered a more comprehensive support for TensorFlow `1.x`. For each benchmark run, warm up batches are passed before anything is timed. The TensorFlow profiler was used to ensure the GPU inference was indeed dominated by computation time and not an unforeseen bottleneck (Table 2).

Across all networks tested the IPU is faster than the GPU at generating events using small batch sizes. For the fully connected networks, both of which have two hidden layers, the GPU becomes more efficient at higher batch sizes which are not accessible by the IPU that was used due to memory constraints. As the batch size approaches the limit

Table 2 Benchmarking results calculated using optimal batch size for each hardware option.

Network name	Number of parameters	IPU/CPU rate	IPU/GPU rate
DijetGAN	3×10^4	36.3	6.0
LAGAN	4×10^6	86.5	8.0
SHiP non-prompt	5×10^6	3.4	0.6
SHiP prompt	6×10^5	6.7	0.7

for a single IPU, the performance appears to degrade. This is most likely due to overheads in the computation associated with organising large tensors in memory. At the most efficient point, the fully connected networks were 1.4 and 1.7 times faster using the GPU for the smaller and larger networks, respectively.

In contrast, the IPU outperforms the GPU for both of the convolutional networks tested. For optimal batch sizes, the IPU presents an increase in event-generation rate compared to the GPU by a factor of 6.0 and 8.0 for the small and large networks, respectively.

Training Models

The results of "Event Generation" show that IPUs outperform GPUs for networks with a small batch size. Trained GANs used for event generation are implemented using the optimal batch size, which generally corresponds to the largest batch size accessible to the hardware. However, a small batch sizes contain a stochastic component originating from the random selection of training samples. This stochastic effect can help to move network configurations out of local minima. Larger batch sizes have advantages too, more efficient computation per training sample and a more accurate assessment of the gradient at each step. So called mini-batch gradient descent aims to operate with a batch size that balances this stochastic effect with the accuracy of gradient updates computed with large batch sizes. Appropriate choice of the batch size during training of the network can provide a faster overall convergence to an optimal configuration. Commonly the batch size chosen for training a GAN is $\mathcal{O}(50)$.

This section investigates the performance of the IPU for training the GANs described in "Event Generation". The smaller models of the dijetGAN and SHiP prompt GAN, are trained on a single IPU. The larger models cannot currently be trained on the IPU as the generator and discriminator networks must fit onto a single IPU. Graphcore do offer *sharding*, which allows networks to be split across multiple IPUs. Whilst the sharding approach works well for a single network, it is not yet possible for a GAN model. The GAN case is complicated by the continual interactions between models. This may be possible in the future.

The training time is defined as the time taken to run over 1000 batches using the batch sizes reported in their respective publications. As for the inference benchmarks, a warm-up phase containing all compilation overheads is discarded from the test. The batch sizes are 50 for the SHiP prompt GAN and 128 for the dijetGAN. The IPU training times are then compared to the same test completed on the GPU and CPU from Table 1. The results are presented in Fig. 3. Both networks train significantly faster on the IPU as expected from the inference performance discussed in

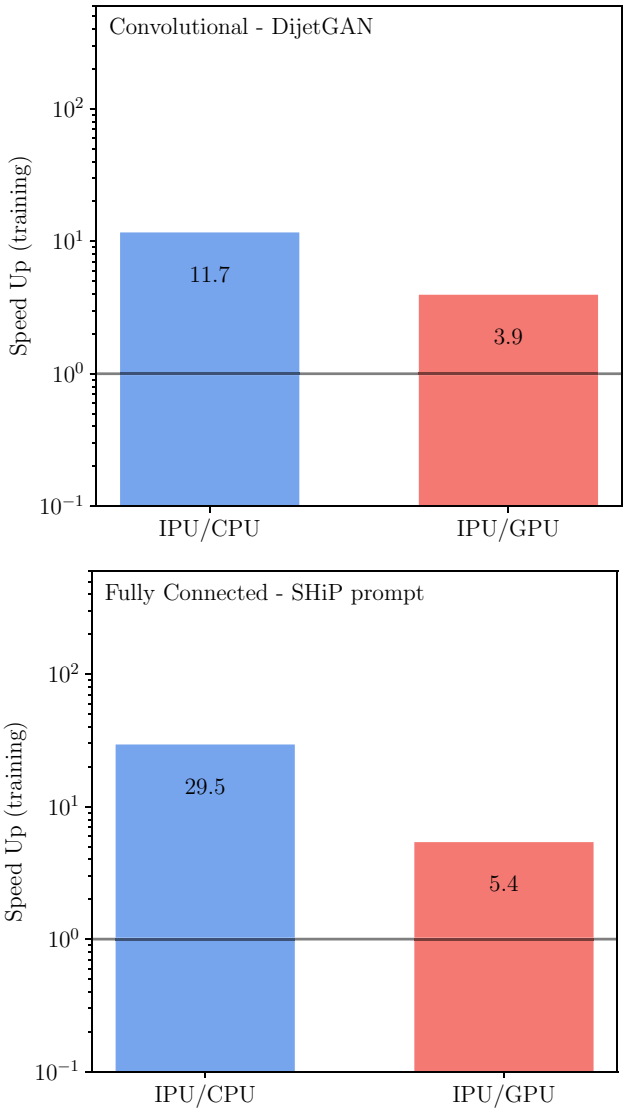

Fig. 3 Comparison of the time to train the IPU relative to the CPU or GPU of Table 1

"Event Generation", where for lower batch sizes, the IPU consistently outperforms the GPU.

Track Corrections

As observed in previous sections the IPU significantly outperforms the GPU at lower batch sizes. This section presents an example algorithm that would typically be executed with a batch size of 1. The algorithm presented has not yet been employed in a working particle physics environment but is used here as an example of where the IPU might thrive.

The use of GANs extends beyond event generation and can be employed in data processing. Charged particles traversing a medium are deflected through multiple small-angle scatters due to the Coulomb force acting between

the charged particle and the nucleus of the material. The resulting trajectory of the particle is therefore modified by this scattering and traditional tracking methods rely on techniques such as the Kálmán filter, discussed in "Kálmán Filter Implementations Across SeveralArchitectures", to account for this effect. Such methods can be computationally expensive. Therefore, employing a fast pre-processing stage prior to the track-fit that corrects for the effects of multiple scattering could be desirable.

Previous work on GANs has shown that in addition to conditional class information, a generator can be conditioned with an input state to be manipulated. This is typically an input image to which a style transfer can be applied [35], or the resolution upscaled to recover sub-pixel information [36]. This family of transformations is of particular interest in particle physics and other scientific domains, as it shows that using a GAN high-fidelity information can be correctly recovered. In the context of particle physics, this could mean correcting for the resolution of the detector, accounting for detector misalignment or upscaling the reconstructed hit information of charged particles to correct for effects such as multiple scattering prior to a track fitting algorithm.

To provide a simple concrete example, the algorithm presented in this paper aims to correct for the effect of multiple scattering from the trajectory of a charged particle in two dimensions. A simplified simulation is developed to model the multiple scattering of a charged particle traversing a series of active detection material made of silicon. The multiple scattering of the charged particle with each layer of silicon is modelled according to Ref. [60], where the particle's path is deflected according to a Gaussian distribution whose width depends on the original particle's momentum and velocity as well as the thickness of the scattering medium. The same initial conditions are used to generate a second, 'true', charged particle that does not undergo scattering. The GAN is trained to perform a style transform from the scattered track to true track.

