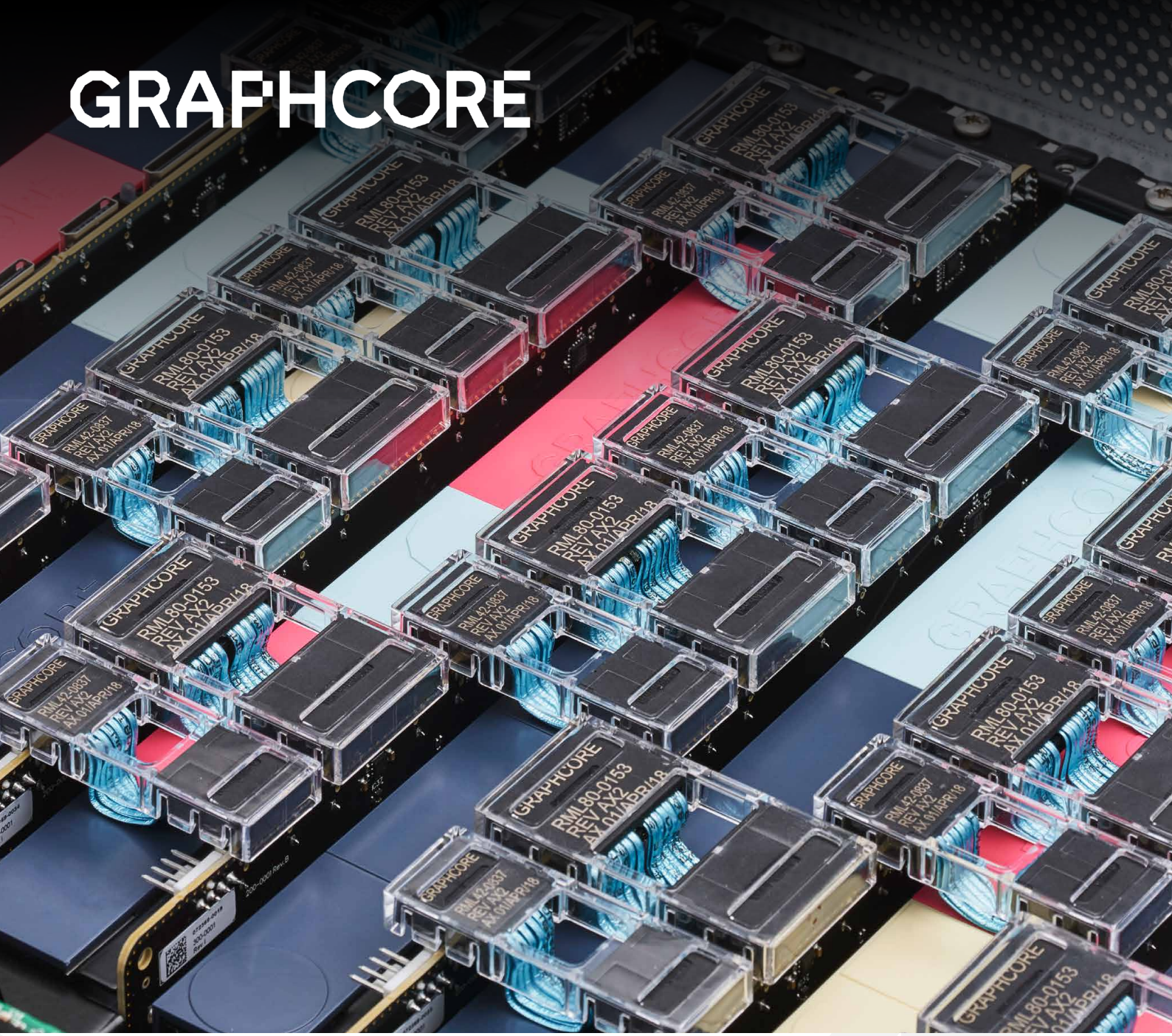
GRAPHCORE




# DELL DSS8440 GRAPHCORE IPU SERVER

White Paper

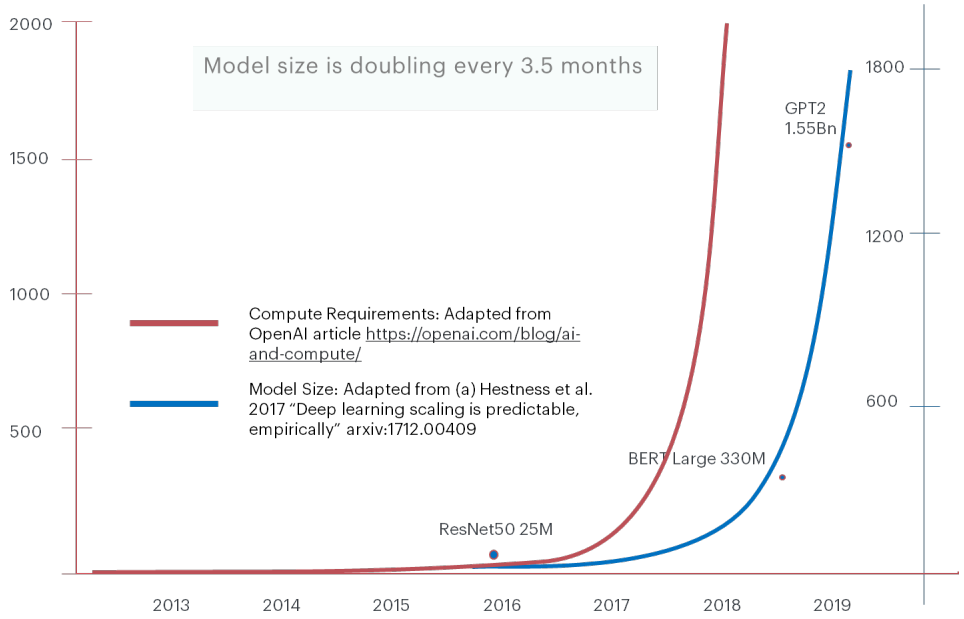
# TABLE OF CONTENTS

# MACHINE INTELLIGENCE WITH THE IPU

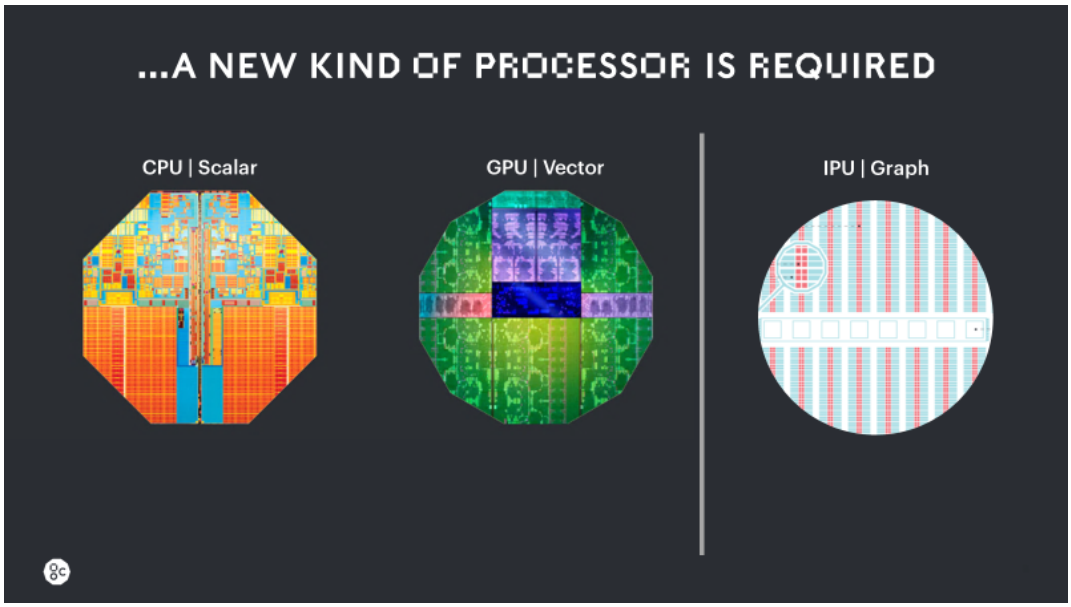
For over 70 years we have told computers what to do step by step in a program. However, now, with machine intelligence, computers are starting to learn from data. This represents the biggest shift we have seen in computers since the birth of the microprocessor or perhaps since the arrival of electronic computers themselves.

In just the last few years we have seen massive advances in machine intelligence systems. Using advanced machine learning we can now accurately recognize images and we are building models that can understand the structure of language. These systems are already having a massive impact. However, we are just at the very beginning. New types of machine intelligence models are possible which deliver much higher accuracy and efficiency for perception and which will also help us build machine intelligence systems that can outperform humans at specific tasks. Transformer based Attention networks such as BERT, for example, will need higher efficiency approaches such as leveraging sparse model structures in the future so that they can more efficiently scale to billions or trillions of parameters, which will be required for richer language understanding. We will need to develop techniques that will allow us to build machines that are able to learn from experience.

