SCALABLE MACHINE INTELLIGENCE SYSTEMS

GRAPHCORE

WWW.GRAPHCORE.AI

SCALABLE IPU SYSTEMS

SILICON

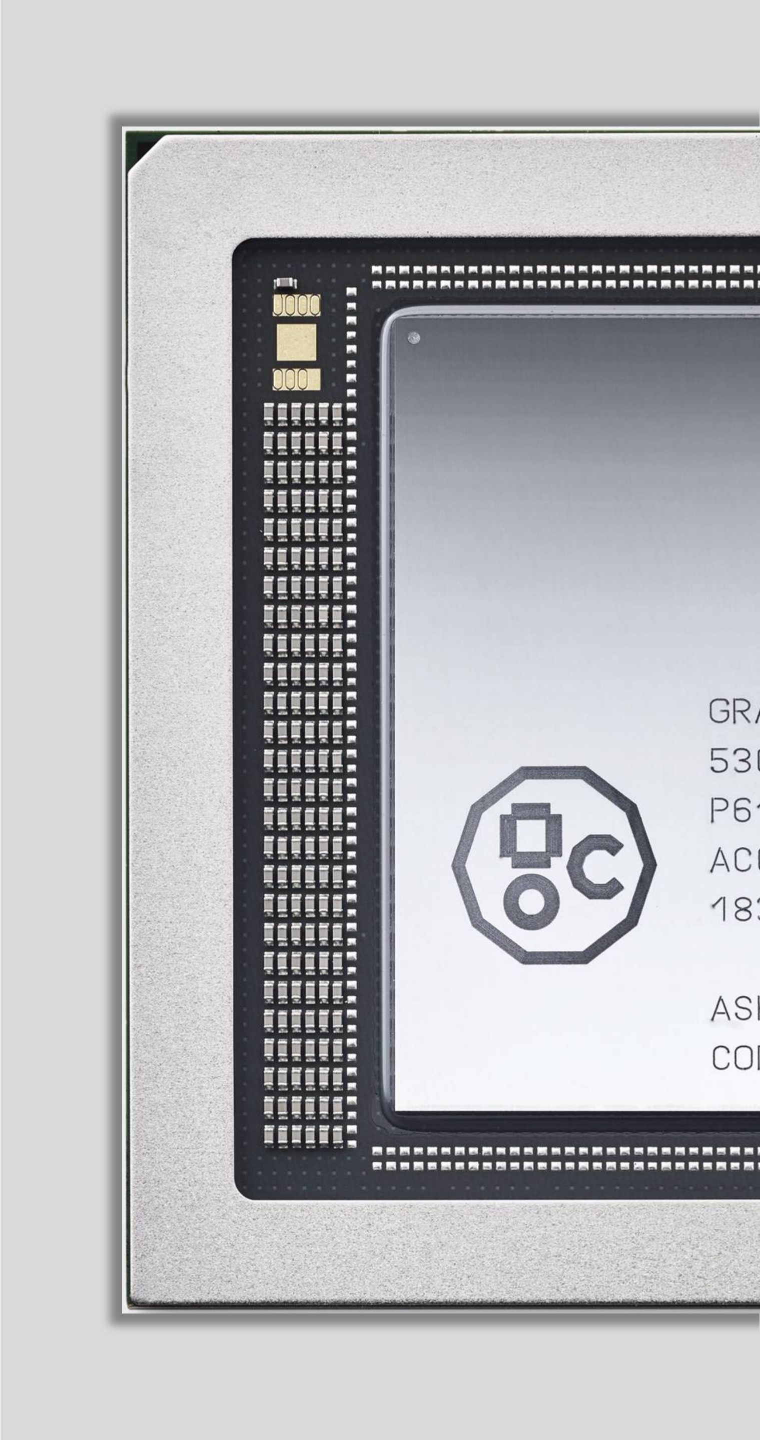
A MULTI-GENERATIONAL SILICON ARCHITECTURE PROVIDING OPTIMIZED SUPPORT FOR HIGH PERFORMANCE MACHINE INTELLIGENCE APPLICATIONS AND WORKLOADS AT SCALE

PLATFORMS

HARDWARE PLATFORMS DESIGNED TO DEPLOY IPU DEVICES WHICH ENABLE OPTIMIZED APPLICATIONS TO EXECUTE EFFICIENTLY IN SUPPORT OF INDUSTRY STANDARD DEPLOYMENT USE CASES

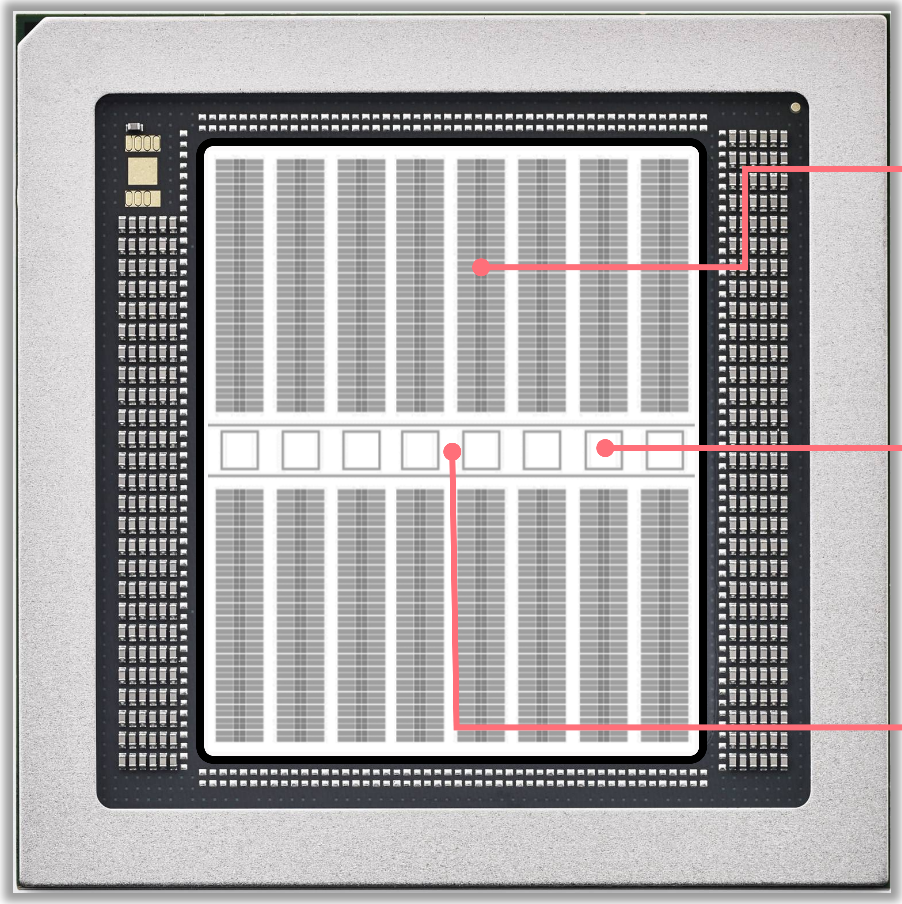
SOFTWARE

SUPPORT FOR COMMON MACHINE INTELLIGENCE DEVELOPMENT FRAMEWORKS AND DIRECT IPU PROGRAMMING THAT ENABLES DEVELOPERS TO SEAMLESSLY INTEGRATE IPUS INTO APPLICATIONS



GRAPHCORE

SCALING ON DEVICE



FIRST GENERATION IPU – COLOSSUS MK1

23.6 BILLION TRANSISTORS, 7296 FULLY INDEPENDENT WORKERS, 45 TB/S MEMORY BANDWIDTH
300MB IN-PROCESSOR MEMORY™, PCIe GEN4 INTERFACE, IPU-LINK™ INTERFACE

COMPUTE

- 1216 FULLY PROGRAMMABLE IPU COMPUTE TILES
- EACH WITH 256KB IN-PROCESSOR MEMORY™
- ACCELERATED FLOATING POINT HARDWARE ENGINES
- 6 INDEPENDENT HARDWARE WORKERS PER TILE