The generator model used for this study is based closely on the `pix2pix` algorithm [35] as it has been shown to generalise over different applications without major changes to the network architecture. The generator model consists of a U-Net encoder–decoder structure [61] with "skip" layers between each of the layers. The skip connections allow to scale specific information to directly pass across the generator and bypass the bottle neck. The key difference to GANs used for image generation is an additional super resolution layer to upscale the output. The variation of this model used to model charged tracks is referred to as qSRGAN.

An example of how this algorithm performs on a pair of tracks is shown in Fig. 4.

In contrast to event generation methods described in "Event Generation" where the maximal throughput is obtained using larger batches, track corrections would typically be done on an event-by-event basis. This allows the performance of the IPU at low batch size to be utilised efficiently. The performance of the qSRGAN algorithm for inference is tested on the CPU, the GPU and the IPU given in Table 1. Two key results are presented. Firstly the throughput of the algorithm as a function of batch size, and secondly the ratio of the rates of the CPU and GPU to the IPU for a batch size of one image. The results are shown in Fig. 5 where the rate of the image generation using an IPU is larger by a factor of 22 relative to a CPU, and 4.5 relative to the GPU. The increased generation rate of the IPU compared to the GPU would allow either a higher total throughput to better cope with higher event rates, or a significantly more complex model for the same total compute budget.

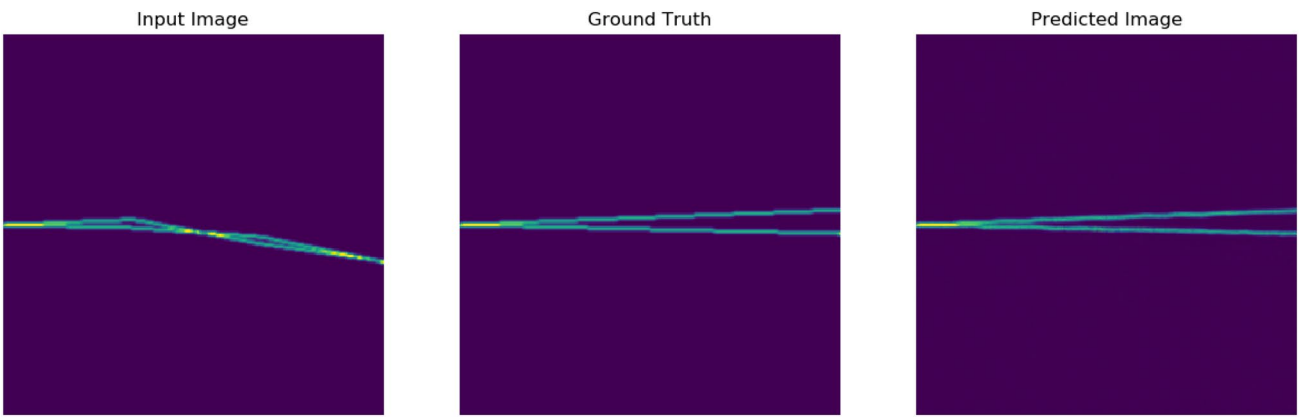

Fig. 4 An example of correcting for the track multiple scattering using the qSRGAN. The left image is the input to the Generator, the middle image is the true image with no scattering, and the right image is the generated output

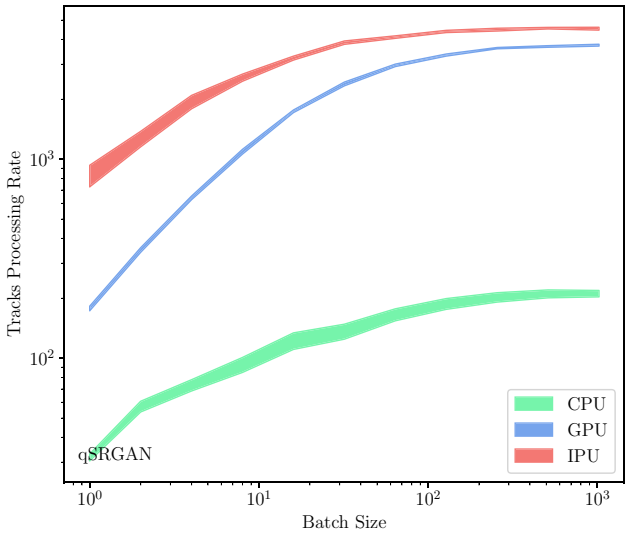

Fig. 5 Benchmarking the qSRGAN algorithm on CPU, GPU, and IPU processors. The inference throughput for each processor is shown as a function of batch size

Determining the Flavour of *B* Mesons

Neural networks are commonly used to combine lower-level detector-specific information to determine the identity or quark composition of a particle. Given the large number of particles produced in each collision event, inference speed is an important consideration, regardless of whether these are evaluated 'online' as part of the reconstruction and trigger framework, or 'offline' after the initial rate reduction from the trigger.

For some applications, such as the determination of the flavour of the $B^0_{(s)}$ meson at production time, significantly increased classification accuracy is achieved by applying a network over all particles in the event, rather than selecting particles thought to be of particular interest ahead of time. In this way, correlations between the features of different particle tracks can also inform the resulting flavour determination [62]. Two canonical neural network components that enable this multidimensional data to be taken into account are convolutional and recurrent neural networks. In general, gated recurrent networks are able to better exploit long-distance dependencies between the input sequence, whereas convolutional networks tend to be faster to train and execute. However, the trade-offs between each in terms of the classification accuracy and execution speed are beyond the scope of this paper, which rather focuses on the performance of each network on different hardware.

In each case, the convolutional or recurrent layers operate over an input of shape $[n_{\text{batch}}, n_{\text{tracks}}, n_{\text{features}}]$, where n_{batch} is the number of examples per training or inference batch, n_{tracks} is the number of input tracks, each with n_{features} features. Here, the recurrent network implementation uses a 'long short-term memory unit' (LSTM) [63] followed by a number of fully connected layers operating on the output of the last element in the sequence. For the convolutional network, several one-dimensional convolution operations with learnable kernel parameters, are applied sequentially. These convolutional layers are followed by a downsampling 'max-pooling' operation that propagates only the maximum of its inputs over a fixed range, and subsequently flattened to one dimension before entering a set of fully connected layers. The corresponding network configuration, and example parameters, can be seen in Table 3.

Both of these networks are constructed in PyTorch `1.2.0` [64], and exported to the ONNX [65] interchange format. For execution on the IPU, the ONNX models are imported into the Graphcore PopART framework. For the CPU and GPU benchmarks however, the networks are executed directly in PyTorch, which for GPU execution ensures that the optimised Nvidia CuDNN LSTM [66] implementation is used. The CPU is one single core of an Intel Xeon Platinum 8168 processor, the GPU is an Nvidia P100 (using CUDA toolkit `10.0` and CuDNN `10.1`), and the IPU is a Graphcore C2 IPU (using Poplar `1.3.0`). In general on the IPU, performance using ONNX and PopART is equivalent to using TensorFlow.

The networks are configured with hyperparameters that result in a modest total number of trainable parameters, whilst still permitting execution in reasonable time for particle physics applications. A critical parameter that affects inference time, particularly for SIMD processors such as GPUs, is the batch size (i.e., the number of inputs present on the device and executed over in a single inference step). The variation of inference time per event as a function of the total number of events per batch, can be seen in Fig. 6. Here, events of size of $n_{\text{tracks}} = 100$ and $n_{\text{features}} = 18$ are used (in addition to the parameters given in Table 3), which are typical for tagging at LHCb.

In each case, the IPU dominates the execution performance of the GPU and CPU at low batch sizes, and therefore has a lower single event latency (i.e., at batch size 1), which could be useful for some applications. Nevertheless, the GPU saturates to a higher overall throughput at higher batch sizes.

