GRAPHCORE WORKSHOP FOR TAMU

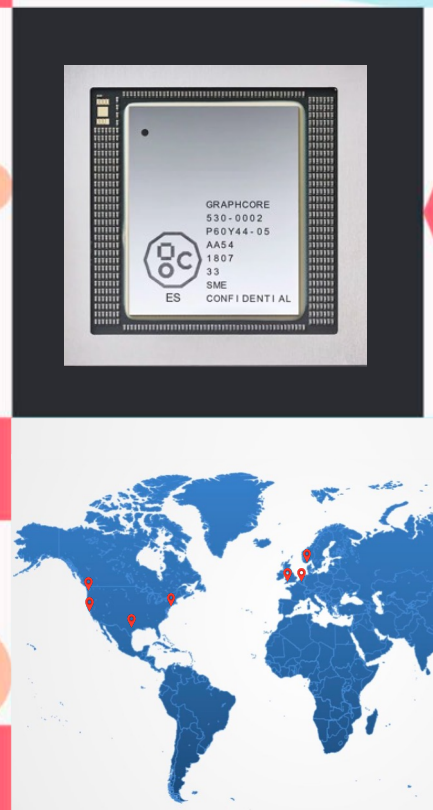
October 3, 2023

Alexander Tsyplikhin

AGENDA

- Introduction to Graphcore, IPU, and Poplar
  - Hands-on: access the POD, enable the SDK, run an example
- TensorFlow2/Keras
  - Hands-on: Port a Keras script, leverage loop on device, replicate and run data-parallel, pipeline
- PyTorch
  - Hands-on: PopTorch example, DataLoader, options to optimize performance