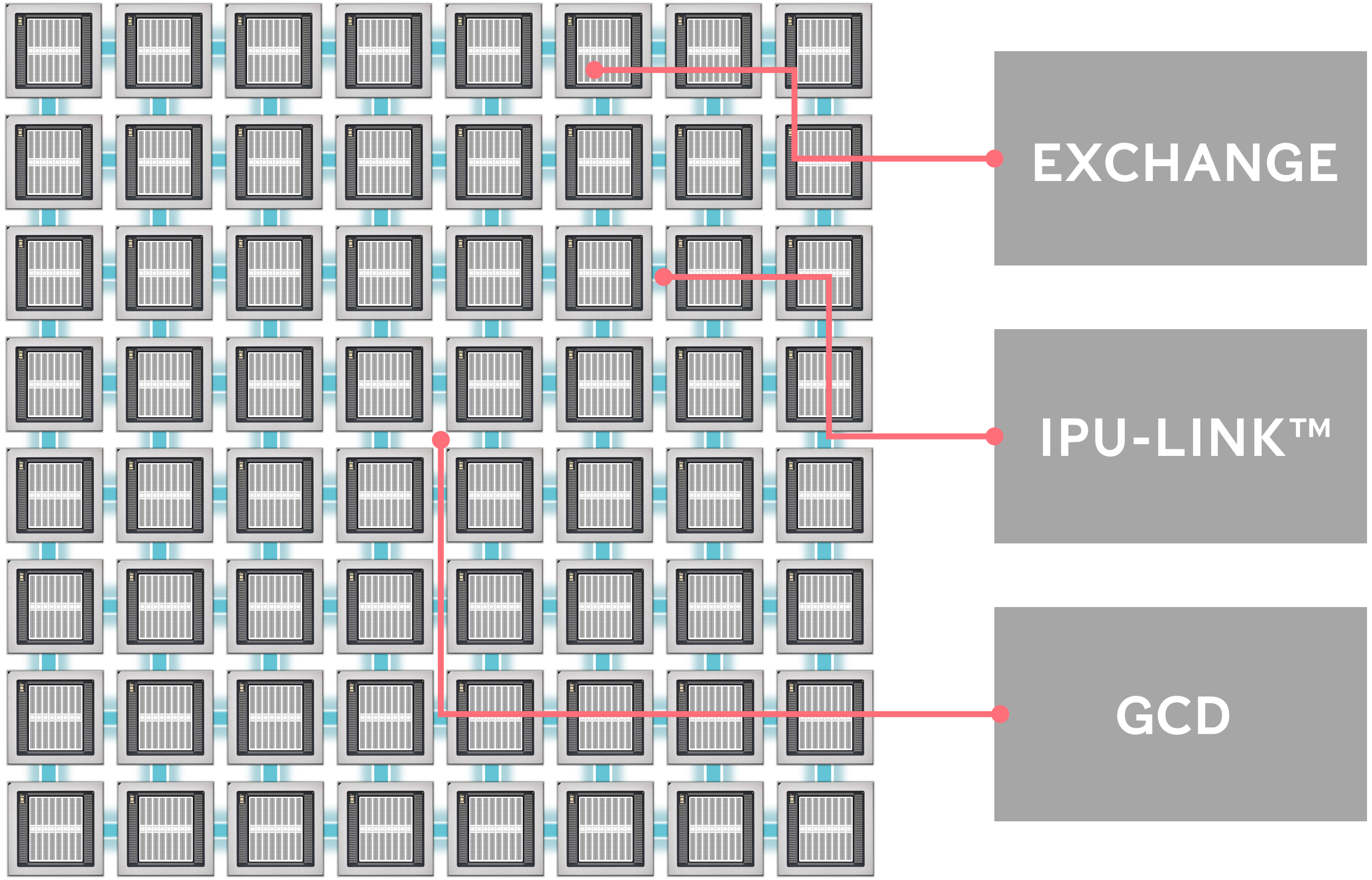
EXCHANGE

- 8 TB/S TILE TO TILE COMMUNICATION
- SUPPORTS ANY COMMUNICATION PATTERN
- FULLY TIMING DETERMINISTIC EXECUTION
- DRIVEN BY COMPILER SUPPORT IN SOFTWARE

SYNC

- HARDWARE ACCELERATED TILE SYNCHRONISATION
- LOW LATENCY MESSAGING OF WORK COMPLETION
- USED TO MOVE BETWEEN COMPUTE AND EXCHANGE
- EXTENDED INSTRUCTION SET FOR SYNCHRONISATION

SCALING ACROSS DEVICES



- THE IPU EXCHANGE MODEL EXTENDS OFF DEVICE
- ALLOWS TILE MESSAGING ACROSS IPU DEVICES
- SUPPORTS OPTIMIZED CROSS IPU SYNCHRONISATION
- DRIVEN BY COMPILER SUPPORT IN SOFTWARE

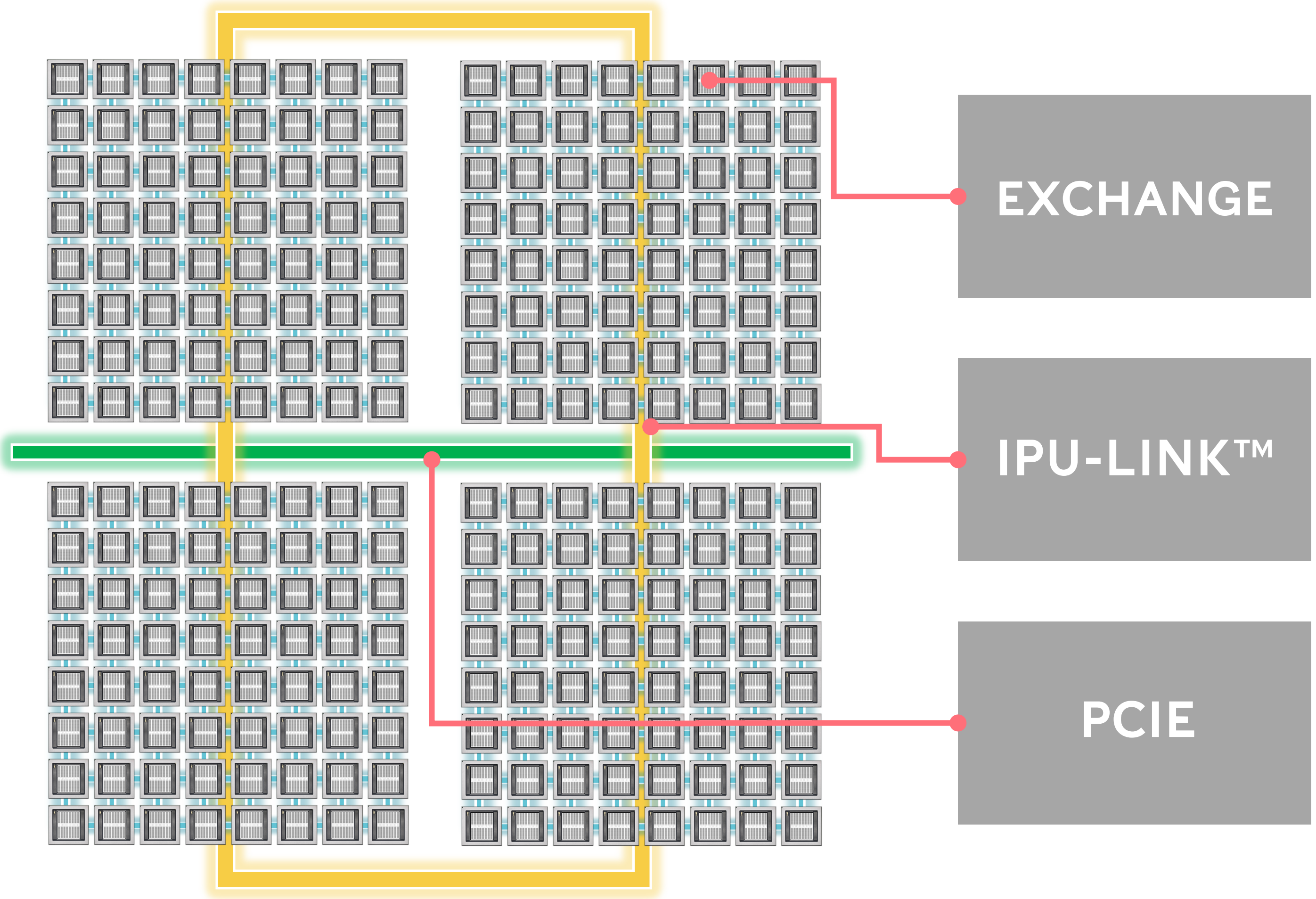
- PROVIDES 320 GB/S IPU TO IPU BANDWIDTH
- SUPPORTS COMMUNICATION BETWEEN IPUS
- LAYOUT FULLY SOFTWARE CONFIGURABLE
- SUPPORTS POINT TO POINT TILE MESSAGING

- GRAPH COMPILE DOMAIN FOR APPLICATIONS
- CREATES A SINGLE LARGE IPU SOFTWARE TARGET
- FULLY CONFIGURABLE PARTITIONING OF IPUS
- BOTH DATA PARALLEL AND MODEL PARALLEL

UP TO 64 IPU DEVICES USABLE AS A SINGLE LARGE IPU FROM APPLICATIONS

466944 FULLY INDEPENDENT WORKERS, 19.2GB IN-PROCESSOR MEMORY™, LEVERAGING OVER 1.5 TRILLION TRANSISTORS

SCALING ACROSS SYSTEMS



- IPU EXCHANGE SUPPORT ACROSS DOMAINS
- DRIVEN BY COMPILER SUPPORT IN SOFTWARE
- ENABLES APPLICATION COLLECTIVES SUPPORT
- ALLOWS SCALING UP TO 4096 IPU DEVICES