The batch size is expected to be the dominant factor controlling performance for SIMD processors, all else being equal. However, it is instructive to explore how the variation of network parameters affects relative GPU and IPU performance, particularly given that the IPU does not primarily gain its performance from SIMD processing, so whilst being used for similar purposes, GPUs and IPUs are architecturally quite different. For the recurrent network architecture, scans are performed over the batch size, number of hidden units (common to each layer), the number of input features

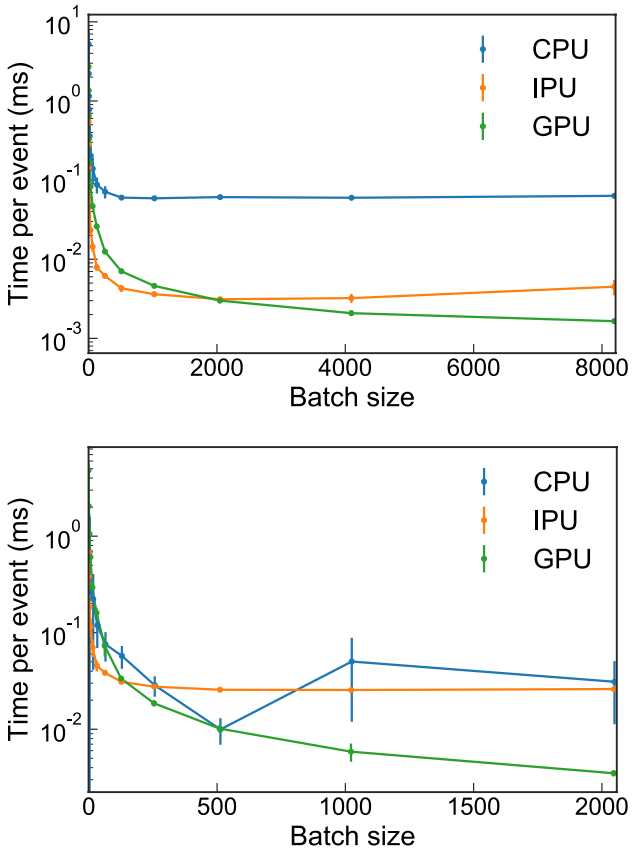

Fig. 6 Recurrent (top) and convolutional (bottom) neural network execution time per event as a function of the batch size

Table 3 Convolutional and recurrent neural networks used in the flavour tagging example

Convolutional network	Recurrent network
Conv1D(hidden = 8, k = 20)	LSTM(hidden = 8)
Conv1D(hidden = 8, k = 10)	Linear(hidden = 8)
MaxPool1D(pool = 2)	
Flatten()	
Linear(hidden = 8)	
Linear(hidden = 8)	

Parameters correspond specifically to plots in Fig. 6, and inputs are processed sequentially from the upper to the lower layers, with an implicit sigmoid activation at the end to express the probability of being a B^0 or $\overline{B}^0$

per track, and the number of input tracks. Projections of the ratio of the time per input for the GPU and IPU versus each of these parameters can be seen in Figs. 7 and 8.

In each plot, the black curve is the average across all other parameters, holding the x-axis parameter constant, and the coloured band spans the minimum and maximum variation of the ratio of execution times. Therefore, it is expected that if the dependence on relative performance is due to a single of these parameters, then the extent of the coloured band in the plot of this parameter will be small, indicating no or little variation due to the other parameters; at the same time, the black curves in the plots of the other parameters will have little variation as a function of that parameter.

For the RNN in these configurations, we observe a weak dependence on the input length and hidden size, however moderate dependence is seen on the batch size and the number of input features. That no parameter is sufficient to entirely describe the behaviour indicates that the relative performance of the GPU and IPU is a complicated function of all neural network parameters. However, it is clear from these plots that the IPU is better performing for smaller batch sizes, and a smaller number of input features, compared to the GPU.

For the CNN, a more mixed picture is observed, where no single parameter significantly represents the difference between the IPU and GPU performance; however, the largest dependence is on the batch size and number of input features. In this case, it is clear that the kernel size has a significant impact on the difference in execution time between the IPU and GPU, where the IPU tends to perform better in some cases with large values, and in some cases with small values.

Kálmán Filter Implementations Across Several Architectures

Kálmán filters are a ubiquitous technique for state-space estimation from multiple noisy measurements, and are used in fields as diverse as robotics, signal processing, and econometrics. In particle physics they are most commonly used as a method to incorporate kinematical constraints and detector-material interactions when estimating the particle track state from clustered hits in tracking stations. As such, Kálmán filters often form the basis of event reconstruction algorithms.

Recent emphasis on complete online processing of full events motivates the need for more efficient reconstruction algorithms. In particular, from Run 3 of the LHC, the LHCb experiment intends to perform full event reconstruction at 30MHz in the high-level trigger, to exploit the efficiency gain from performing analysis-level selections earlier in the pipeline. As such, the execution speed of this reconstruction, of which the Kálmán filter is a dominant contributor [67], is strictly limited from a cost-performance perspective.

As many of these operations are inherently parallelisable, implementation of the reconstruction and track filtering on graphics processing units (GPUs) shows good promise, and is potentially a more cost effective alternative to CPUs. Nevertheless, as GPUs are generally designed as single-instruction multiple-data processors, they lack many features that