The figure below illustrates that, in general, the size of machine intelligence models is growing exponentially, doubling about every 3.5 months. Add to this the fact that compute requirements scale vs model size at something just over a power of two and this adds up to truly significant growth for compute requirements. Model size increases to drive increases in model accuracy. Innovation in machine intelligence is all about getting more accuracy with fewer and fewer parameters, hence the need for techniques like separable convolutions or sparse transformers, for example.

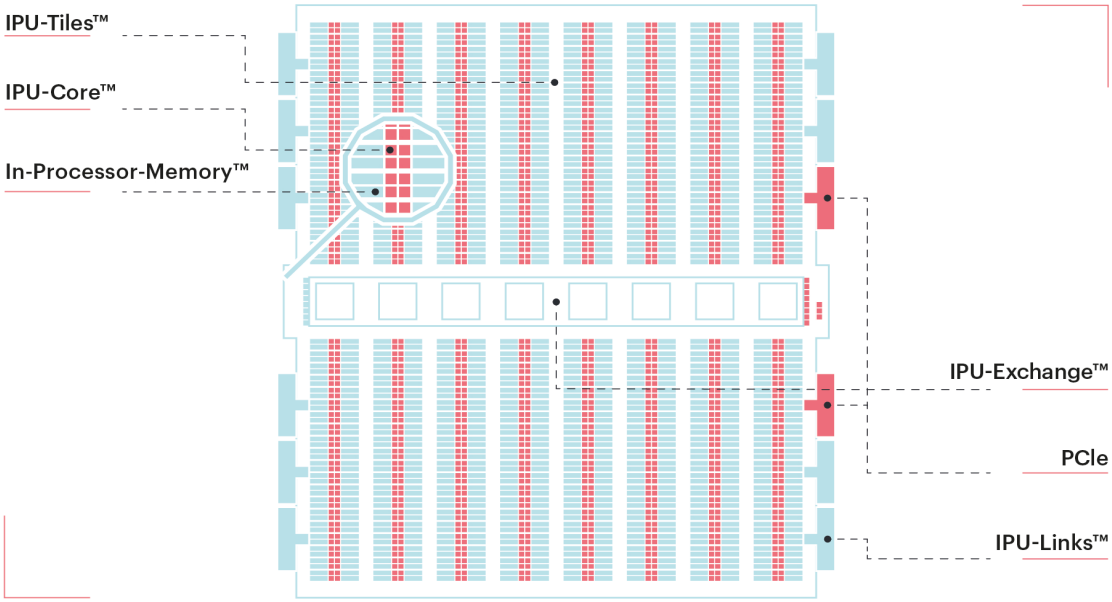


At Graphcore we have been able to work with many of the leading innovators in machine intelligence. They have all told us that today's CPUs and GPUs are holding them back. What they need is a new type of processor that can efficiently support much more complex knowledge models for both training and inference. This is why we have created the Intelligence Processing Unit – the IPU. The IPU is built from the ground up to meet these innovators needs and to help them make new breakthroughs in machine intelligence.