- IPU LINK™ CAN BE EXTENDED ACROSS DOMAINS
- SUPPORTS OPTIMIZED IPU LINK™ COLLECTIVES
- ALLOWS REPLICATION ACROSS SYSTEMS
- SUPPORTS A STANDARD IPU SOFTWARE MODEL

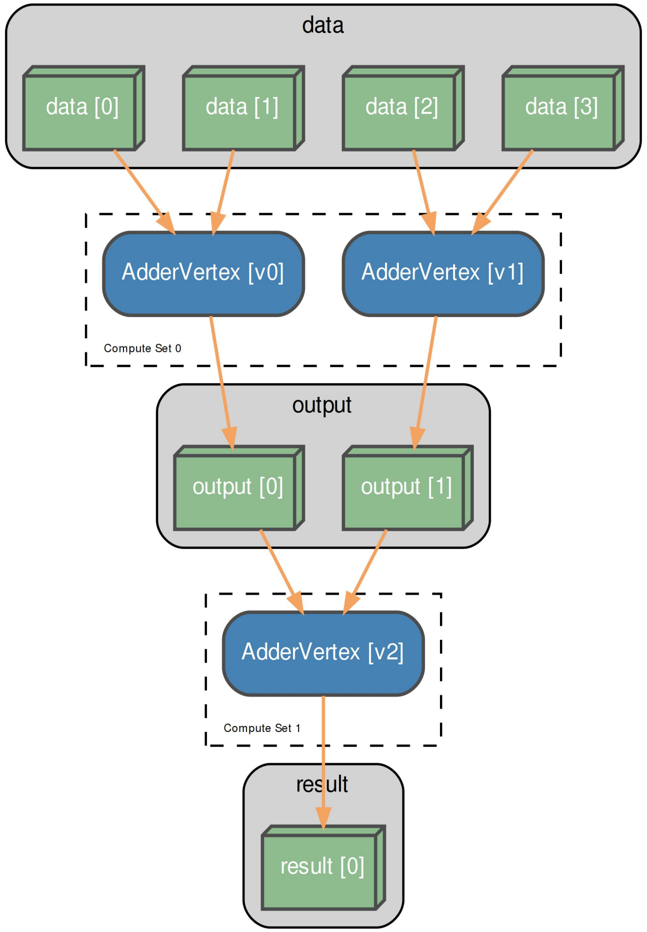
- IPUS CAN ACCESS MEMORY AND DEVICES OVER PCIE
- ALLOWS INTERFACING WITH HOST BASED SOFTWARE
- APPLICATIONS CAN BUILD ON HOST NETWORKING
- ALLOWS SCALING IN STANDARD SERVER PLATFORMS

256 IPU APPLICATION TARGET BUILT FROM INTERCONNECTED 64 IPU DOMAINS

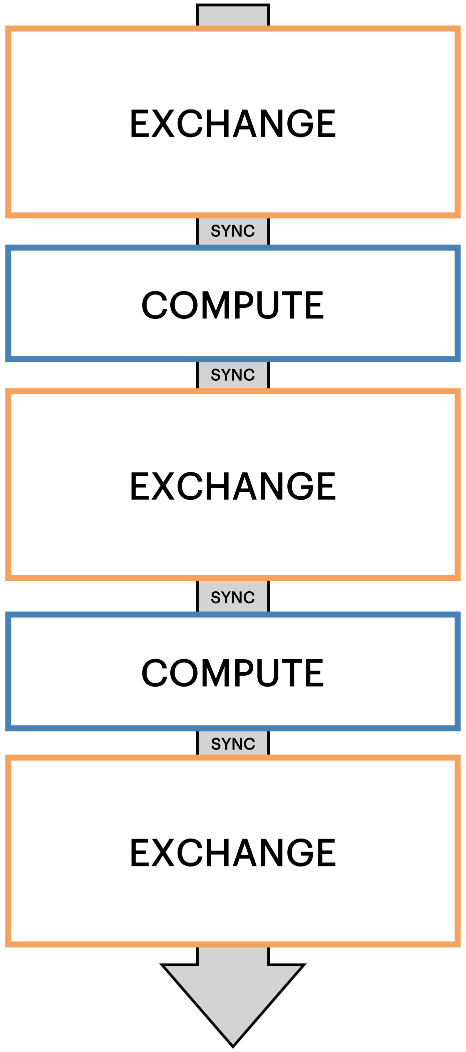
SCALE OUT SUPPORT UP TO A MAXIMUM OF 4096 IPUS WITH FIRST GENERATION COLOSSUS MK1