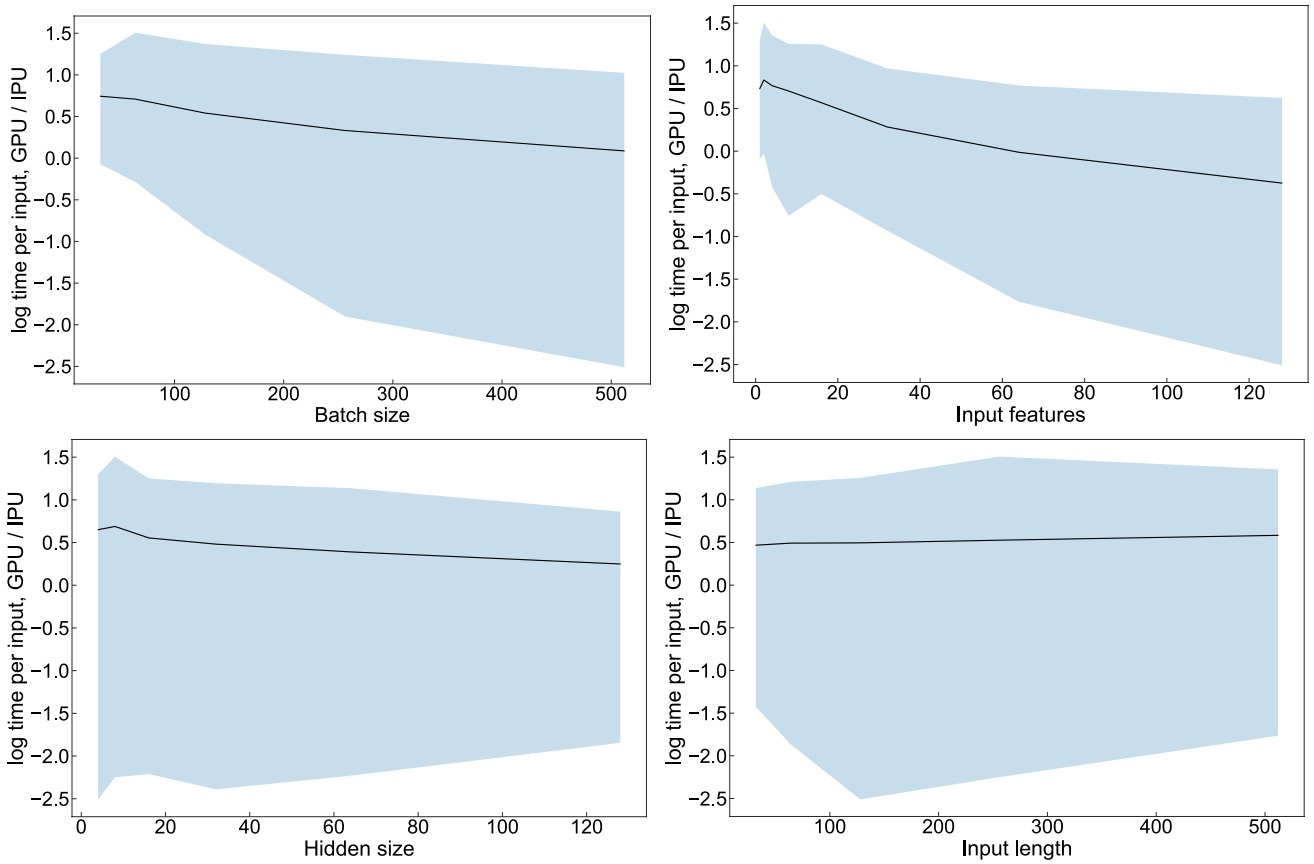

Fig. 7 Variation of the logarithm of the ratio between the time taken for each input event as a function of batch size, number of input features, hidden layer size, and input length, for the recurrent neural network. In each case, the black curve indicates the average time ratio when holding the x-axis value constant, and the coloured band spans the spans the range of possible ratios with constant x-axis value. A value of 0 indicates identical execution time for the GPU and IPU

are found in CPUs, such as support for conditional program flow, large caches, and fast interconnects between the compute cores.

Kálmán Filter Formalism

Kálmán filters recursively compute closed-form least-squares estimates for the state and its covariance matrix, under the assumption that all uncertainties can be well described by multidimensional normal distributions; and that only linear relations exist between the state at step t and the state at step $t+1$, and the state and the measurement process. The application of a Kálmán filter can be broken down into three stages: a prediction (or projection) stage where the state at step t is projected linearly to a state at step $t+1$; a filtering stage where the state at step $t+1$ is corrected using the measurement and covariance matrix of the measurement at step $t+1$; and a smoothing stage after all filtering steps, where state and covariance matrix updates are propagated backwards through the states to achieve a globally optimal configuration. The formulation here follows that of Refs. [68, 69] (Fig. 9).

The first projection step is described by a set of recurrence relations that extrapolate the state described by a vector $\mathbf{p}$ at step t to the values at step $t+1$, given by

$$\mathbf{p}_{t+1,\text{proj}} = \mathbf{F}_t \mathbf{p}_t, \tag{1}$$

with the covariance matrix of $\mathbf{p}$ given by $\mathbf{C}$, where

$$\mathbf{C}_{t+1,\text{proj}} = \mathbf{F}_t \mathbf{C}_t \mathbf{F}_t^{\top} + \mathbf{Q}_t. \tag{2}$$

These relations are expressed in terms of the transfer matrix $\mathbf{F}_t$, and the random error matrix $\mathbf{Q}_t$. The expression in Eq. 1 uses the underlying modelling assumptions (in the case of this particular track reconstruction, simple kinematics) that generate p_{t+1} from p_t via the application of the linear operator $\mathbf{F}_t$. The error matrix $\mathbf{Q}$ contains the process noise that involves terms that describe additive errors to the estimated state, such as those that are picked up after each propagation step from material interactions.

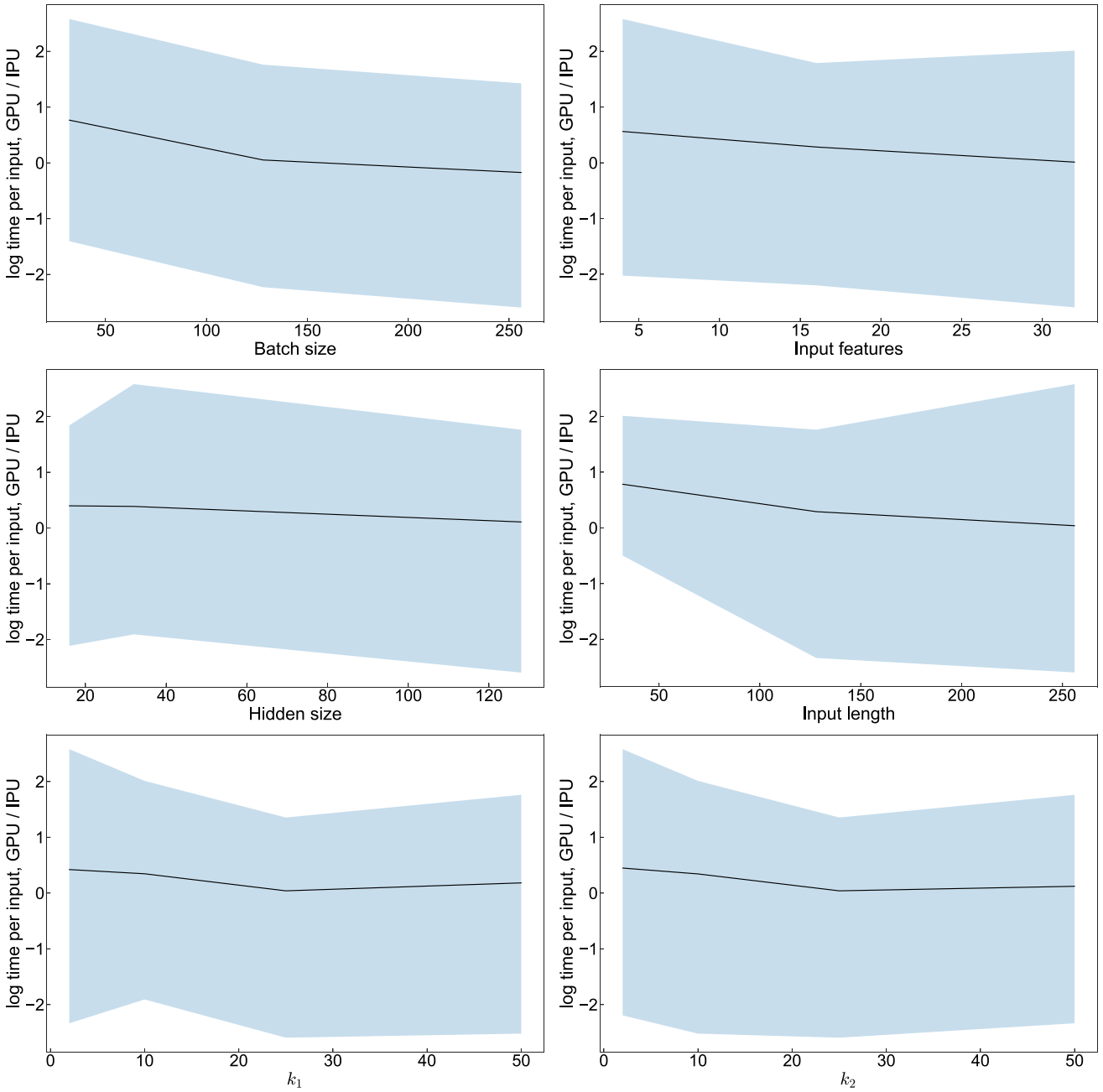

Fig. 8 Variation of the logarithm of the ratio between the time taken for each input event as a function of batch size, number of input features, hidden layer size, input length, and the size of the two convolutional kernels, for the convolutional neural network. In each case, the black curve indicates the average time ratio when holding the x-axis value constant, and the coloured band spans the range of possible ratios with constant x-axis value. A value of 0 indicates identical execution time for the GPU and IPU