The IPU supports massively parallel processing across thousands of independent processing threads. This is coupled with massive memory bandwidth to enable tens of thousands of parallel threads all running independently.

**The Graphcore Colossus GC2 IPU Architecture**



The Graphcore IPU is designed from the ground up for this new world of Machine Intelligence with several key aspects to its approach:

- it is designed for flexible address access across thousands of independent processing threads
- the whole knowledge model is held inside the processor using In-Processor-Memory to maximize memory bandwidth and minimize latency
- the IPU-Cores support very high-performance mixed precision floating point arithmetic that is optimized for work on tensors and includes hardware support for generating random numbers
- each IPU-Core can run up to 6 independent hardware program threads making a total of over 7,000 separate programs working in parallel on the knowledge model in a single IPU
- the IPU supports existing conventional DNN models but is also opening up new avenues for innovation in Machine Intelligence such as probabilistic systems and evolution strategies
- Importantly the IPU is efficient for both training and inference and provides the lowest latency for Machine Intelligence compute

On each IPU there are 1,216 high performance machine learning processor cores (IPU-Cores) each of which contains 6 processor threads. There are therefore 7,296 processor threads running in parallel at full speed during compute, with each IPU-Core delivering over 100 gigaflops of compute performance.

Each IPU-Core is tightly coupled to 256kB of very fast local In-Processor-Memory and together they make an IPU-Tile. The 6 processor threads can access the In-Processor-Memory directly. Overall the IPU has roughly 300MB of memory with a memory bandwidth of 45TBps and with no off-chip memory the IPU avoids memory bandwidth issues. EEE FP32 and FP16 data formats are supported and the IPU can handle all layers in a deep neural network and is capable of 250 teraflops mixed precision 16.32bit arithmetic compute.

Every core connects directly to the IPU-Exchange, a crossbar in the middle of the die which can transfer 62.5Tbps of data. The IPU connects to the host processor through a 16-lane PCI Express interface. There are 10 high-speed IPU-Link™ ports, each port has a bandwidth of 160Gbps in each direction and 8 lanes. The IPU-Link ports allow IPUs to exchange data directly, without going through the host processor or host memory, enabling multiple IPUs to work efficiently together with ~450GB/s of IPU-Link inter-IPU bandwidth. There is also hardware support for random number generation for configural entropy injection.

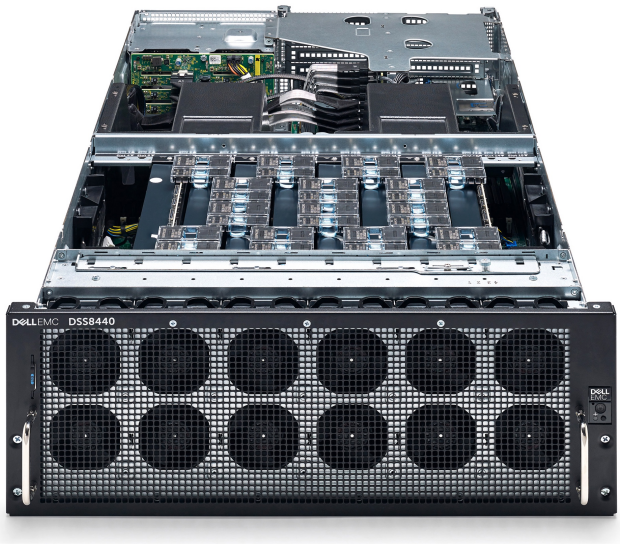
# GRAPHCORE C2 IPU-PROCESSOR CARD

The Graphcore C2 IPU-Processor PCIe Card is a dual-slot, full-height PCI Express Gen3/4 card containing two IPUs. There are 80 IPU-Links, each Link at 32Gbps, for a total of about 2.5Tbps or 450GB/s of chip to chip bandwidth. On the C2 card, 192GB/s of IPU-Link bandwidth is used to connect the two IPUs on the C2 card itself, whilst 256GB/s of IPU bandwidth is used to connect C2 cards together enabling the IPU-Processors across multiple C2 cards to work as a contiguous processing resource (see the DSS 8440 system topology figure below).

With two IPU-Processors on the card, there is a total of 200 TFLOPS of processing capability over 2416 individual processor cores. This results in a 315W card passively cooled in the DSS 8440 IPU-Server.

# DELL DSS 8440 IPU-SERVER

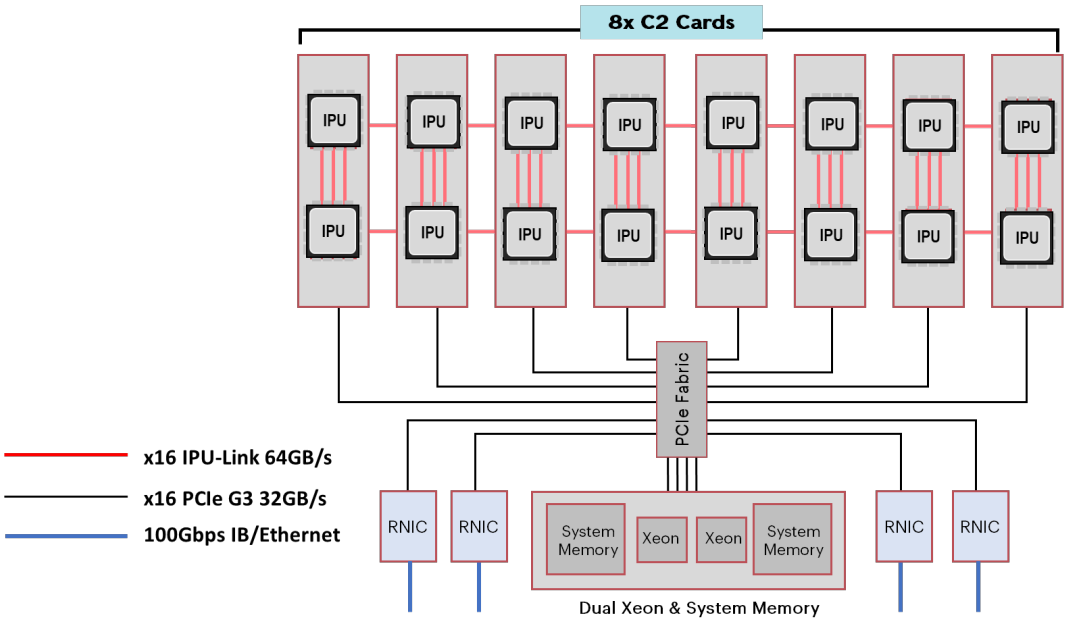
The DSS 8440 IPU-Server is a two socket, 4U server designed to deliver exceptionally high performance for machine intelligence applications, with eight Graphcore dual-IPU C2 cards providing a total of 1.6PetaFLOPs of mixed precision machine intelligence compute power.