EXECUTION MODEL

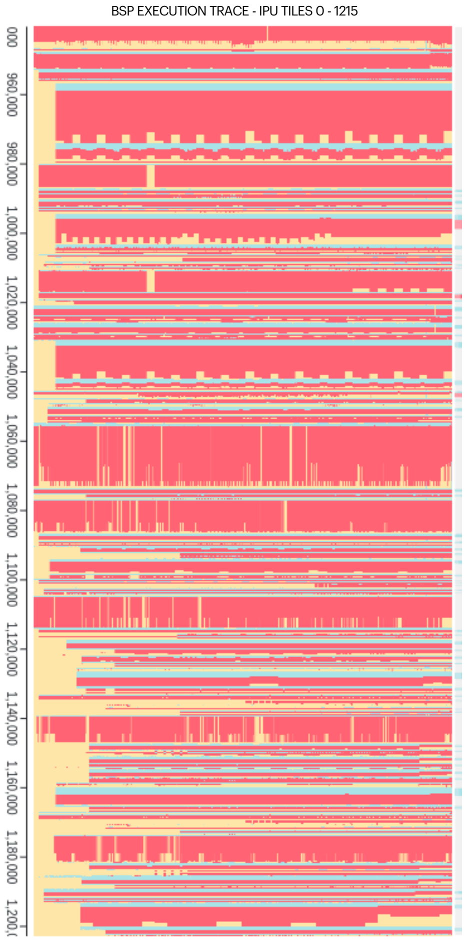
COMPUTATIONAL GRAPH



BSP SCHEDULE



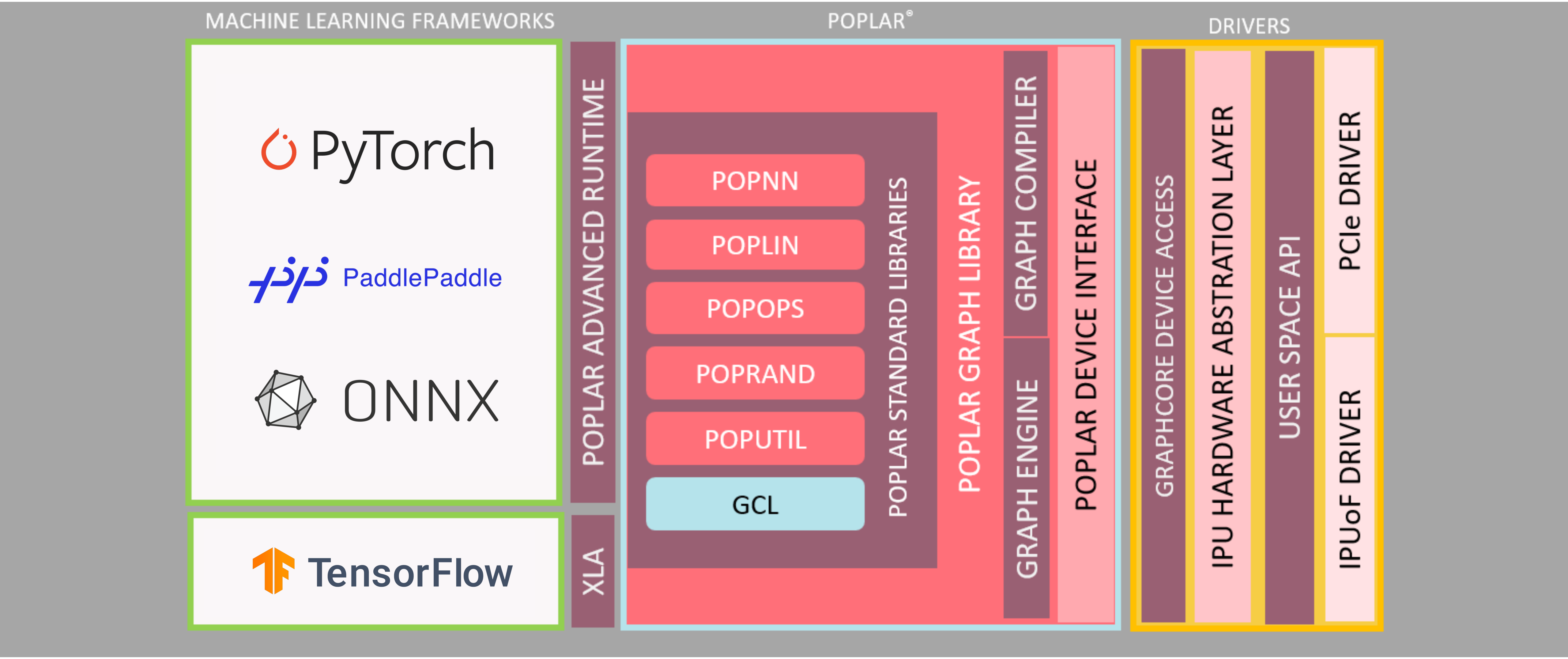
OPTIMIZED IPU EXECUTION



OUTPUT FROM POPVISION GRAPH ANALYSER

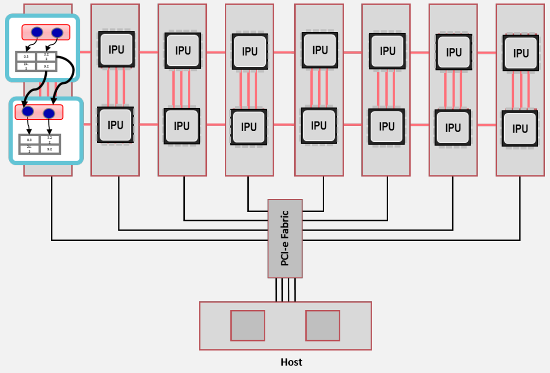
GRAPHCORE

POPLAR® SDK


MACHINE LEARNING FRAMEWORKS
PyTorch
PaddlePaddle
ONNX
TensorFlow
POPLAR ADVANCED RUNTIME
XLA
POPLAR®
POPNN
POPLIN
POPOPS
POPRAND
POPUTIL
GCL
POPLAR STANDARD LIBRARIES
POPLAR GRAPH LIBRARY
GRAPH COMPILER
GRAPH ENGINE
POPLAR DEVICE INTERFACE
DRIVERS
GRAPHCORE DEVICE ACCESS
IPU HARDWARE ABSTRACTION LAYER
USER SPACE API
PCIe DRIVER
IPUoF DRIVER

MULTI-IPU CONSTRUCTS

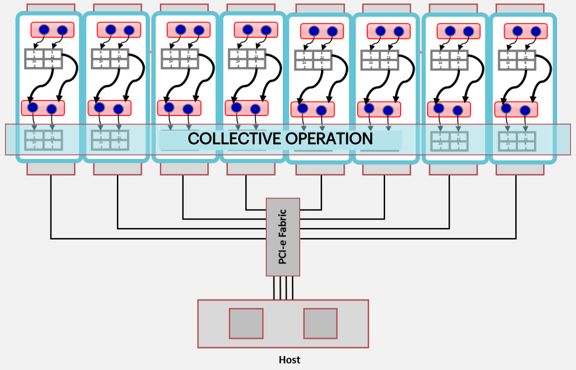
MODEL SHARDING



SUPPORT THE SPLITTING OF MODELS ACROSS MULTIPLE IPU DEVICES

ALLOW USER DRIVEN SOFTWARE CONTROL OF MODEL PARALLELISM

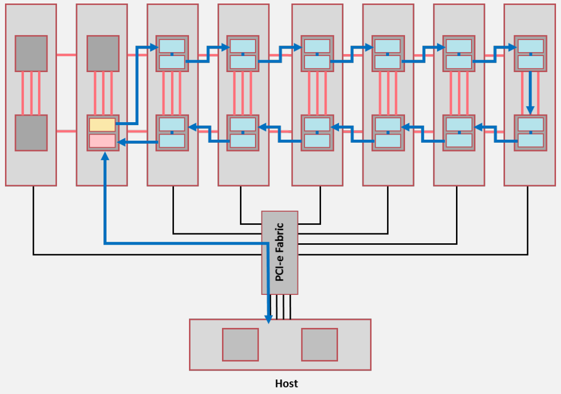
MODEL REPLICATION



SUPPORT THE REPLICATION OF MODELS ACROSS AN ENTIRE IPU SYSTEM

ENABLE DATA PARALLEL TRAINING AND AUTOMATIC REPLICATION OF MODELS

MODEL PIPELINING



SUPPORT THE PIPELINING OF MODELS ACROSS MULTIPLE IPU DEVICES

EXTRACT MAXIMUM PERFORMANCE FOR MODEL PARALLEL EXECUTION

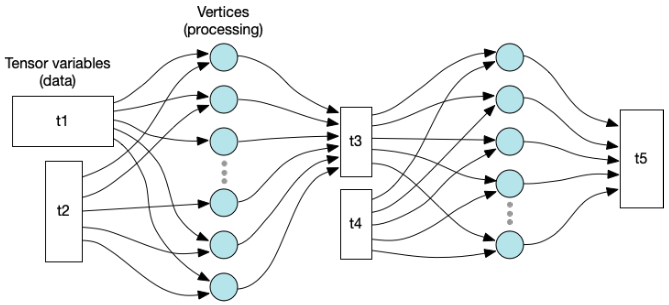
FULLY SUPPORTED IN PYTORCH, TENSORFLOW, POPART™ AND POPLAR®

GRAPHCORE

POPLAR®

GRAPH LIBRARY

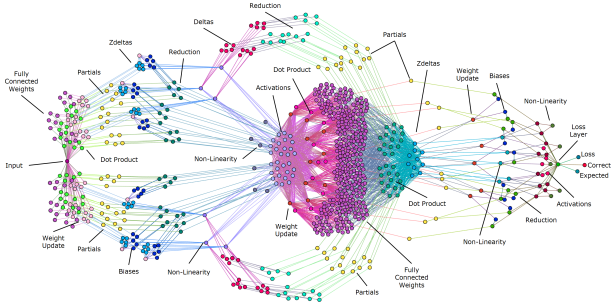
- SIMPLE C++ GRAPH BUILDING API
- IMPLEMENT ANY APPLICATION
- FULL CONTROL FLOW SUPPORT



USER DEFINED COMPUTATIONAL GRAPH

GRAPH COMPILER

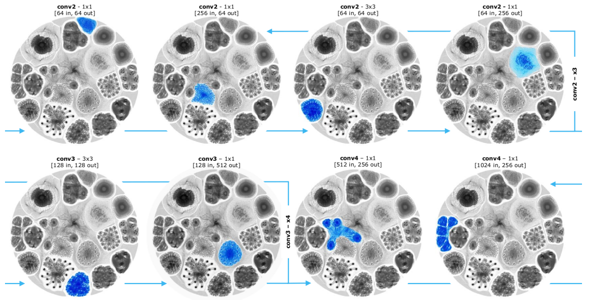
- OPTIMIZING IPU GRAPH COMPILER
- IMPLEMENTS IPU EXECUTION MODEL
- CODE GENERATION USING LLVM