GRAPHCORE ENABLING MACHINE INTELLIGENCE

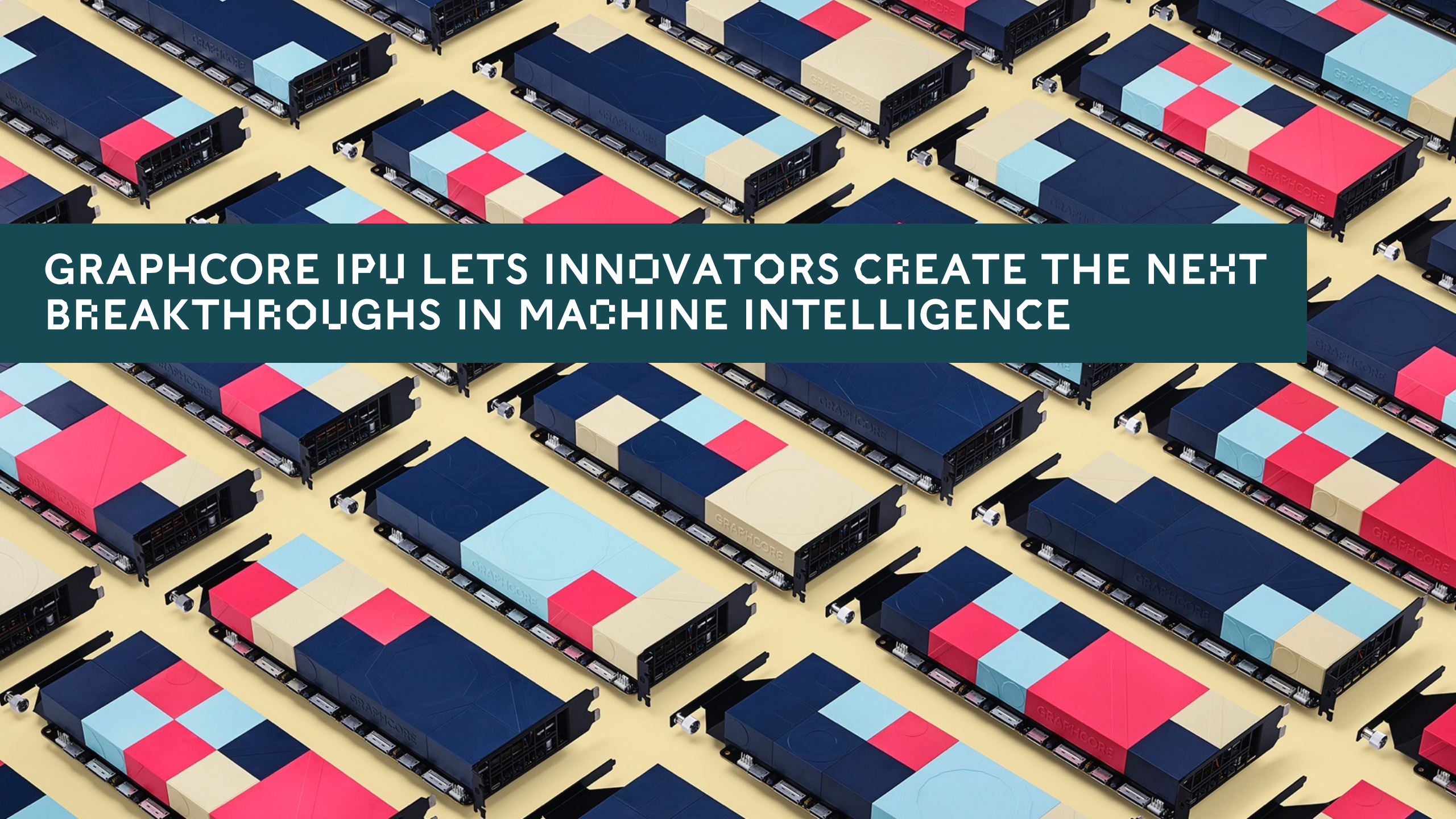


- Founded in 2016
- Technology: Intelligence Processor Unit (IPU)
- Team: ~500
- Offices: UK, US, China, Poland
- Raised >$710M

SEQUOIA ATOMICO SOFINA Microsoft

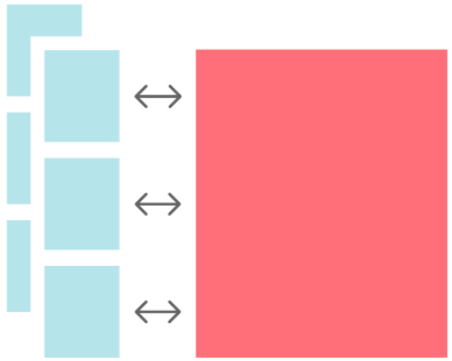
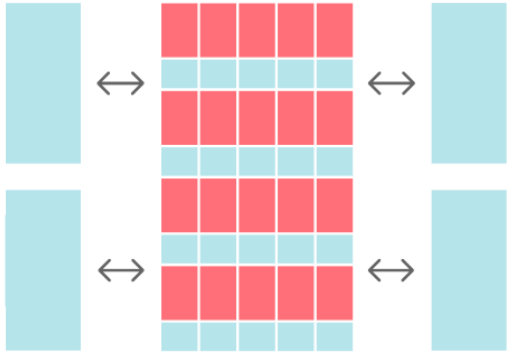
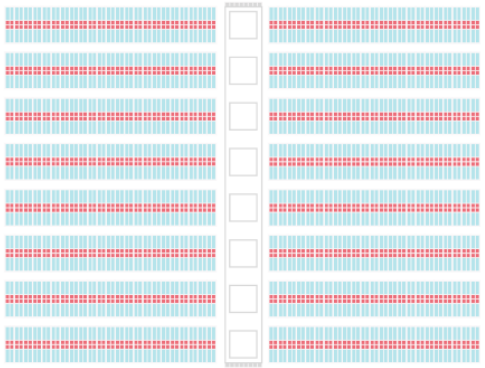
BMW i VENTURES DELL BOSCH SAMSUNG Merian GLOBAL INVESTORS

Amadeus Capital Partners pitango VENTURE CAPITAL draperesprit Foundation CAPITAL


GRAPHCORE IPU LETS INNOVATORS CREATE THE NEXT
BREAKTHROUGHS IN MACHINE INTELLIGENCE

IPU – ARCHITECTURED FOR AI

Massive parallelism with ultrafast memory access

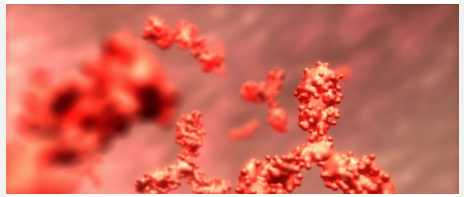
	CPU	GPU	IPU
Parallelism	Designed for scalar processes	SIMD/SIMT architecture. Designed for large blocks of dense contiguous data	Massively parallel MIMD. Designed for fine-grained, high-performance computing
Processors / Memory			
Memory Access	Off-chip memory	Model and data spread across off-chip and small on-chip cache, and shared memory	Model and data tightly coupled, and large locally distributed SRAM