At step $t+1$, the prediction from step t to $t+1$, $\mathbf{p}_{t+1,\text{proj}}$ is updated using the measurements at $t+1$, $\mathbf{m}_{t+1}$. The relation between the measurement $\mathbf{m}$ and the state $\mathbf{p}$ is given by $\mathbf{H}$ (which in general is independent of t), and the updated *filtered* expectation of $\mathbf{p}_{t+1}$ becomes

$$\mathbf{p}_{t+1,\text{filt}} = \mathbf{C}_{t+1,\text{filt}}\left[\mathbf{C}^{-1}_{t+1,\text{proj}}\mathbf{p}_{t+1,\text{proj}} + \mathbf{H}^{\top}\mathbf{G}_{t+1}\mathbf{m}_{t+1}\right], \tag{3}$$

where

$$\mathbf{C}_{t+1,\text{filt}} = \left[\mathbf{C}_{t+1,\text{proj}} + \mathbf{H}^{\top}\mathbf{G}_{t+1}\mathbf{H}\right] \tag{4}$$

is the corresponding covariance matrix. Here, $\mathbf{G}_t$ is the matrix that describes weights corresponding measurement noise, such as the detector resolution, at step t.

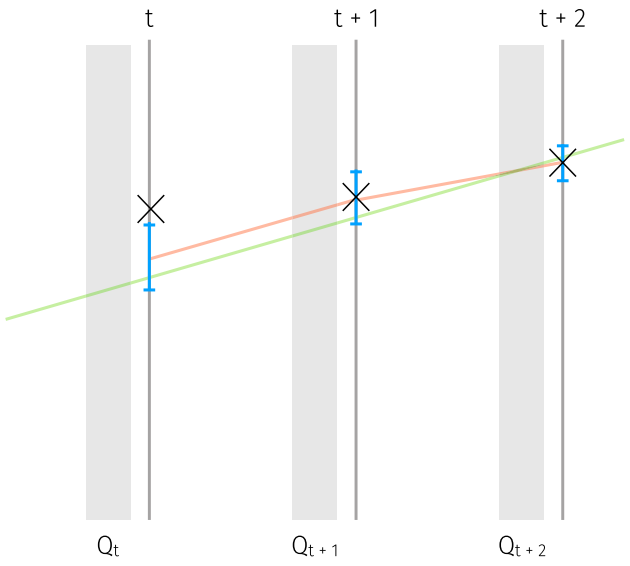

Fig. 9 Schematic of the Kálmán filter application with active detector planes (dark grey) with hits (crosses), and inactive medium (light grey). The Kálmán filter first calculates the extrapolation of the track state and uncertainty to the next detector plane (blue regions), and corrects this using the true hits and their uncertainties to form an estimate of the track state at the plane (red curve). Lastly, the most likely values of the track states and uncertainties at the planes are obtained in a backwards pass (green curve)

Up until this point, all information is updated in the forward direction, however information downstream can also be used to update upstream state estimates, to obtain a globally optimal set of states. To do this propagation, a backward transport operator is defined as

$$\mathbf{A}_t = \mathbf{C}_{t,\text{filt}}\mathbf{F}_t^\top \mathbf{C}_{t+1,\text{proj}}^{-1}, \tag{5}$$

which is used to perform the *smoothing* step in the backward direction and updating the state

$$\mathbf{p}_{t,\text{smooth}} = \mathbf{p}_{t,\text{filt}} + \mathbf{A}_t(\mathbf{p}_{t+1,\text{smooth}} - \mathbf{p}_{t+1,\text{proj}}), \tag{6}$$

and covariance matrix

$$\mathbf{C}_{t,\text{smooth}} = \mathbf{C}_{t,\text{filt}} + \mathbf{A}_t(\mathbf{C}_{t+1,\text{smooth}} - \mathbf{C}_{t+1,\text{proj}})\mathbf{A}_t^\top, \tag{7}$$

at t using the now smoothed state and covariance matrix at $t+1$.

The covariance matrix can also be used to form a χ^2 test statistic to determine the consistency of a hit with the fitted track,

$$\chi_t^2 = \mathbf{r}_t^T\mathbf{G}_t\mathbf{r}_t + (\mathbf{p}_{t,\text{filt}} - p_{t,\text{proj}})\mathbf{C}_{t,\text{proj}}^{-1}(\mathbf{p}_{t,\text{filt}} - p_{t,\text{proj}}), \tag{8}$$

where r_k is the residual,

$$\mathbf{r}_k = \mathbf{m} - \mathbf{H}\mathbf{p}_{t,\text{filt}}. \tag{9}$$

Kálmán Filter Configuration

To investigate the performance characteristics of a Kálmán filter implemented in Poplar on the IPU, a tracker with 2D active planes of 1m × 1m in $\hat{x} - \hat{y}$ is considered, separated by a homogeneous inactive medium that induces multiple scattering. Five of these planes are used, separated in $\hat{z}$ by $d = 1$m of the inactive medium, and indexed by t. Each of these detector planes record measured track hits, $\mathbf{m} = \{m_x, m_y\}$, discretised according to the physical resolution of the detector planes, σ.

No magnetic field is considered, however its inclusion would only result in a minor modification of the track state (to infer momentum) and inclusion of the magnetic field description in $\mathbf{F}$. It is assumed initially that each track registers a hit on each of the five planes, and the matching of hits to tracks is perfect. In reality, dummy hits can be introduced to the tracking algorithms, and tracks are often post-processed to find the most likely set, so neither of these effects compromise the generality of this proof of principle.

A state vector, $\mathbf{p}_t = \{x_t, \tan\theta_t, y_t, \tan\phi_t\}$, corresponding to the most likely values of the track x-position, x_t; y-position, t_t; tangent of the track slope in $\hat{x} - \hat{z}$, $\tan\theta$; and tangent of the track slope in $\hat{y} - \hat{z}$, $\tan\phi$; is estimated at each plane, t. It follows that the model parameters for such a system are

$$\mathbf{F} = \begin{bmatrix} 1 & d & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} 1/\sigma^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/\sigma^2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{10}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} z_0^2\theta_0^2 & z_0\theta_0^2 & z_0^2\theta_0^2 & z_0\theta_0^2 \\ z_0\theta_0^2 & \theta_0^2 & z_0\theta_0^2 & \theta_0^2 \\ z_0^2\theta_0^2 & z_0\theta_0^2 & z_0^2\theta_0^2 & z_0\theta_0^2 \\ z_0\theta_0^2 & \theta_0^2 & z_0\theta_0^2 & \theta_0^2 \end{bmatrix}, \tag{11}$$

where the parameterisation for $\mathbf{Q}$ is obtained from Ref. [70] disregarding higher order terms in the track slopes; z_0 is the material depth; and θ_0^2 is the variance of the multiple scattering angle.

The initial state for the first projection step is set to be equal to the hits on the first plane, $\mathbf{p}_{0,\text{proj}} = \{m_{0,x}, 0, m_{0,y}, 0\}$, and the covariance matrix set to equal the full uncertainty on the track state,

$$\mathbf{C}_{0,\text{proj}} = \begin{bmatrix} (\Delta x)^2 & (\Delta\tan\theta)^2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & (\Delta y)^2 & (\Delta\tan\phi)^2 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{12}$$

where $\Delta x = \Delta y = 1$m, and $\Delta\theta = \Delta\phi = 1$.