GRAPH COMPILER INTERMEDIATE REPRESENTATION

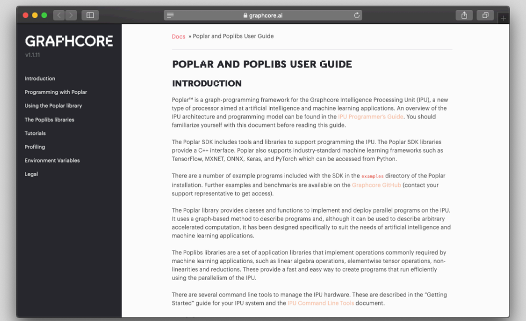
GRAPH ENGINE

- HIGH PERFORMANCE GRAPH RUNTIME
- INTERFACES TO HOST MEMORY SYSTEM
- HIGHLY OPTIMIZED IPU DATA TRANSFER

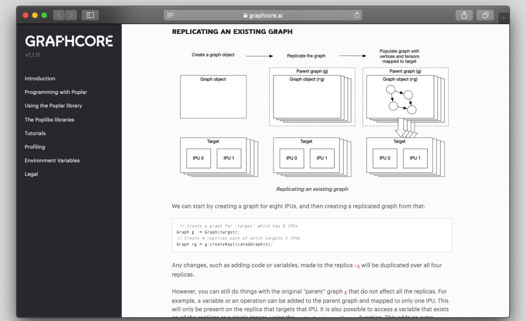


OPTIMIZED GRAPH EXECUTION WITH HOST SYSTEM

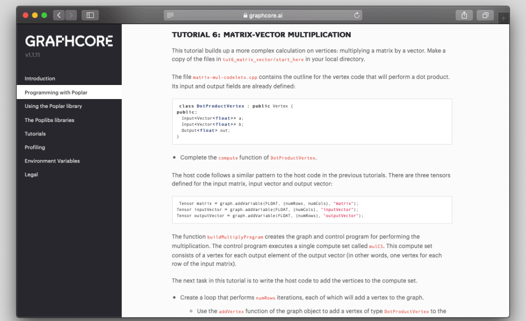
WWW.GRAPHCORE.AI/DEVELOPER



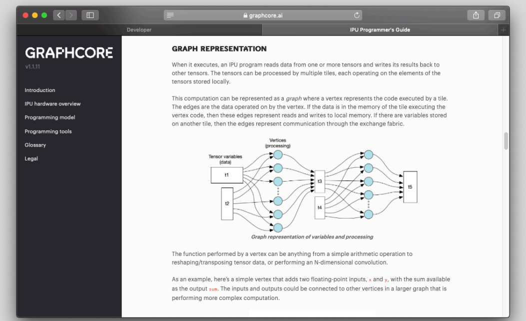
COMPREHENSIVE POPLAR USER GUIDE



USING MULTIPLE IPUS IN APPLICATIONS



TUTORIALS AND CODE EXAMPLES



DETAILED DESCRIPTIONS OF UNDERLYING CONCEPTS

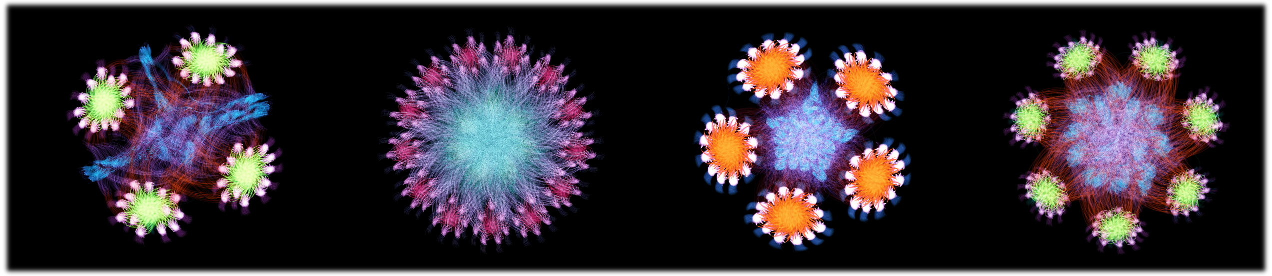
GRAPHCORE

POPLAR® LIBRARIES

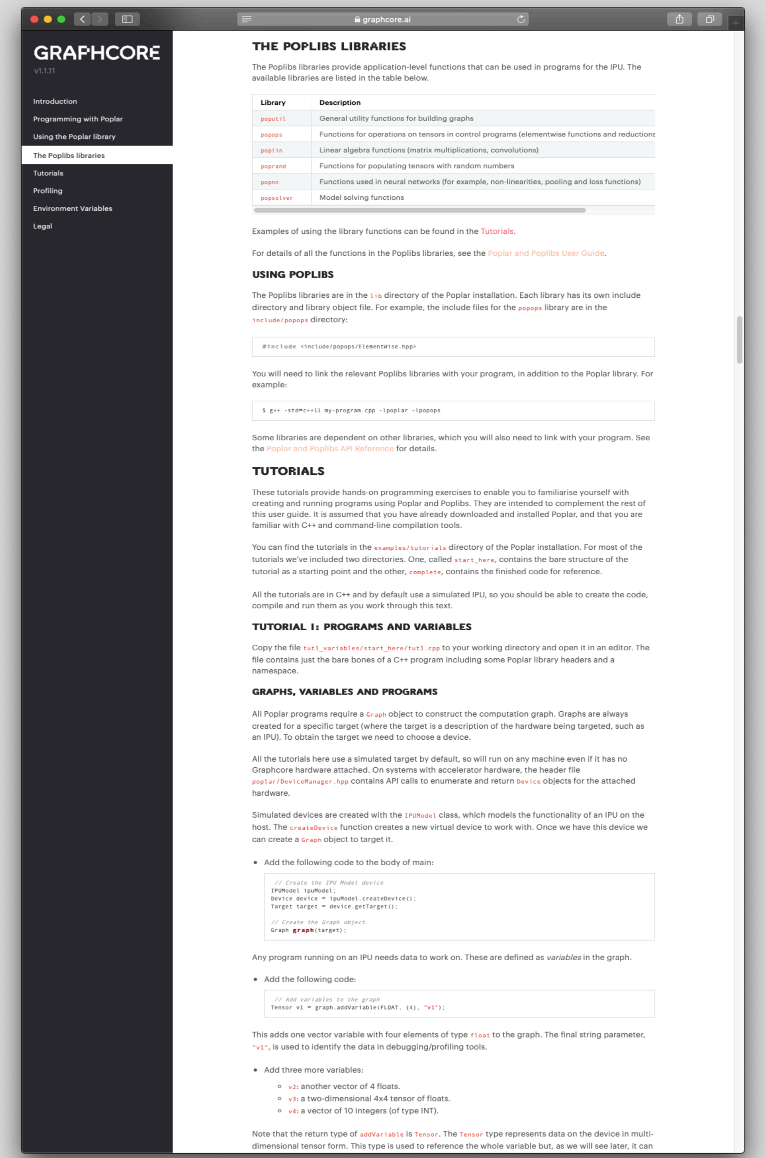
- OVER 50 OPTIMISED FUNCTIONS FOR COMMON ML MODELS
- MORE THAN 750 HIGH PERFORMANCE COMPUTE ELEMENTS

POPNN	FUNCTIONS USED IN NEURAL NETWORKS (NON-LINEARITIES, POOLING, LOSS FUNCTIONS)
POPLIN	OPTIMIZED LINEAR ALGEBRA FUNCTIONS (MATRIX MULTIPLICATION, CONVOLUTIONS)
POPOPS	FUNCTIONS FOR PERFORMING ELEMENTWISE OPERATIONS ON TENSOR DATA
POPRAND	HIGH PERFORMANCE FUNCTIONS FOR POPULATING TENSORS WITH RANDOM NUMBERS
POPUTIL	GENERAL UTILITY FUNCTIONS FOR BUILDING GRAPHS FOR IPU DEVICES
GCL	OPTIMIZED COLLECTIVES LIBRARY SUPPORTING MODEL AND DATA PARALLEL

OPTIMIZED WORK PLANNING OF FUNCTIONS ACROSS IPU DEVICES



POPLAR GRAPH COMPILER INTERMEDIATE REPRESENTATION FOR MATRIX MULTIPLICATION OPERATIONS, CUSTOM GRAPH LAYOUT SPECIALISED BASED ON SHAPES OF INPUTS AND OUTPUTS



graphcore.ai

GRAPHCORE
v1.1.11

Introduction
Programming with Poplar
Using the Poplar library
The Poplibs libraries
Tutorials
Profiling
Environment Variables
Legal

THE POPLIBS LIBRARIES

The Poplibs libraries provide application-level functions that can be used in programs for the IPU. The available libraries are listed in the table below.

Library	Description
poputil	General utility functions for building graphs
popops	Functions for operations on tensors in control programs (elementwise functions and reductions
poplin	Linear algebra functions (matrix multiplications, convolutions)
poprand	Functions for populating tensors with random numbers
popnn	Functions used in neural networks (for example, non-linearities, pooling and loss functions)
popsolver	Model solving functions

Examples of using the library functions can be found in the Tutorials.

For details of all the functions in the Poplibs libraries, see the Poplar and Poplibs User Guide.

USING POPLIBS

The Poplibs libraries are in the `lib` directory of the Poplar installation. Each library has its own include directory and library object file. For example, the include files for the `popops` library are in the `include/popops` directory:

```
#include <include/popops/ElementWise.hpp>
```

You will need to link the relevant Poplibs libraries with your program, in addition to the Poplar library. For example:

```
$ g++ -std=c++11 my-program.cpp -lpoplar -lpopops
```

Some libraries are dependent on other libraries, which you will also need to link with your program. See the Poplar and Poplibs API Reference for details.

TUTORIALS

These tutorials provide hands-on programming exercises to enable you to familiarise yourself with creating and running programs using Poplar and Poplibs. They are intended to complement the rest of this user guide. It is assumed that you have already downloaded and installed Poplar, and that you are familiar with C++ and command-line compilation tools.

You can find the tutorials in the `examples/tutorials` directory of the Poplar installation. For most of the tutorials we've included two directories. One, called `start_here`, contains the bare structure of the tutorial as a starting point and the other, `complete`, contains the finished code for reference.

All the tutorials are in C++ and by default use a simulated IPU, so you should be able to create the code, compile and run them as you work through this text.

TUTORIAL 1: PROGRAMS AND VARIABLES

Copy the file `tut1_variables/start_here/tut1.cpp` to your working directory and open it in an editor. The file contains just the bare bones of a C++ program including some Poplar library headers and a namespace.

GRAPHS, VARIABLES AND PROGRAMS

All Poplar programs require a `Graph` object to construct the computation graph. Graphs are always created for a specific target (where the target is a description of the hardware being targeted, such as an IPU). To obtain the target we need to choose a device.

All the tutorials here use a simulated target by default, so will run on any machine even if it has no Graphcore hardware attached. On systems with accelerator hardware, the header file `poplar/DeviceManager.hpp` contains API calls to enumerate and return `Device` objects for the attached hardware.

Simulated devices are created with the `IPUModel` class, which models the functionality of an IPU on the host. The `createDevice` function creates a new virtual device to work with. Once we have this device we can create a `Graph` object to target it.

- Add the following code to the body of main:

```
// Create the IPU Model device
IPUModel ipuModel;
Device device = ipuModel.createDevice();
Target target = device.getTarget();

// Create the Graph object
Graph graph(target);
```

Any program running on an IPU needs data to work on. These are defined as *variables* in the graph.