PROVEN IPU ADVANTAGE

SELECT CASE STUDIES ACROSS MANY INDUSTRIES & FIELDS

LabGenius

HEALTHCARE



CASE STUDY : NLP

Man

FINANCE – OPTION PRICING

CASE STUDY : SIM

pienso

AI SaaS – TEXT ANALYTICS

CASE STUDY : NLP

Argonne NATIONAL LABORATORY

RESEARCH / BIG LABS

CASE STUDY

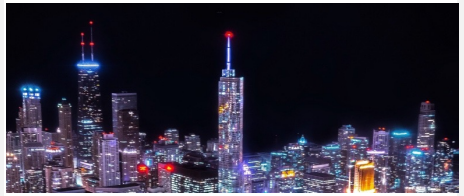
Pacific Northwest NATIONAL LABORATORY

COMPUTATIONAL CHEMISTRY

CASE STUDY : GNN

SENSORO

SMART CITY



CASE STUDY : CV

TRACTABLE

FINANCE - INSURANCE

CASE STUDY : CV

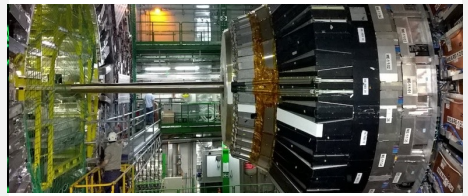
ECMWF

WEATHER FORECASTING

CASE STUDY : SIM

SLAC NATIONAL ACCELERATOR LABORATORY

HIGH ENERGY PHYSICS



CASE STUDY

UNIVERSITY OF OXFORD

DYNAMIC GRAPHS

CASE STUDY : GNN

IPU COMPUTATIONAL ADVANTAGES

Heterogeneous gather/scatter operations. E.g. GNNs

Group and depthwise convolutions. E.g. ResNeXt, EfficientNet

Vector operations with low arithmetic intensity. E.g. Sparse matmuls

Dense as well as Sparse Matrix Multiplication. E.g. Transformers

Hardware accelerated Random Number Generation. E.g. Random Projections

Hard to vectorize workloads. E.g. DFT in Computational Chemistry

References:
https://www.graphcore.ai/performance-results
https://www.graphcore.ai/posts/how-we-made-efficientnet-more-efficient
https://www.graphcore.ai/posts/delving-deep-into-modern-computer-vision-models
https://www.graphcore.ai/posts/training-neural-networks-in-low-dimensional-random-bases
https://www.graphcore.ai/posts/man-group-unlocks-massively-parallel-option-pricing-with-graphcore-ipu

WORKLOADS THAT CAN'T EASILY BE VECTORIZED

Workloads with while loops that continue until convergence is achieved e.g. ray tracing

Workloads where different compute paths are required depending on the inputs e.g. DFT or CRR model

Tree-based models with unbalanced trees

PySCF on IPU

Installation guide | Example DFT Computations | Generating data | Training SchNet | QM1B dataset

Run on Gradient | pytest notebooks passing

Port of PySCF to Graphcore IPU.

Limitations

- Restricted Kohn Sham DFT (based on RKS, KohnShamDFT and hf.RHF).
- Number of atomic orbitals less than 70 `mol.nao_nr() <= 70`.
- Larger numerical errors due to `np.float32` instead of `np.float64`.
- Limited support for `jax.grad(.)`

Installation

PySCF on IPU requires Python 3.8, JAX IPU experimental, TessellateIPU library and Graphcore Poplar SDK 3.2.

To run this package on a standard CPU machine (laptop or server), install the base Python requirements:

```
pip install -r requirements.txt
```

On IPU machines, please additionally use the IPU requirements file:

```
pip install -U pip
pip install -r requirements_ipu.txt
```

This will configure Graphcore research experimental JAX support in your python environment.