In this study, simulated particles are produced at (0, 0, 0) and travel in the positive $\hat{z}$ direction towards the detector

planes. At each plane, the particle interacts with the active detector material according to its projection on the $\hat{x} - \hat{y}$ plane of the detector, with a location that is subject to a random fluctuation in each direction depending on the total path length to simulate the effect of multiple scattering. Subsequently the location of the hit is discretised according to the granularity of the active detector area. These two effects determine the Kálmán-filter process and covariance matrices of the measurement uncertainty. An example of the simulated detector configuration can be seen in Fig. 10, with the corresponding hits and reconstructed track states.

Benchmarks

The Kálmán filter described in "Kálmán Filter Configuration" is implemented for the IPU hardware using the Poplar C++ SDK. To exploit the independence of the particle tracks, each track is assigned to a single IPU tile, where all operations in "Kálmán Filter Formalism" are performed. In principle, this results in 1, 216 Kálmán filter operations proceeding in parallel, however, optimal throughput is only achieved when several batches of tracks are copied to each tile initially, and then operated on sequentially. From Fig. 11, it can be seen that for batches of size greater than ~ 10 tracks, almost perfect parallelism is achieved, with a peak throughput of around 2.2×10^6 tracks per second for this configuration.

It is interesting to study the behaviour of the IPU implementation of the Kálmán filter with a workload that relies on program branch statements and random memory accesses. To this end, a modification of the above Kálmán filter configuration is implemented, where a proportion of hits are forced to be inconsistent with tracks they have been assigned to. This results in a large value of the χ^2 expression in Eq. 8. At each step the χ^2 value is evaluated, and if it is above a certain threshold, the state is not updated and the previous state is propagated to the next state under the assumption that no hit was observed at this stage.

On the IPU, this is implemented by a branch statement in the vertex code, which is executed on each tile separately. By way of comparison, an equivalent Kálmán filter configuration is also implemented in TensorFlow (`v2.1.0`) for execution on the GPU. In TensorFlow the subsequent filtering step is modified using a conditional gather-scatter update to the state and state propagation parameters. Despite the sub-optimal TensorFlow-based GPU implementation, it is instructive to compare the relative throughput in the case where the states are conditionally modified, and the case where no conditional execution is performed. On the IPU, the reduction in peak throughput is approximately half that of the GPU—where it operates at 91% of peak throughput in this case, compared to 80% for the GPU. This is likely because the conditional execution results in an

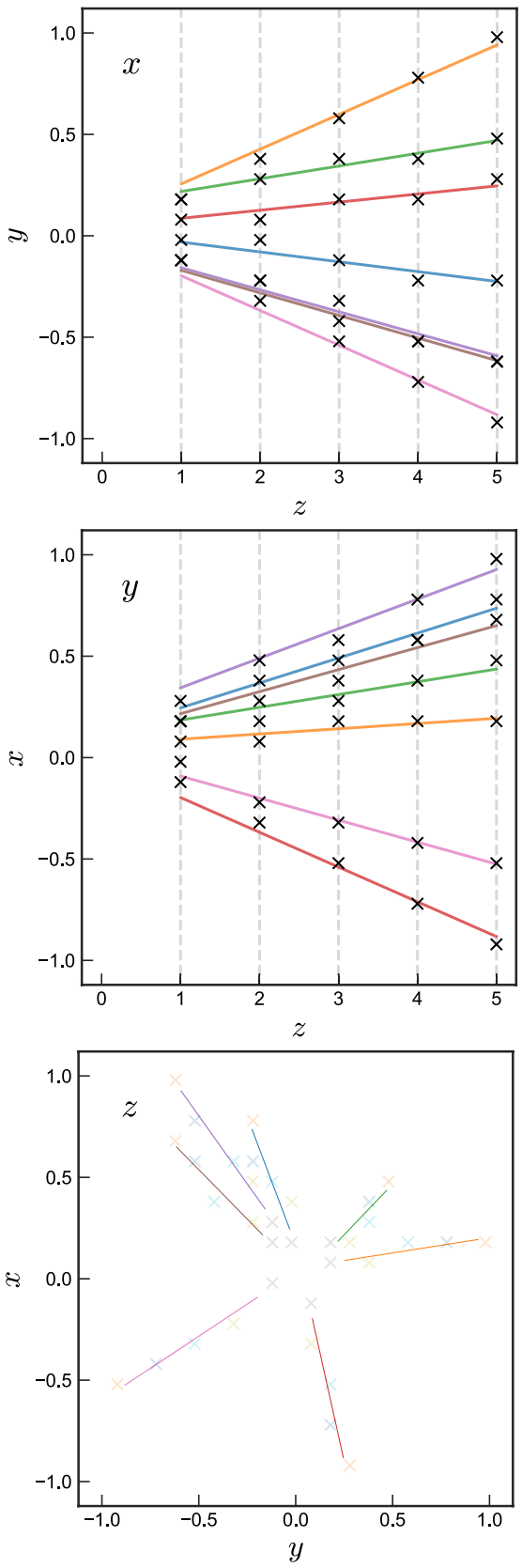

Fig. 10 Projections of the tracks (coloured lines) reconstructed from hits (crosses) using the detector and Kálmán filter configuration given in the text

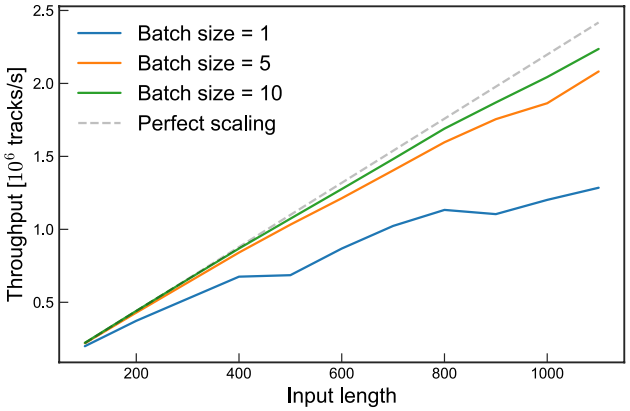

Fig. 11 Tracks per second processed by the Kálmán filter, as a function of the number of tracks processed in parallel on the tiles ('input length'). This is given for the cases where multiple 'batches' of this size are copied to the tiles before execution. The theoretical maximum throughput evolution as a function of input size is also indicated

inefficiency caused by divergence of parallel threads on the GPU ('warp divergence'), whereas on the IPU these execute independently.

Summary and Conclusions

This paper represents the first study of IPUs, a new processor type optimised for ML applications, in the context of particle physics. TensorFlow and PyTorch-based ML applications were used to compare the performance of a 1st generation IPU to that of a GPU of comparable price, but with twice the power consumption, and two high-end CPUs (see Table 1). Both GPU and IPU outperform the CPUs. The performance of the IPU and GPU is studied for a variety of neural network architectures and parameters. The batch size is identified as a key variable. For batch sizes accessible to both processors, the IPU out-performs the GPU, in some cases by orders of magnitude. For GAN event generation, large batch sizes are usually optimal. Here, the larger memory capacity of the GPU, allowing larger batch sizes, can be a decisive advantage. This is the case for the fully connected GAN architectures studied; for the convolutional- and locally connected GANs, the IPU generates events faster than the GPU despite using a smaller batch size. It is worth noting in this context that the second-generation IPU has triple the memory per tile compared to the first-generation IPU used here. In all cases, GANs train faster on the IPU. For applications with small batch size $\lesssim \mathcal{O}(100)$, such as neural network training or the track-correction algorithm studied, the IPU nearly always outperforms the GPU significantly.