- Add the following code:

```
// Add variables to the graph
Tensor v1 = graph.addVariable(FLOAT, {4}, "v1");
```

This adds one vector variable with four elements of type `float` to the graph. The final string parameter, `"v1"`, is used to identify the data in debugging/profiling tools.

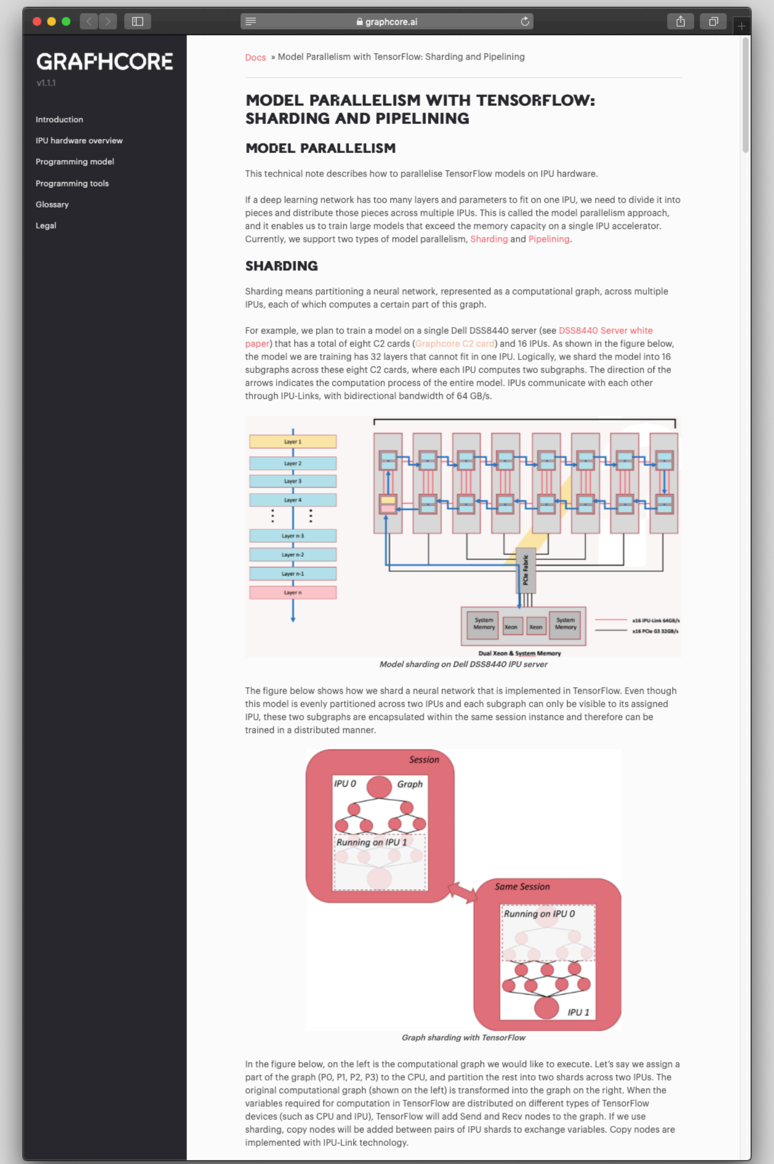
- Add three more variables:
  - `v2`: another vector of 4 floats.
  - `v3`: a two-dimensional 4x4 tensor of floats.
  - `v4`: a vector of 10 integers (of type INT).

Note that the return type of `addVariable` is `Tensor`. The `Tensor` type represents data on the device in multi-dimensional tensor form. This type is used to reference the whole variable but, as we will see later, it can

WWW.GRAPHCORE.AI/DEVELOPER

GRAPHCORE

MACHINE LEARNING FRAMEWORKS


graphcore.ai

GRAPHCORE
v1.1.1

Introduction
IPU hardware overview
Programming model
Programming tools
Glossary
Legal

Docs » Model Parallelism with TensorFlow: Sharding and Pipelining

MODEL PARALLELISM WITH TENSORFLOW: SHARDING AND PIPELINING

MODEL PARALLELISM

This technical note describes how to parallelise TensorFlow models on IPU hardware.

If a deep learning network has too many layers and parameters to fit on one IPU, we need to divide it into pieces and distribute those pieces across multiple IPUs. This is called the model parallelism approach, and it enables us to train large models that exceed the memory capacity on a single IPU accelerator. Currently, we support two types of model parallelism, Sharding and Pipelining.

SHARDING

Sharding means partitioning a neural network, represented as a computational graph, across multiple IPUs, each of which computes a certain part of this graph.

For example, we plan to train a model on a single Dell DSS8440 server (see DSS8440 Server white paper) that has a total of eight C2 cards (Graphcore C2 card) and 16 IPUs. As shown in the figure below, the model we are training has 32 layers that cannot fit in one IPU. Logically, we shard the model into 16 subgraphs across these eight C2 cards, where each IPU computes two subgraphs. The direction of the arrows indicates the computation process of the entire model. IPUs communicate with each other through IPU-Links, with bidirectional bandwidth of 64 GB/s.

Model sharding on Dell DSS8440 IPU server

The figure below shows how we shard a neural network that is implemented in TensorFlow. Even though this model is evenly partitioned across two IPUs and each subgraph can only be visible to its assigned IPU, these two subgraphs are encapsulated within the same session instance and therefore can be trained in a distributed manner.

Graph sharding with TensorFlow

In the figure below, on the left is the computational graph we would like to execute. Let's say we assign a part of the graph (P0, P1, P2, P3) to the CPU, and partition the rest into two shards across two IPUs. The original computational graph (shown on the left) is transformed into the graph on the right. When the variables required for computation in TensorFlow are distributed on different types of TensorFlow devices (such as CPU and IPU), TensorFlow will add Send and Recv nodes to the graph. If we use sharding, copy nodes will be added between pairs of IPU shards to exchange variables. Copy nodes are implemented with IPU-Link technology.

WWW.GRAPHCORE.AI/DEVELOPER

TensorFlow

- TENSORFLOW SUPPORT FOR IPU AS FAMILIAR TARGET FOR MODELS
- FULL PERFORMANT INTEGRATION WITH TENSORFLOW XLA BACKEND
- SUPPORT FOR VERSION 1 & 2 WITH EXAMPLES AND DOCUMENTATION