The DSS 8440 IPU-Server utilizes an IPU-Link “ladder” topology with 192GB/s “rungs” between the two IPUs on the C2 cards and 256GB/s total bidirectional bandwidth between C2 cards. This configuration allows for high speed sharing of model parameters, activations, or data between IPUs and across C2 cards.

The C2 cards are connected to the DSS 8440 IPU-Server Xeon host CPUs via a high speed PCIe Gen3 switched fabric. There are actually four separate PCIe switches in the DSS 8440 IPU-Server but are shown topologically here as a “black box” PCIe fabric. As will be explained in more depth in the section on the Poplar SDK, the host CPUs are not directly involved in Machine Intelligence processing tasks since the IPUs are able to run their ML programs completely independently of the host.

The DSS 8440 IPU-Server is also equipped with extensive local storage and the IPUs are able to make direct use of this host system memory which is especially useful for applications involving large (multi-billion parameter) models.



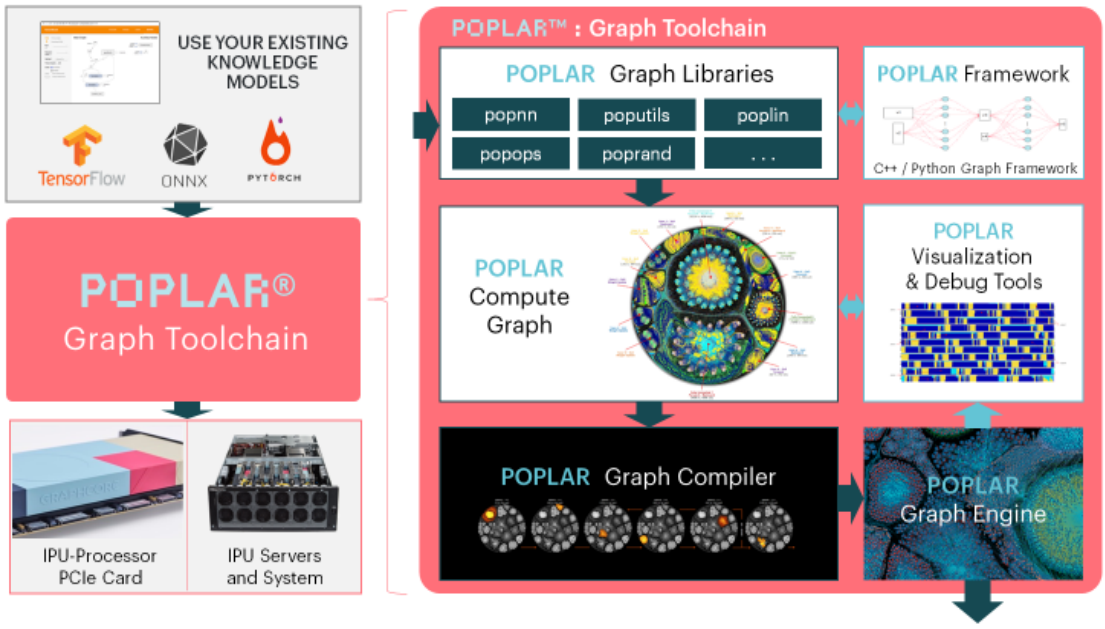

For distributed training applications, the DSS 8440 IPU-Server also provides multiple 100Gbps network links for server-to-server scalability, for example using the default dual 100Gpbs InfiniBand adapters. This view of the DSS 8440 IPU-Server topology is used frequently when discussing machine intelligence model implementations either for training or inference in data and model parallel configurations.

With all these attributes it is clear that the DSS 8440 IPU-Server is ideal for machine intelligence training and inference applications.

## POPLAR® SDK

Machine intelligence applications are built for the DSS 8440 IPU-Server using Graphcore's Poplar SDK, providing out-of-the-box state of the art performance on today's leading edge models. The Poplar SDK works with widely used frameworks like TensorFlow, Pytorch, and ONNX. It accepts high level machine intelligence graph descriptions and compiles the optimized Poplar graph and an associated control program, both of which are loaded onto the IPU(s). The large In-Processor-Memory means that entire models can be loaded on an IPU taking full advantage of the massive In-Processor-Memory bandwidth. The Poplar SDK also offers rich support for model parallelism and pipelining approaches, for example, to support next generation NLP models with billions of parameters. These various Poplar SDK features are described in detail below.

The Poplar SDK supports a range of standard frameworks. With TensorFlow, the Poplar SDK directly accepts XLA graphs and compiles the XLA output as a Poplar graph and control program. The output of other ML frameworks are accepted as intermediate representations by PopART™, the Poplar Advanced Runtime. PopART, besides being a ML framework in its own right, also provides wide flexibility in its ability to connect with other ML frameworks.