We recommend upgrading `pip` to the latest stable release when using the IPU requirements. This may be an optional step depending on the overall configuration of your python environment.

And finally, make our sub-packages available:

```
pip install -e .
```

Kernel sessions

nanoDFT-demo.ipynb
notebooks/nanoDFT-de...

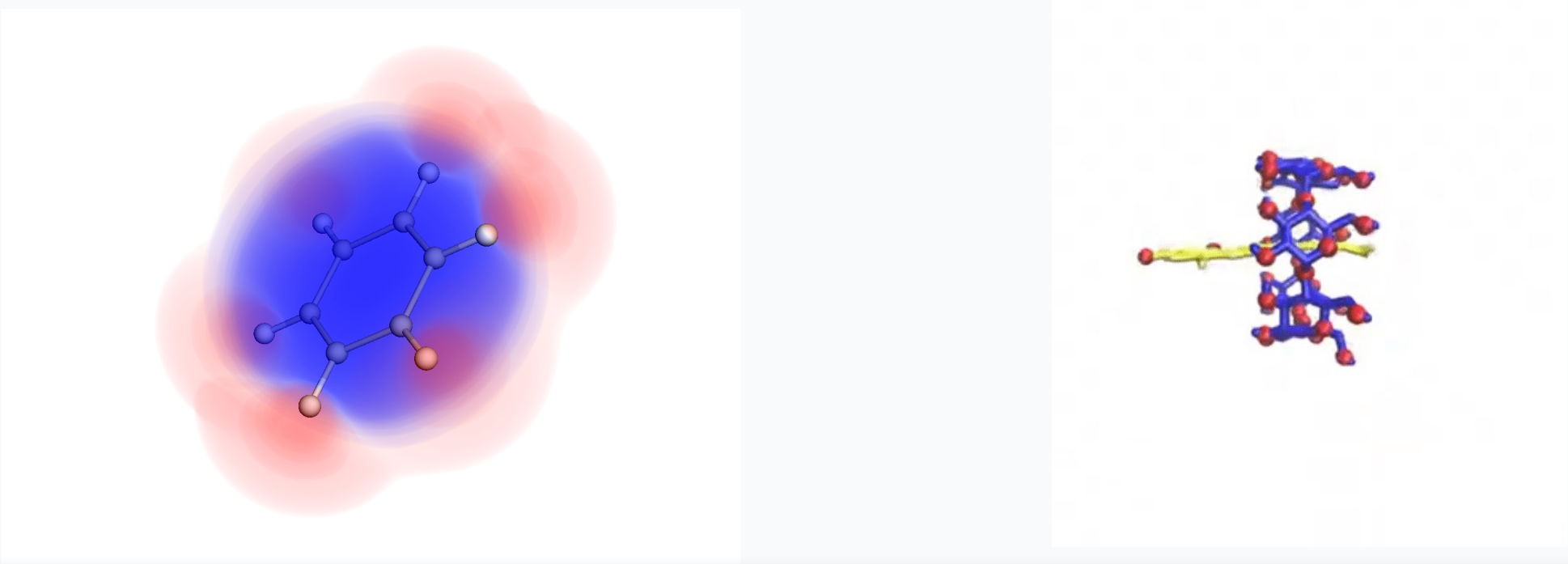
```
def plot_orbital(orbital, mol):
    xyzfmt = f"{len(mol.atom)}\n\n" + mol.tostring()
    v = py3Dmol.view(data=xyzfmt, style={"stick": {"radius": 0.06}, "sphere": {"radius": 0.2}})
    v.addVolumetricData(cube_data(axes, orbital), "cube", build_transferfn(orbital))
    return v
```

Try changing the `mo_index` variable to select the different molecular orbitals benzene.