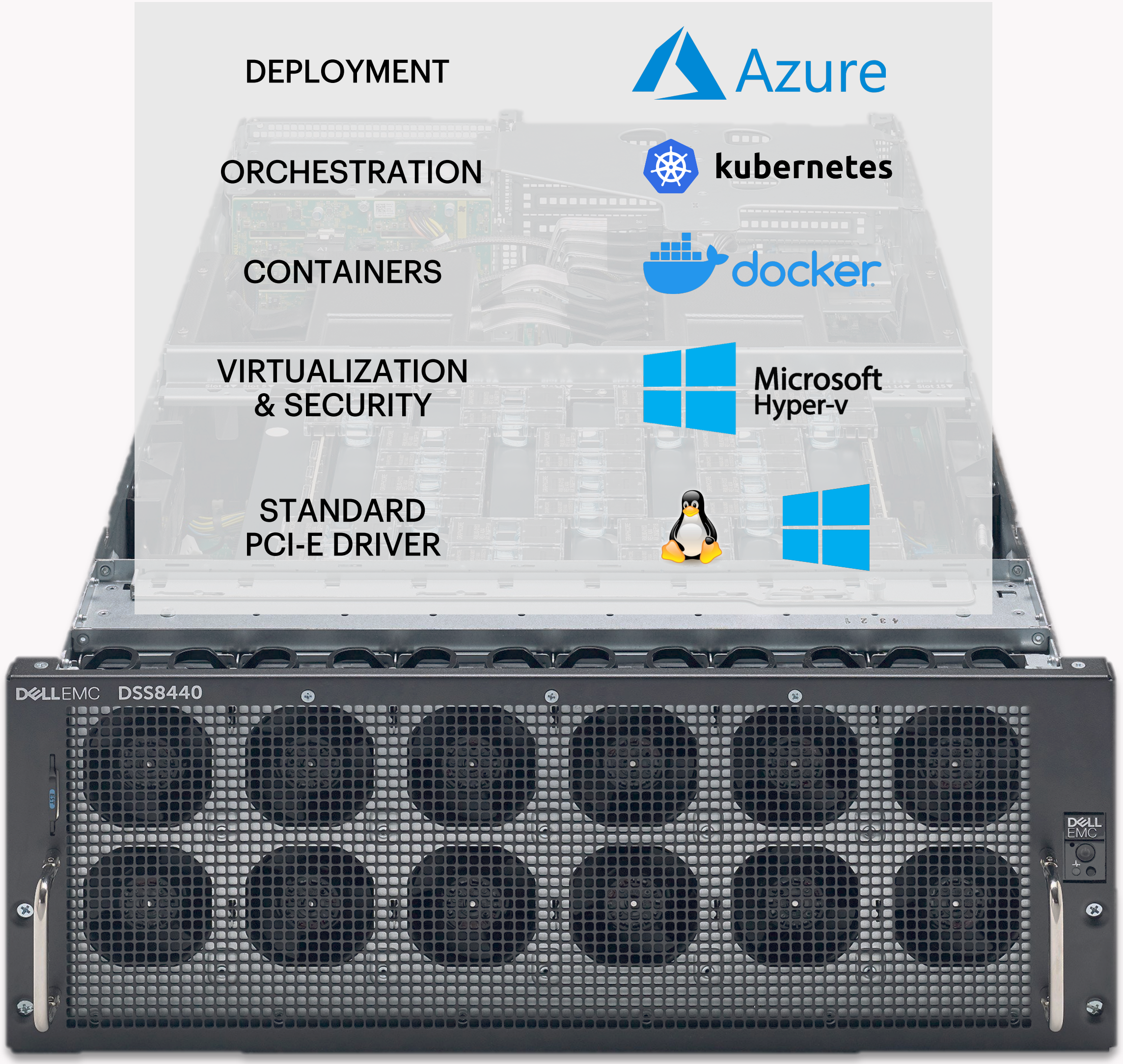
PyTorch

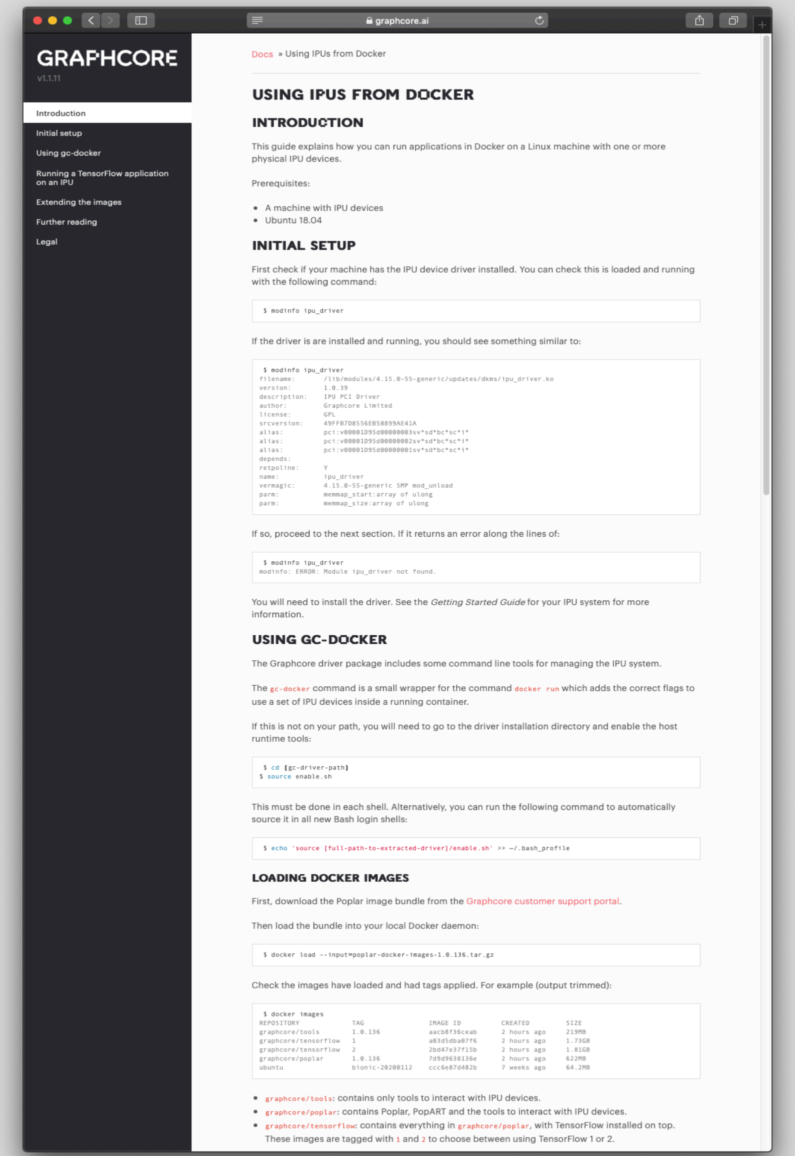
- PYTORCH SUPPORT FOR TARGETING IPU WITH SIMPLE EXTENSIONS
- TAKE NATIVE PYTORCH MODELS, DEPLOY AND TRAIN ON IPU DEVICES
- SUPPORT FOR MULTI-IPU PRIMATIVES FROM PYTORCH MODELS

POPART™

- THE POPLAR ADVANCED RUNTIME FOR INFERENCE AND TRAINING
- SUPPORTS ONNX MODEL INPUT AND PYTHON / C++ MODEL BUILDING
- AN OPTIMIZED LIGHTWEIGHT APPLICATION RUNTIME FOR DEPLOYMENT

SOFTWARE ECOSYSTEM





Docs » Using IPUs from Docker

USING IPUS FROM DOCKER

INTRODUCTION

This guide explains how you can run applications in Docker on a Linux machine with one or more physical IPU devices.

Prerequisites:

- A machine with IPU devices
- Ubuntu 18.04

INITIAL SETUP

First check if your machine has the IPU device driver installed. You can check this is loaded and running with the following command:

```
$ modinfo ipu_driver
```

If the driver is are installed and running, you should see something similar to:

```
$ modinfo ipu_driver
filename:       /lib/modules/4.15.0-55-generic/updates/dkms/ipu_driver.ko
version:        1.0.39
description:    IPU PCI Driver
author:         Graphcore Limited
license:        GPL
srcversion:     49FFB7D8556EB58899AE41A
alias:          pci:v00001D95d00000003sv*sd*bc*sc*i*
alias:          pci:v00001D95d00000002sv*sd*bc*sc*i*
alias:          pci:v00001D95d00000001sv*sd*bc*sc*i*
depends:
retpoline:      Y
name:           ipu_driver
vermagic:       4.15.0-55-generic SMP mod_unload
parm:           memmap_start:array of ulong
parm:           memmap_size:array of ulong
```

If so, proceed to the next section. If it returns an error along the lines of:

```
$ modinfo ipu_driver
modinfo: ERROR: Module ipu_driver not found.
```

You will need to install the driver. See the *Getting Started Guide* for your IPU system for more information.

USING GC-DOCKER

The Graphcore driver package includes some command line tools for managing the IPU system.

The `gc-docker` command is a small wrapper for the command `docker run` which adds the correct flags to use a set of IPU devices inside a running container.

If this is not on your path, you will need to go to the driver installation directory and enable the host runtime tools:

```
$ cd [gc-driver-path]
$ source enable.sh
```

This must be done in each shell. Alternatively, you can run the following command to automatically source it in all new Bash login shells:

```
$ echo 'source [full-path-to-extracted-driver]/enable.sh' >> ~/.bash_profile
```

LOADING DOCKER IMAGES

First, download the Poplar image bundle from the Graphcore customer support portal.

Then load the bundle into your local Docker daemon:

```
$ docker load --input=poplar-docker-images-1.0.136.tar.gz
```

Check the images have loaded and had tags applied. For example (output trimmed):

```
$ docker images
REPOSITORY             TAG               IMAGE ID        CREATED        SIZE
graphcore/tools        1.0.136           aacb8f36ceab    2 hours ago    219MB
graphcore/tensorflow   1                 a03d5dba07f6    2 hours ago    1.73GB
graphcore/tensorflow   2                 2bd47e37f15b    2 hours ago    1.81GB
graphcore/poplar       1.0.136           7d9d9638136e    2 hours ago    622MB
ubuntu                 bionic-20200112   ccc6e87d482b    7 weeks ago    64.2MB
```

- `graphcore/tools`: contains only tools to interact with IPU devices.
- `graphcore/poplar`: contains Poplar, PopART and the tools to interact with IPU devices.
- `graphcore/tensorflow`: contains everything in `graphcore/poplar`, with TensorFlow installed on top. These images are tagged with `1` and `2` to choose between using TensorFlow 1 or 2.