This paper also presents the first implementation of a Kálmán filter on an IPU. The algorithm is implemented using Graphcore's Poplar SDK, and also on a GPU using TensorFlow. While the IPU implementation is much faster, the two implementations are too different for a fair comparison. Comparing the processing speeds on each processor with and without the final clean-up step indicates that the IPU's MIMD architecture is a significant advantage when executing conditional control-flow programs.

An important factor in considering the usefulness of IPUs in particle physics, alongside their performance, is the ease with which they can be programmed. The IPU software for the studies presented here [11] was written within less than 6 months of the group's first access to Graphcore's IPUs, by a small team of particle physics postdocs and Ph.D. students with no prior experience of IPU programming.

This first investigation of IPUs in a particle physics context suggests that IPUs, due to a combination of performance, flexibility and ease of programming, have the potential to play a central role in meeting the fast-increasing compute needs of particle physics. As promising as these results are, they can only be a starting point that motivates further, detailed study using realistic particle physics workflows.

Acknowledgements We are grateful to Graphcore for providing cloud access to their IPUs and for technical support. We also benefited from using the computational facilities of the Advanced Computing Research Centre, University of Bristol—http://www.bris.ac.uk/acrc. We would like to thank Dr Conor Fitzpatrick (University of Manchester) and Dr Mika Vesterinen (University of Warwick) for their careful reading of an earlier draft of this manuscript, and their helpful comments. This research was supported by the Science and Technology Facilities Research Council, UK.

Funding This research was funded by the Science and Technology Facilities Research Council, UK, and supported through in-kind contributions by Graphcore, and the Advanced Computing Research Centre, University of Bristol - http://www.bris.ac.uk/acrc.

Data Availability Statement This manuscript has associated data in a data repository. [Authors' comment:No associated data except for code. The associated code to replicate the studies in this paper can be found at: https://doi.org/10.5281/zenodo.3993387.]

Declarations

Conflict of interest Some authors of this publication are members of Graphcore, the manufacturer of the IPU evaluated in this paper. Graphcore supported the University of Bristol team by providing free access to its hardware and technical/software support. One member of the University of Bristol team became Graphcore employee in the course of this project.

Availability of data and material N/A.

Code availability The code used for this research can be accessed at the doi given in [11].

References

1. Aaij R et al (2018) Physics case for an LHCb Upgrade II—opportunities in flavour physics, and beyond, in the HL-LHC era
2. Abi B et al (2020) Deep Underground Neutrino Experiment (DUNE), far detector technical design report, volume III DUNE far detector technical coordination
3. Leggett C, Shapoval I (2018) Simulating HEP workflows on heterogeneous architectures. In: 14th International Conference on e-Science, p 343. https://doi.org/10.1109/eScience.2018.00087
4. Yeo B, Lee M, Kuno Y (2019) GPU-accelerated event reconstruction for the COMET phase-I experiment
5. Cenci R, Di Luca A, Lazzari F, Morello MJ, Punzi G (2020) Real-time reconstruction of long-lived particles at LHCb using FPGAs. J Phys 1525(1):012101. https://doi.org/10.1088/1742-6596/1525/1/012101
6. Lazzari F, Bassi G, Cenci R, Morello MJ, Punzi G (2020) Real-time cluster finding for LHCb silicon pixel VELO detector using FPGA. J Phys 1525(1):012044. https://doi.org/10.1088/1742-6596/1525/1/012044
7. Aaij R et al (2020) Allen: a high level trigger on GPUs for LHCb. Comput Softw Big Sci 4(1):7. https://doi.org/10.1007/s41781-020-00039-7
8. Andreassen R, Meadows B, de Silva M, Sokoloff M, Tomko K (2014) GooFit: a library for massively parallelising maximum-likelihood fits. J Phys 513:052003. https://doi.org/10.1088/1742-6596/513/5/052003
9. Morris A, Poluektov A, Mauri A, Merli A, Mathad A, Martinelli M (2018) Using TensorFlow for amplitude fits. In: PyHEP workshop. Sofia, Bulgaria. https://doi.org/10.5281/zenodo.1415413
10. Eschle J, Puig Navarro A, Silva Coutinho R, Serra N (2019) zfit: scalable pythonic fitting. https://doi.org/10.1016/j.softx.2020.100508
11. Mohan LRM, Marshall A, O'Hanlon D, Maddrell-Mander S (2020) dpohanlon/IPU4HEP. https://doi.org/10.5281/zenodo.3993387
12. Jia Z, Tillman B, Maggioni M, Scarpazza DP (2019) Dissecting the graphcore ipu architecture via microbenchmarking
13. Graphcore: Graphcore.ai (2020 (accessed 24 July, 2020)). https://www.graphcore.ai/
14. Intel: Intel Xeon Platinum 8168 specifications (2020 (accessed 18 Aug, 2020)). https://ark.intel.com/content/www/us/en/ark/products/120504/intel-xeon-platinum-8168-processor-33m-cache-2-70-ghz.html
15. Intel: Intel Xeon Processor E5-2680 v4 specifications (2020 (accessed 18 Aug, 2020)). https://ark.intel.com/content/www/us/en/ark/products/91754/intel-xeon-processor-e5-2680-v4-35m-cache-2-40-ghz.html
16. Nvidia: NVIDIA TESLA P100 specifications (2020 (accessed 18 Aug, 2020)). https://www.nvidia.com/en-gb/data-center/tesla-p100/
17. Graphcore: private communication
18. Graphcore (2020) Performance Benchmarks of the Graphcore IPU. https://www.graphcore.ai/benchmarks
19. Mathew G, Graphcore (2020) Accelerating Text to Speech Models with the IPU. https://www.graphcore.ai/posts/accelerating-text-to-speech-models-with-the-ipu
20. Masters D, Graphcore (2020) Delving deep into modern computer vision models. https://www.graphcore.ai/posts/introducing-second-generation-ipu-systems-for-ai-at-scale
21. Therhaag J (2012) TMVA: Toolkit for multivariate data analysis. AIP Conf Proc 1504(1):1013–1016. https://doi.org/10.1063/1.4771869
22. Nickolls J, Buck I, Garland M, Skadron K (2008) Scalable parallel programming with cuda. Queue 6:2. https://doi.org/10.1145/1365490.1365500
23. Abadi M et al (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. http://tensorflow.org/. Software available from tensorflow.org
24. Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, Lerer A (2017) Automatic differentiation in pytorch
25. Aaij R et al (2015) LHCb detector performance. Int J Mod Phys A 30(07):1530022. https://doi.org/10.1142/S0217751X15300227
26. Aaij R et al (2016) A new algorithm for identifying the flavour of B_s^0 mesons at LHCb. JINST 11(05):P05010. https://doi.org/10.1088/1748-0221/11/05/P05010
27. Aaij R et al (2013) The LHCb trigger and its performance in 2011. JINST 8:P04022. https://doi.org/10.1088/1748-0221/8/04/P04022
28. Gligorov V, Williams M (2013) Efficient, reliable and fast high-level triggering using a bonsai boosted decision tree. JINST 8:P02013. https://doi.org/10.1088/1748-0221/8/02/P02013
29. Rinnert K, Cristoforetti M (2019) Deep learning approach to track reconstruction in the upgraded VELO. EPJ Web Conf 214:06038. https://doi.org/10.1051/epjconf/201921406038
30. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Advances in neural information processing systems, pp 2672–2680
31. Karras T, Aila T, Laine S, Lehtinen J (2017) Progressive growing of gans for improved quality, stability, and variation. arXiv:1710.10196
32. Yu J, Lin Z, Yang J, Shen X, Lu X, Huang TS (2018) Generative image inpainting with contextual attention. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 5505–5514
33. Zhang H, Goodfellow I, Metaxas D, Odena A (2019) Self-attention generative adversarial networks. In: International Conference on Machine Learning, pp 7354–7363
34. Zhang H, Xu T, Li H, Zhang S, Wang X, Huang X, Metaxas DN (2017) Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In: Proceedings of the IEEE international conference on computer vision, pp 5907–5915
35. Isola P, Zhu JY, Zhou T, Efros AA (2016) Image-to-image translation with conditional adversarial networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 5967–5976
36. Ledig C, Theis L, Huszár F, Caballero JA, Aitken A, Tejani A, Totz J, Wang Z, Shi W (2017) Photo-realistic single image super-resolution using a generative adversarial network. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 105–114
37. de Oliveira L, Paganini M, Nachman B (2017) Learning particle physics by example: location-aware generative adversarial networks for physics synthesis. Comput Softw Big Sci 1(1):4
38. Ahdida C, Albanese R, Alexandrov A, Anokhina A, Aoki S, Arduini G, Atkin E, Azorskiy N, Back J, Bagulya A et al (2019) Fast simulation of muons produced at the ship experiment using generative adversarial networks. J Instrum 14(11):P11028