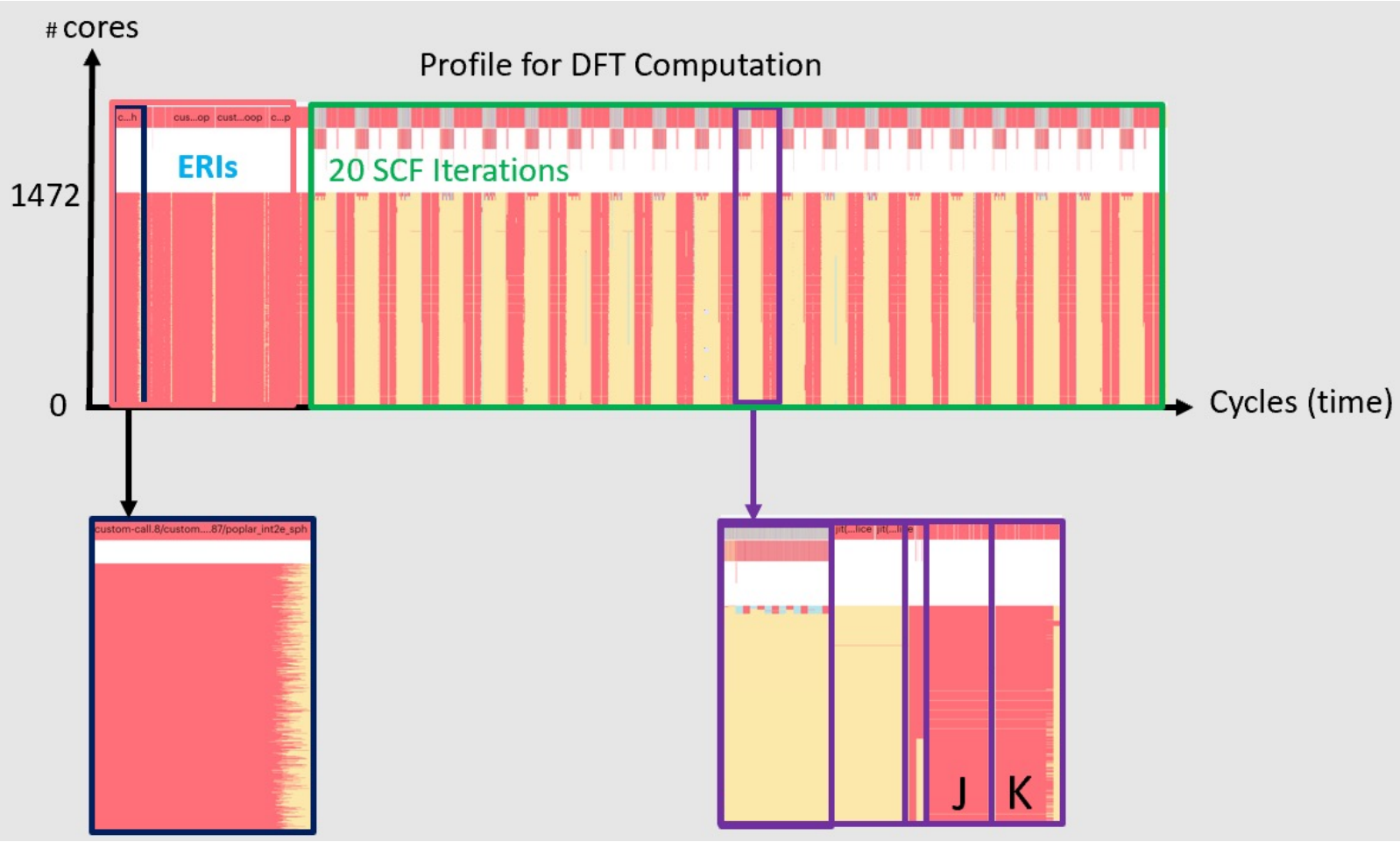
Run 10

```
mo_index = 5

orbital = molecular_orbitals[:, mo_index]
mol_view = plot_orbital(orbital, mol)
mol_view.spin()
```