Poplar standard libraries, PopLibs, has a wide range of operators and functions available for customization by developers. Source code is already available to customers to use and to contribute to and PopLibs will be made open source during 2020 to enable the IPU community to openly share innovations on the IPU. PopLibs features:

- **popnn** - Neural network functions (activation functions, pooling, loss)
- **poplin** - Matrix multiply and convolution functions
- **popops** - Pointwise and reduction operators

- **poprand** - Random number functions
- **poputil** - Utility functions for building graphs
- **popsolver** – model solving functions

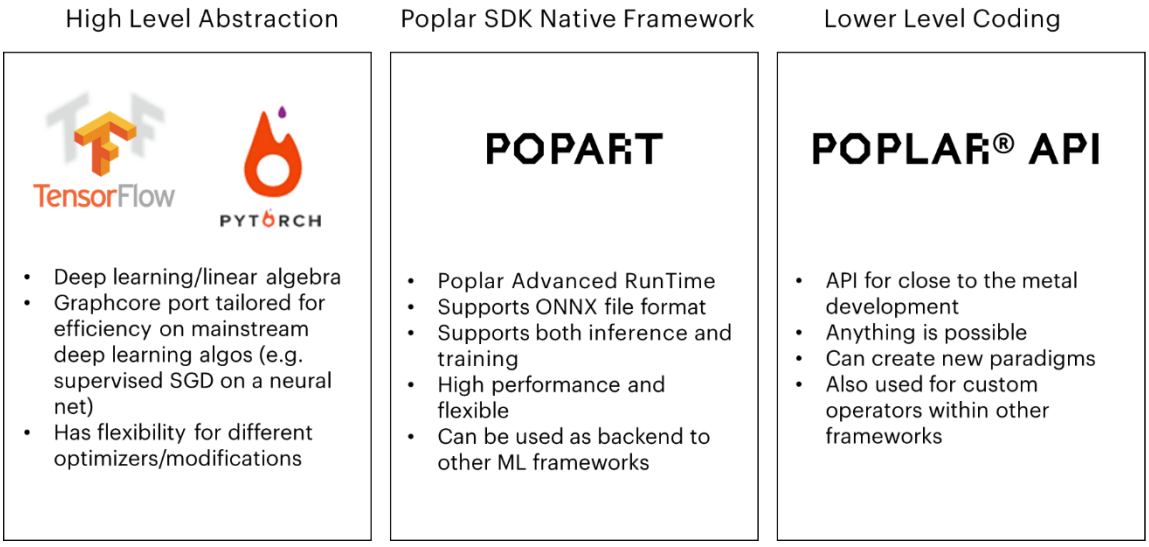
## Flexibility with Poplar SDK

The Poplar SDK provides three different levels of development experience depending on the background and goals of the developer. The figure below describes these three different entry points into the Poplar SDK:

- **High Level Abstraction** – this provides an easy path for developers with existing graphs described in standard ML framework tools like TensorFlow, Pytorch, or ONNX. The existing ML framework model can be implemented and optimized for the IPU through the Poplar SDK with no changes.

- **PopART** - Poplar Advanced RunTime is the Poplar SDK native graph runtime that handles high level computation graphs for both inference and training. PopART offers support for IPU specific graph optimizations and other support bringing the developer closer to the target hardware.

- **Lower Level Coding** – Poplar API is a direct C++ coding experience allowing developers to directly code graphs for the IPU, explicitly specifying tensors, vertices and edges with complete control. This might be used to create custom operators, custom network layers or for other optimization purposes. Developers can access PopLibs source code for examples and inspiration to accelerate development.



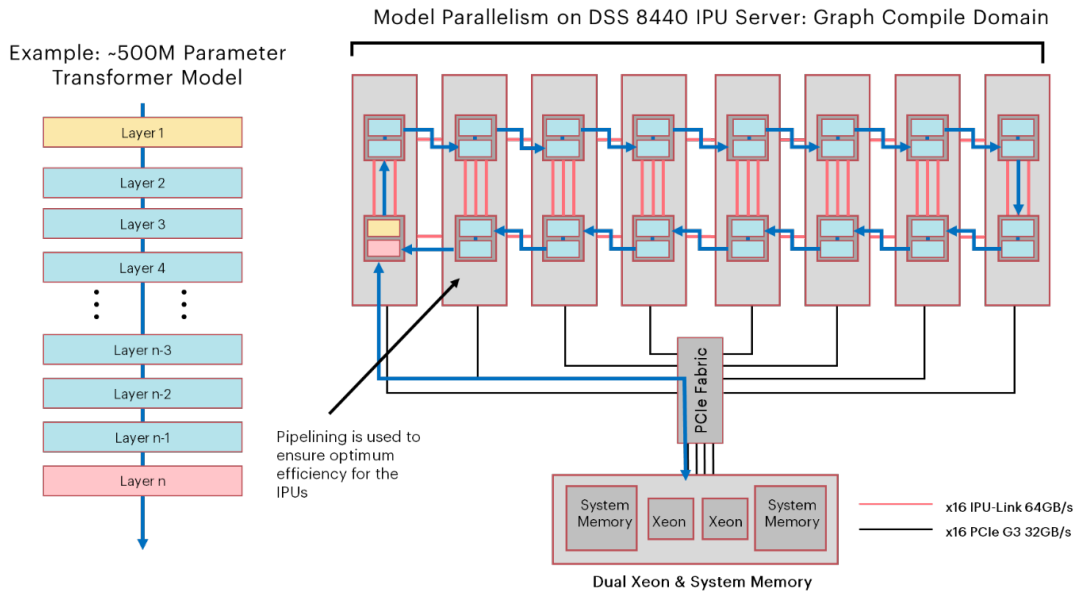
# MACHINE INTELLIGENCE WITH THE IPU-SERVER

The DSS 8440 IPU-Server supports new possibilities for implementing machine intelligence knowledge models for both training and inference, spanning the usual data parallel architectures as well as efficient pipelined model parallel architectures. The decision on the best implementation approach is of course very model dependent. However, since the IPU and its Poplar SDK offer an entirely new platform for machine intelligence processing, users will soon discover that there is a world of opportunities to better optimize model performance. The market has become somewhat conditioned to a conventional processing architecture but at the same time is eager to make breakthroughs in machine intelligence. To some extent innovations become more apparent by viewing model architectures through new eyes with the IPU.

The unique architectural strengths of the IPU are optimally leveraged through innovative features in the Poplar SDK.