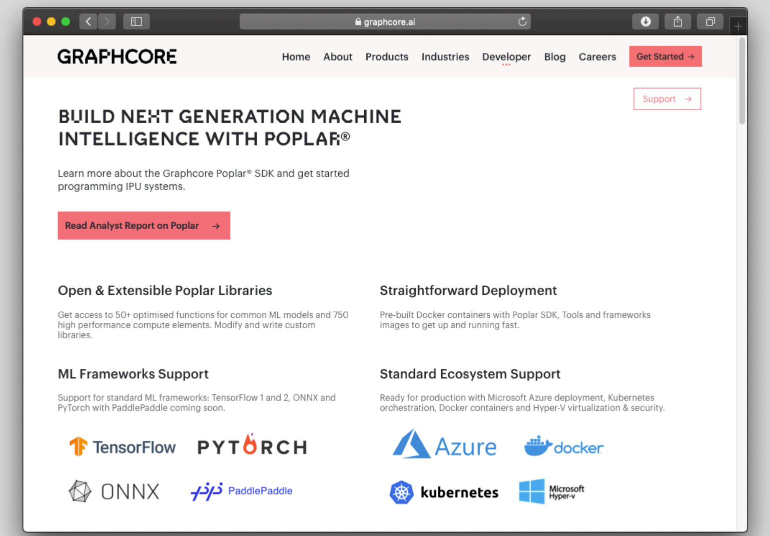
WWW.GRAPHCORE.AI/DEVELOPER

GRAPHCORE

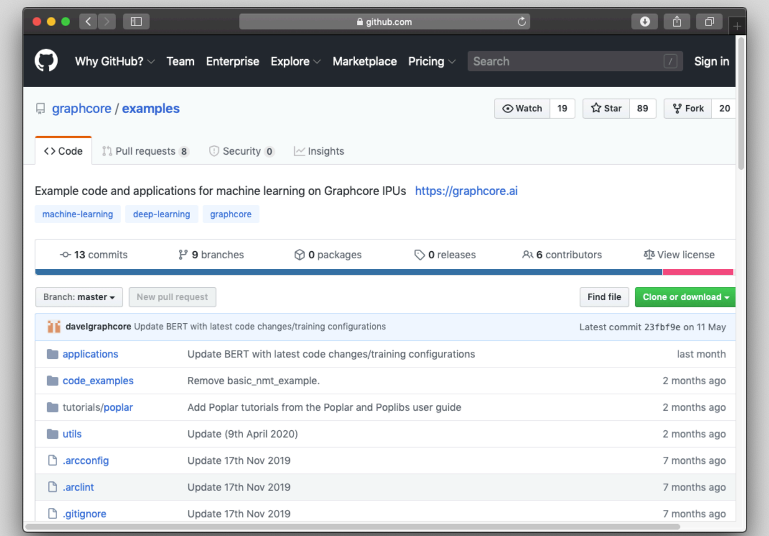
DEVELOPERS

CLOSE TO 1000 USERS SIGNED UP TO WORK WITH GRAPHCORE TECHNOLOGY

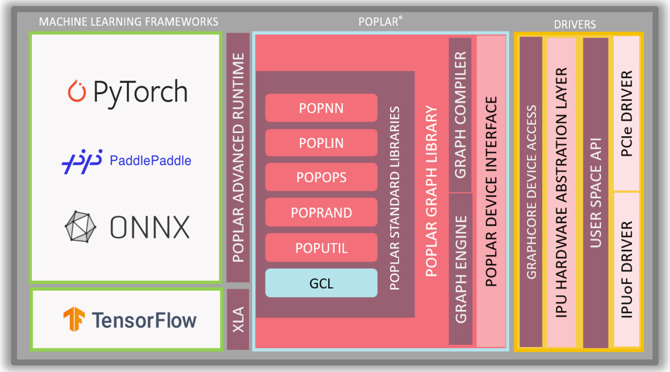
WWW.GRAPHCORE.AI/DEVELOPER



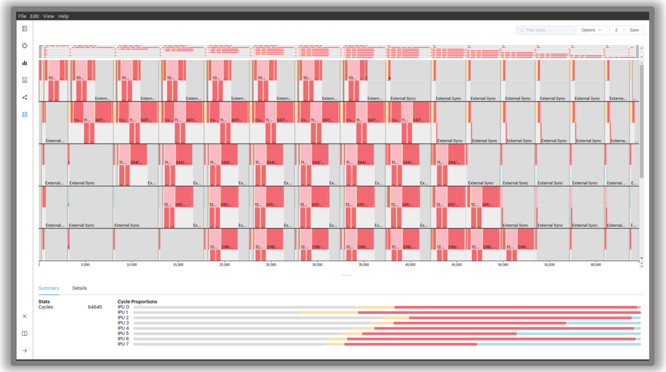
WWW.GITHUB.COM/GRAPHCORE



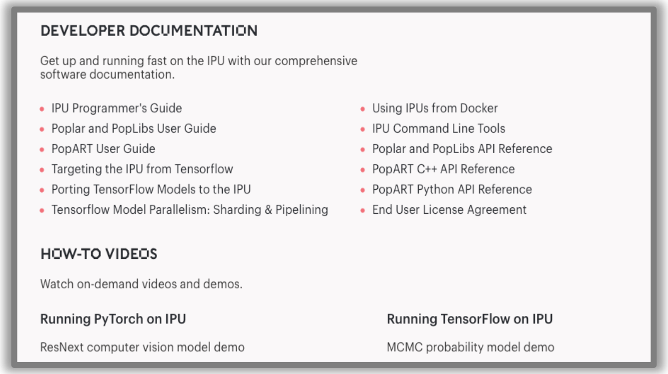
POPLAR SDK



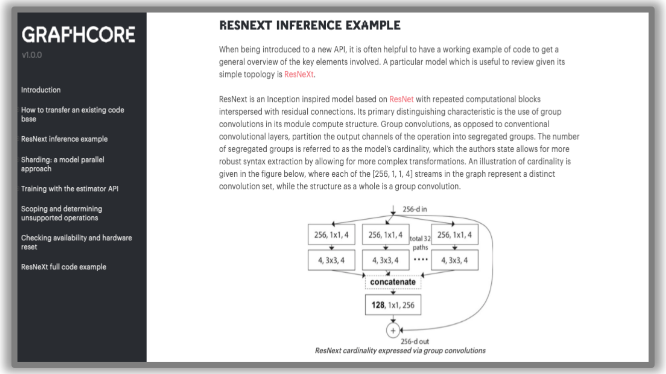
POPVISION TOOLS



DOCUMENTATION

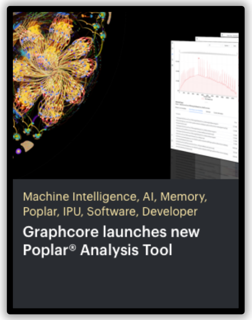


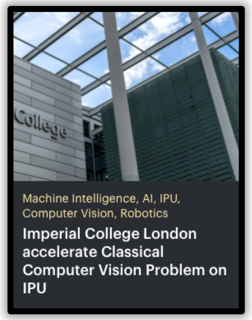
APPLICATION EXAMPLES

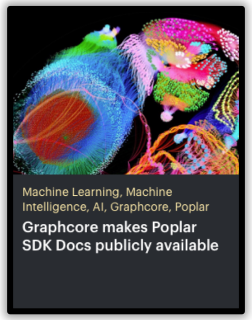


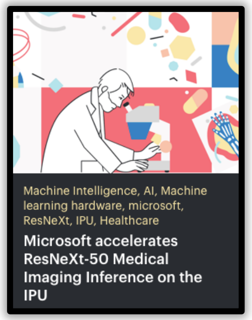
GRAPHCORE

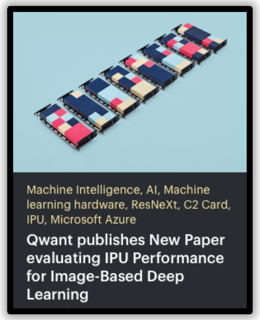
SUMMARY



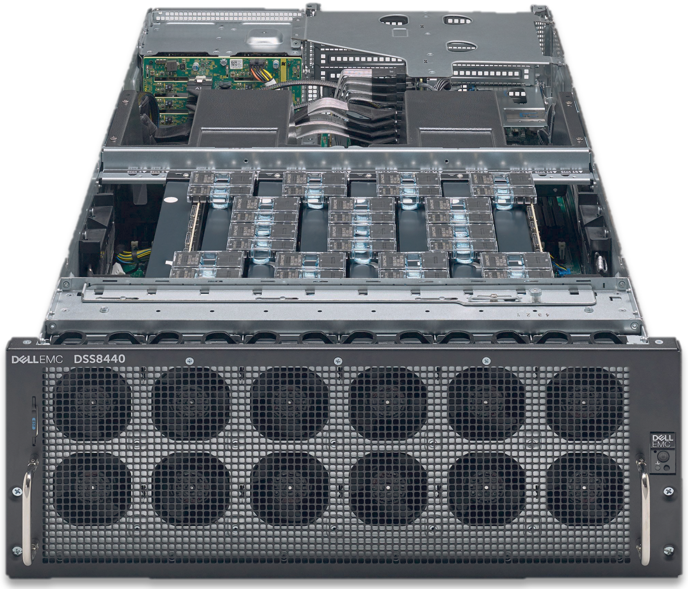








- RAPID PROGRESS WITH PLATFORM CAPABILITY
- CLOSE TO 1000 USERS AND GROWING FAST
- POPLAR SDK CONTINUES TO EVOLVE AT PACE
- DEVELOPER PORTAL MAKES USING IPU SIMPLE
- TRY IPU TODAY IN MICROSOFT AZURE CLOUD



BERT BASE POPART MODEL – POPLAR GRAPH COMPILER IR VISUALISATION

GRAPHCORE

WWW.GRAPHCORE.AI