39. Di Sipio R, Giannelli MF, Haghighat SK, Palazzo S (2019) Dijetgan: a generative-adversarial network approach for the simulation of qcd dijet events at the lhc. J High Energy Phys 2019(8):110
40. Butter A, Plehn T, Winterhalder R (2019) How to GAN event subtraction
41. Arjona Martínez J, Nguyen TQ, Pierini M, Spiropulu M, Vlimant JR (2020) Particle Generative Adversarial Networks for full-event simulation at the LHC and their application to pileup description. J Phys 1525(1):012081. https://doi.org/10.1088/1742-6596/1525/1/012081
42. Carrazza S, Dreyer FA (2019) Lund jet images from generative and cycle-consistent adversarial networks. Eur Phys J C 79(11):979. https://doi.org/10.1140/epjc/s10052-019-7501-1
43. Butter A, Plehn T, Winterhalder R (2019) How to GAN LHC events. SciPost Phys. 7(6):075. https://doi.org/10.21468/SciPostPhys.7.6.075
44. Paganini M, de Oliveira L, Nachman B (2018) Calogan: Simulating 3d high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks. Phys Rev D 97(1):014021
45. Paganini M, de Oliveira L, Nachman B (2018) Accelerating science with generative adversarial networks: an application to 3d particle showers in multilayer calorimeters. Phys Rev Lett 120(4):042003
46. Maevskiy A, Derkach D, Kazeev N, Ustyuzhanin A, Artemev M, Anderlini L (2019) Fast data-driven simulation of Cherenkov detectors using Generative Adversarial Networks. In: 19th International Workshop on Advanced Computing and Analysis Techniques in Physics Research: Empowering the revolution: Bringing Machine Learning to High Performance Computing
47. Erdmann M, Glombitza J, Quast T (2019) Precise simulation of electromagnetic calorimeter showers using a Wasserstein Generative Adversarial Network. Comput Softw Big Sci 3(1):4. https://doi.org/10.1007/s41781-018-0019-7
48. Buhmann E, Diefenbacher S, Eren E, Gaede F, Kasieczka G, Korol A, Krüger K (2020) Getting High: high fidelity simulation of high granularity calorimeters with high speed
49. Bellagente M, Butter A, Kasieczka G, Plehn T, Winterhalder R (2020) How to GAN away detector effects. SciPost Phys 8(4):070. https://doi.org/10.21468/SciPostPhys.8.4.070
50. Ghosh A (2020) Deep generative models for fast shower simulation in ATLAS. J Phys 1525(1):012077. https://doi.org/10.1088/1742-6596/1525/1/012077
51. Carminati F, Khattak G, Loncar V, Nguyen TQ, Pierini M, Da Rocha RB, Samaras-Tsakiris K, Vallecorsa S, Vlimant JR (2020) Generative Adversarial Networks for fast simulation. J Phys Conf Ser 1525(1):012064. https://doi.org/10.1088/1742-6596/1525/1/012064
52. Belayneh D et al (2020) Calorimetry with deep learning: particle simulation and reconstruction for collider physics. Eur Phys J C 80(7):688. https://doi.org/10.1140/epjc/s10052-020-8251-9
53. Karavakis E et al (2014) Common accounting system for monitoring the atlas distributed computing resources. J Phys Conf Ser 513:062024
54. Apollinari G, Béjar Alonso I, Brüning O, Fessia P, Lamont M, Rossi L, Tavian L (2017) High-luminosity large hadron collider (hl-lhc): technical design report v. 0.1. cern yellow reports: Monographs. cern, geneva
55. Anelli M, Aoki S, Arduini G, Back J, Bagulya A, Baldini W, Baranov A, Barker G, Barsuk S, Battistin M et al (2015) A facility to search for hidden particles (ship) at the cern sps. arXiv preprint arXiv:1504.04956
56. Canal P et al (2016) GeantV: from CPU to accelerators. PoS ICHEP2016. https://doi.org/10.22323/1.282.0177
57. Amadio G et al (2020) GeantV: Results from the prototype of concurrent vector particle transport simulation in HEP
58. Albrecht J, Alves AA, Amadio G, Andronico G, Anh-Ky N, Aphecetche L, Apostolakis J, Asai M, Atzori L, Babik M et al (2019) A roadmap for hep software and computing r&d for the 2020s. Comput Softw Big Sci 3(1):7
59. Musella P, Pandolfi F (2018) Fast and accurate simulation of particle detectors using generative adversarial networks. Comput Softw Big Sci 2(1):8
60. Tanabashi M et al. (2018) Review of particle physics. Phys Rev D98:030001. https://doi.org/10.1103/PhysRevD.98.030001
61. Ronneberger O, Fischer P, Brox T (2015) U-net: Convolutional networks for biomedical image segmentation. arXiv:1505.04597
62. Identification of Jets Containing *b*-Hadrons with Recurrent Neural Networks at the ATLAS Experiment. Tech. Rep. ATL-PHYS-PUB-2017-003, CERN, Geneva (2017). https://cds.cern.ch/record/2255226
63. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
64. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L et al (2019) Pytorch: an imperative style, high-performance deep learning library. In: Advances in neural information processing systems, pp 8026–8037
65. Bai J, Lu F, Zhang K et al (2019) Onnx: Open neural network exchange. https://github.com/onnx/onnx
66. Chetlur S, Woolley C, Vandermersch P, Cohen J, Tran J, Catanzaro B, Shelhamer E (2014) cudnn: Efficient primitives for deep learning. arXiv preprint arXiv:1410.0759
67. Campora Perez DH (2017) LHCb Kalman ïňĄlter cross architecture studies. J Phys 898(LHCb-PROC-2017-041. CERN-LHCb-PROC-2017-041. 3):032052. 8. https://doi.org/10.1088/1742-6596/898/3/032052. https://cds.cern.ch/record/2292435
68. Fruhwirth R (1987) Application of Kalman filtering to track and vertex fitting. Nucl Instrum Meth A 262:444–450. https://doi.org/10.1016/0168-9002(87)90887-4
69. Hernando JA The Kalman filter technique applied to track fitting in GLAST http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.1880
70. Wolin E, Ho L (1993) Covariance matrices for track fitting with the Kalman filter. Nucl Instrum Meth A 329:493–500. https://doi.org/10.1016/0168-9002(93)91285-U