## Model parallelism with Graph Compile Domain (GCD)

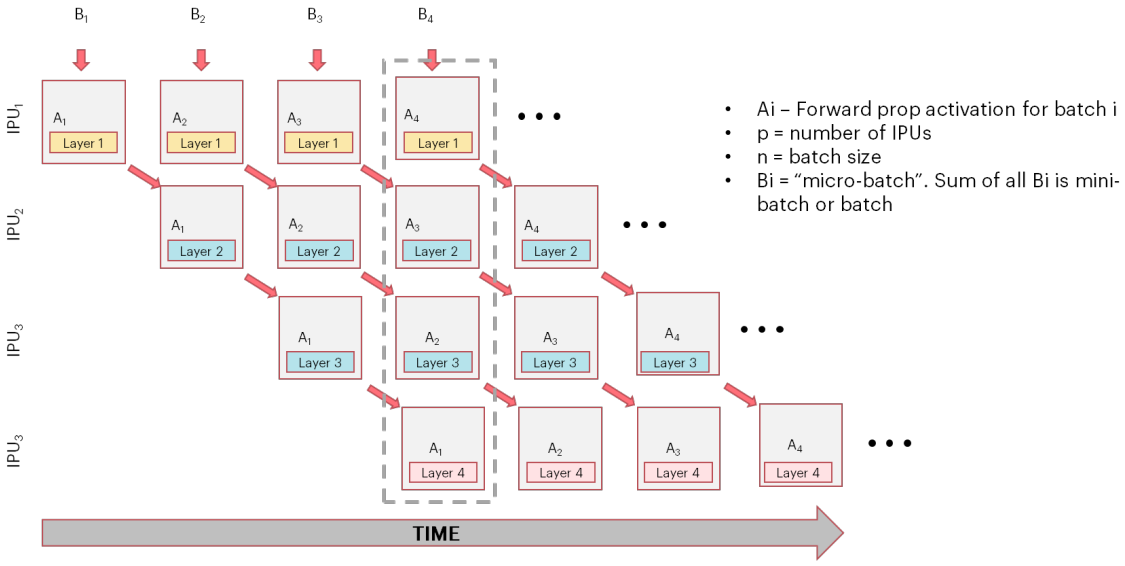
The Graph Compile Domain (GCD) defines the number of IPUs over which the Poplar SDK is able to compile a model parallel implementation, enabling larger models to take full advantage of the high bandwidth IPU In-Processor-Memory. The GCD encompasses 16 IPUs in the DSS 8440 IPU-Server which means that large models can be partitioned by “sharding” the model over all 16 IPUs of the server. Small to moderate models can fit entirely in the In-Processor-Memory of a single IPU, enabling data parallel processing across the 16 IPUs.



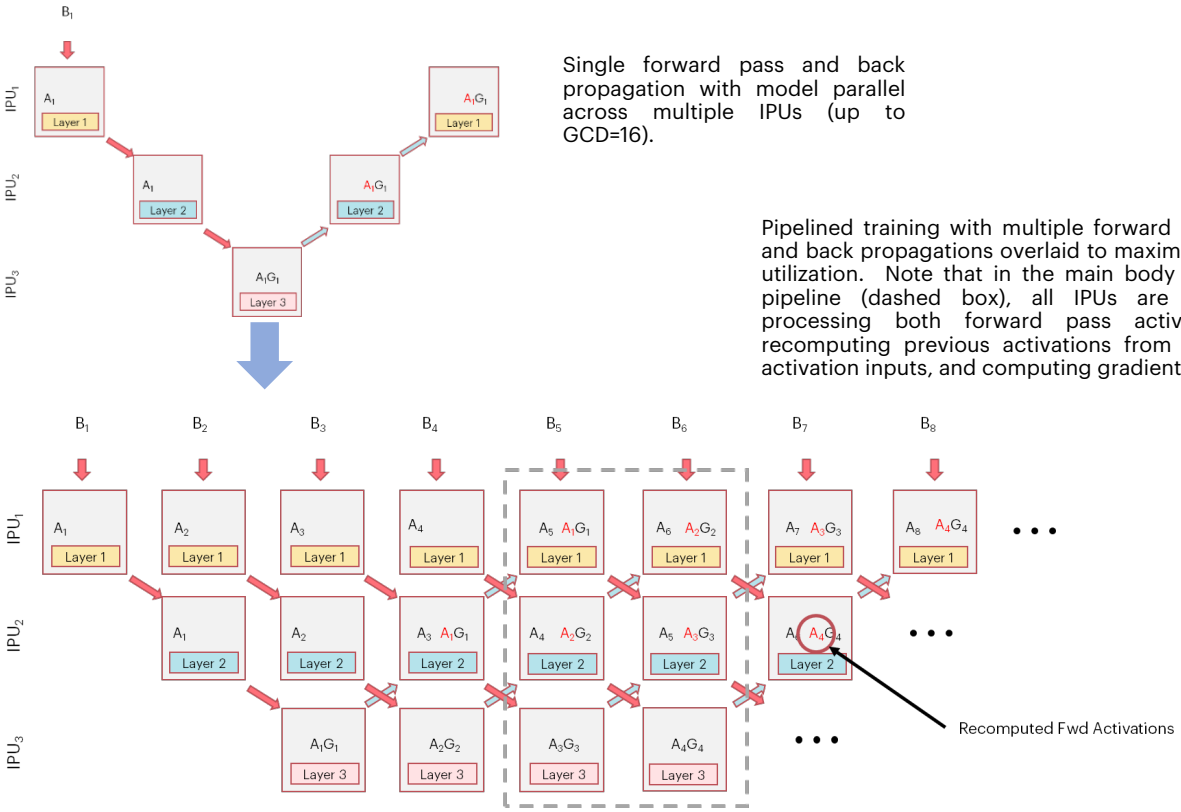

Combined with pipelining (see below) this is particularly useful for minimising latency and maximizing compute across a very large graph. The figure above illustrates a large transformer-based model (typical of an NLP use case) where the model has been sharded across all 16 IPUs of the DSS 8440 IPU-Server.

## Pipelining

Combined with model parallel implementations (through GCD), pipelining ensures that all IPUs involved in model parallel processing are maximally utilized. This results in high processor efficiency and improved performance in throughput and latency. The figure below highlights the use of pipelining in a model parallel approach for inference (the dashed box indicates the point in the main body of the pipeline where all IPUs are maximally utilized). The pipeline can be any depth across (up to 16 IPUs in the DSS 8440 IPU-Server). The first IPU in the model parallel series (IPU1) accepts the first batch (B1) as activations are passed onto subsequent IPUs. IPU1 continues to accept new batches, passing activations on for those batches in turn. The second IPU accepts activations and processes the next layer(s) and so on down the chain. Pipelining enables all the IPUs to be active and by batch 4 (B4) all IPUs are actively processing in parallel. This processing pipeline is relatively straightforward for inference but becomes more complex with training due to back propagation.

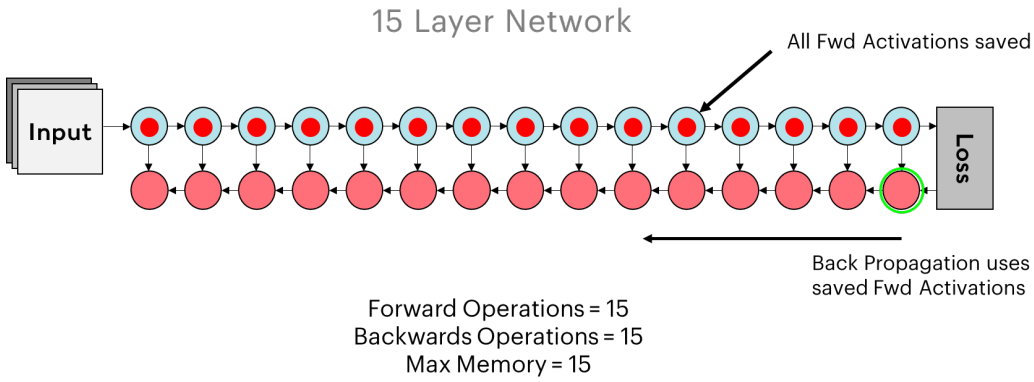


For training, pipelining needs to accommodate both the forward pass as well as the back propagation and weight updates. The figure below shows a single forward pass and back propagation and then the full pipeline with overlaid batches in parallel. The dashed box captures the main body of the pipeline (which can be of any depth, greater depth increasing batch size) where the IPUs involved are maximally utilized processing forward pass activations, recomputing previous activations from stored activation inputs, and calculating gradient updates. Recomputing (see below) is a key enabler for this approach by allowing the developer to take full advantage of the high bandwidth In-Processor-Memory.

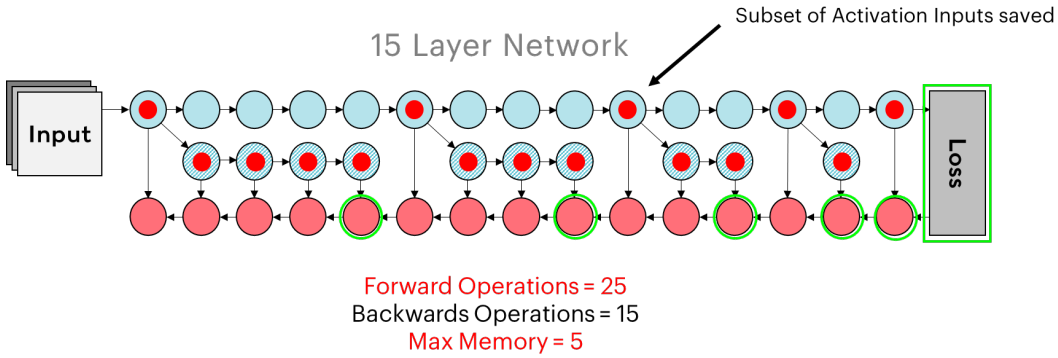


## Recomputation

A major strength of the IPU architecture is the high bandwidth/low latency In-Processor-Memory. Recomputation is one of the features in the Poplar SDK that maximizes both IPU memory resources and the IPUs high performance compute capabilities. In a multi-layer model, activations are computed from layer to layer and are typically saved as intermediate results. With recomputation, the Poplar SDK selects specific activation inputs and uses those to recompute activations as required on the backward pass. The first figure below illustrates a typical approach where all forward activations are saved to support weight update calculations on the backward pass.




The Poplar SDK makes more efficient use of the valuable In-Processor-Memory by saving select activation inputs optimizing on memory savings vs TFLOP expenditure in the recomputation. The figure below shows how the subset of activation inputs that are saved can be used to recompute all the necessary activation history for the backward pass calculation of the weight updates.

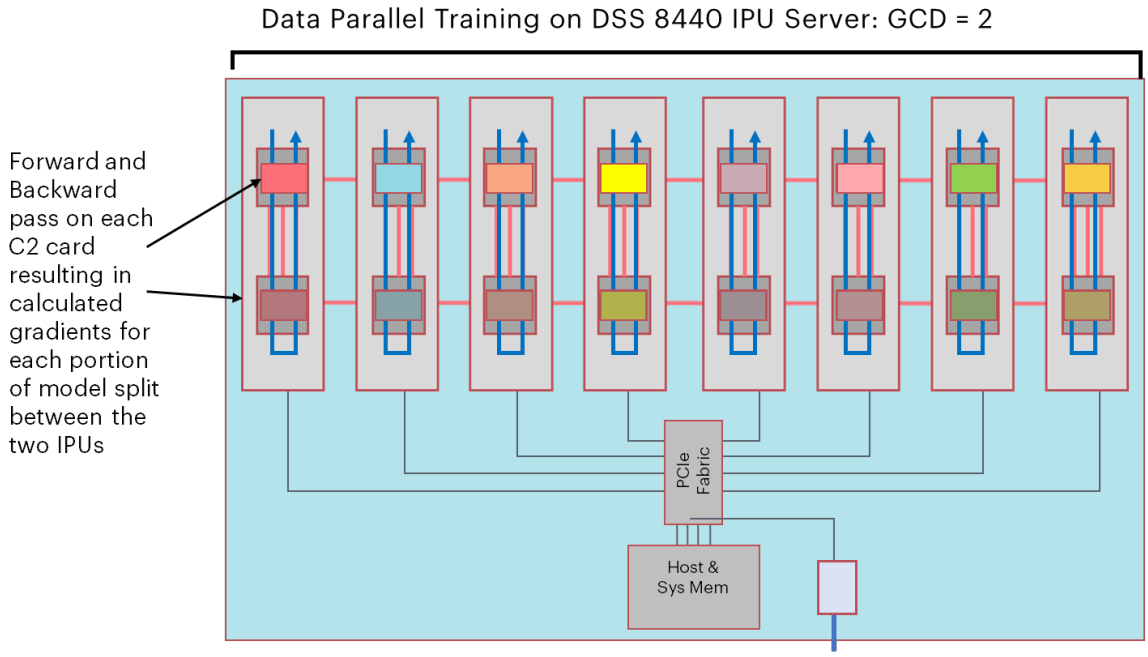



Recomputation provides several benefits in combination with the unique strengths of the IPU architecture:

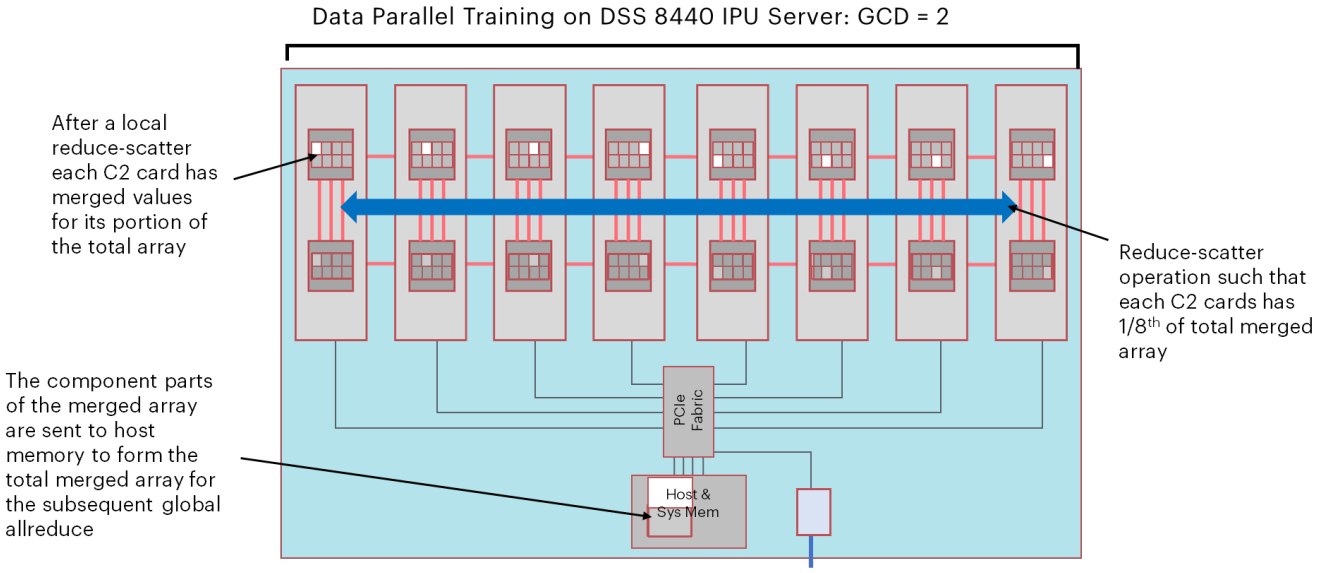
- Recomputation can reduce the memory required to train NNs by up to 5x for ResNets and DenseNets, allowing optimal use of the tightly coupled IPU In-Processor-Memory
- Poplar SDK supports constrained methods to minimize the FLOPs for a target memory overhead
- Additional wall clock time required to compute additional FLOPs may be offset by using larger batches

## Data Parallel Training or Multiple Replica Training

The first step in training DNNs is a forward inference pass of input data. The amount of input data across all of the model replicas is determined by the minibatch size; the error (loss) after the forward pass is computed by comparing the results to correct answers. This loss is used in the back propagation step to calculate the weight gradients and updates to the model parameters. Training with multiple replicas of the model in the DSS 8440 IPU-Server (and by extension across multiple DSS 8440 IPU-Servers using the 100Gbps port available on the server) is data parallel training and can be combined with model parallel where each model replica is also distributed across more than one IPU (GCD >1). The figure below shows a data parallel training example where GCD = 2 (each model replica across the 2 IPUs in a C2 card). Since each of the model replicas have worked with different input data in parallel, the calculated gradient arrays from the back propagation will differ in each case (indicated by the different coloured boxes in each "top" IPU in the figure above).




The goal is to use an allreduce process to combine all of the results into a single model parameter update across all of the model replicas (indicated by the white coloured boxes in the figure below). The Poplar SDK supports different allreduce implementations within the DSS 8440 IPU-Server.




Once the local allreduce is complete then a global allreduce is accomplished with Infiniband support.

## InfiniBand for scaling multiple DSS 8440 IPU-Servers

Along with using IPU-Link for high bandwidth and low latency intra-server communication, the DSS 8440 IPU-Server is able to use a single 100Gbps InfiniBand adapter for flexible scale-out of multiple servers with a total 200Gbps of bidirectional bandwidth.

# SUMMARY

Over the last two years Graphcore has been working with major companies, as well as many of the world's leading innovators, to develop the IPU processor and Poplar software stack to support both training and inference. Poplar allows innovators to use existing high-level machine learning frameworks such as TensorFlow, ONNX, and PyTorch to easily use existing models on the IPU.

Out of the box, the IPU is able to achieve state of the art performance on deep learning models. We invite to you to explore our public benchmarks at https://github.com/graphcore and to begin to explore a new addition to the Machine Intelligence toolkit: the Graphcore IPU.

Graphcore believes that the information hereby provided is accurate as of the date of publication.  As time passes and our technology evolves, this information may become obsolete, incomplete or inaccurate.  Please check back frequently to be certain that you have the most up to date information.

# GET STARTED WITH THE IPU

Learn more about how
Graphcore can accelerate your
next machine intelligence project

graphcore.ai