SHOW MORE



MIMD for
$E \in \mathbb{R}^{N^2 \times N^2}$

65 TB/s for
$\mathrm{J} = E \cdot v \in \mathbb{R}^{N^2}$
$\mathrm{K} = E^T \cdot v \in \mathbb{R}^{N^2}$

BOW IPU

IPU-Tiles™

1472 independent IPU-Tiles™ each with an IPU-Core™ and In-Processor-Memory™

IPU-Core™

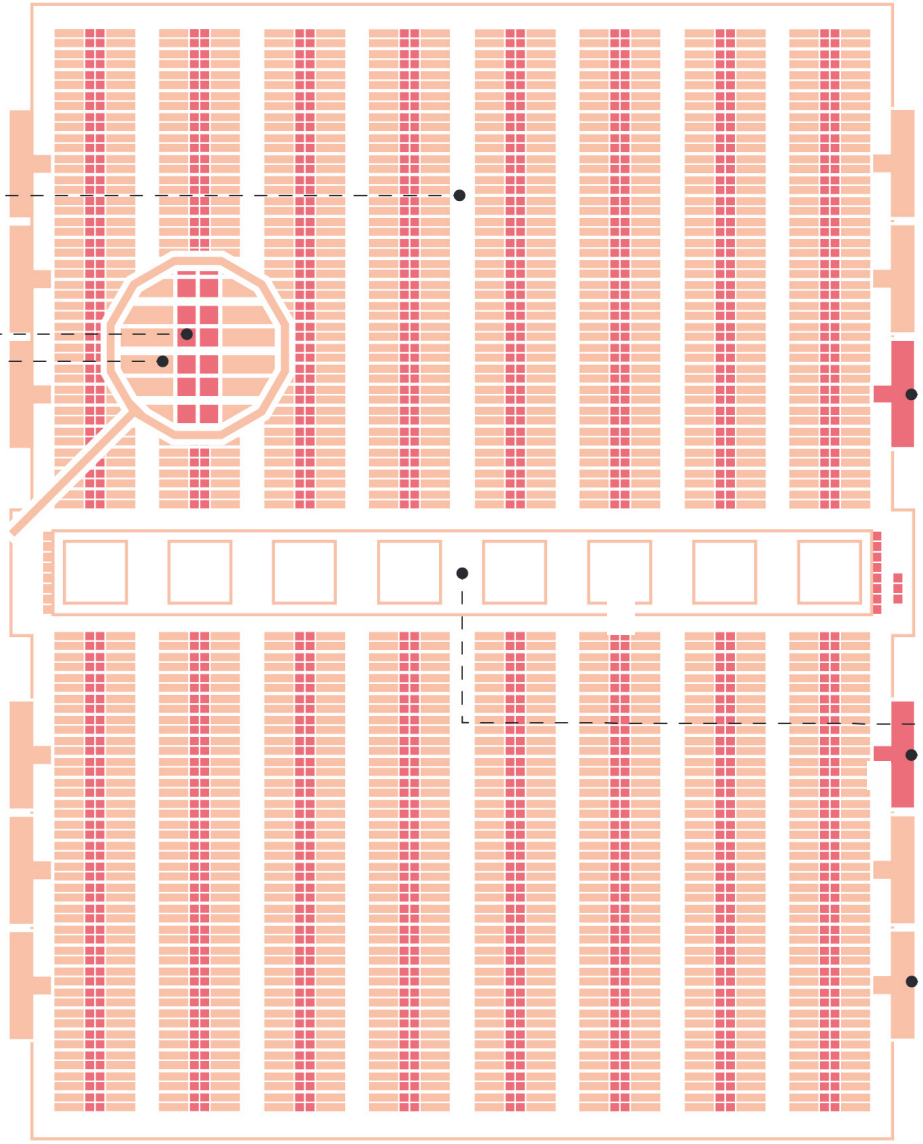
1472 independent IPU-Core™

8832 independent program threads executing in parallel

In-Processor-Memory™

900MB In-Processor-Memory™ per IPU

65TB/s memory bandwidth per IPU



IPU-Exchange™

11 TB/s all to all IPU-Exchange™
Non-blocking, any communication pattern

PCIe

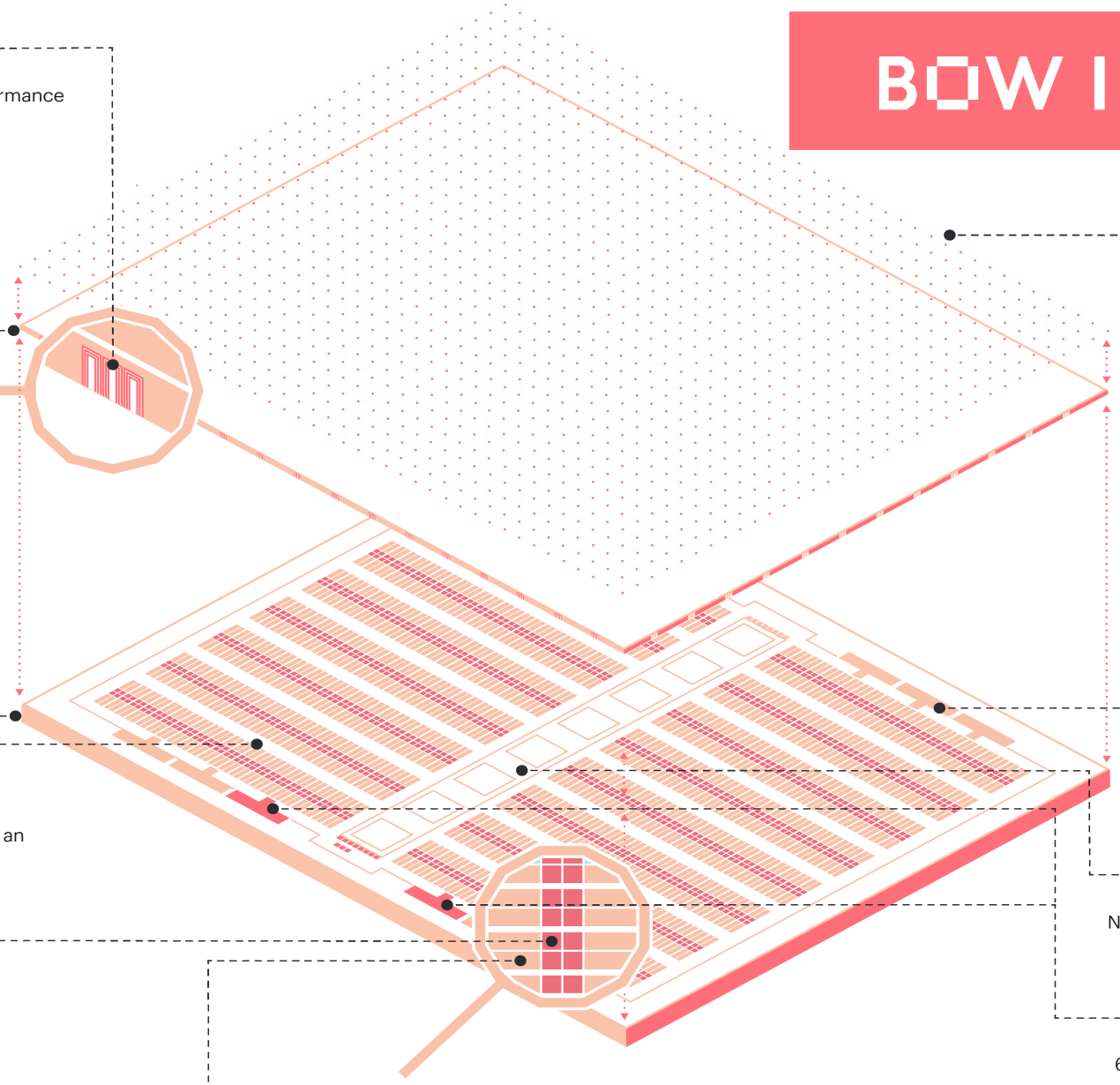
PCI Gen4 x16
64 GB/s bidirectional bandwidth to host

IPU-Links™

10 x IPU-Links,
320GB/s chip to chip bandwidth

GRAPHCORE

BOW IPU PROCESSOR



Deep Trench Capacitor

Efficient power delivery
Enables increase in operational performance

Wafer-On-Wafer

Advanced silicon 3D stacking technology

Closely coupled power delivery die

Higher operating frequency and enhanced overall performance

Solder Bumps

IPU-Links™

10x IPU-Links,
320GB/s chip to chip bandwidth

IPU-Tiles™

1472 independent IPU-Tiles™ each with an IPU-Core™ and In-Processor-Memory™

IPU-Exchange™

11 TB/s all to all IPU-Exchange™
Non-blocking, any communication pattern

IPU-Core™

1472 independent IPU-Core™

8832 independent program threads executing in parallel

PCIe

PCI Gen4 x16
64 GB/s bidirectional bandwidth to host

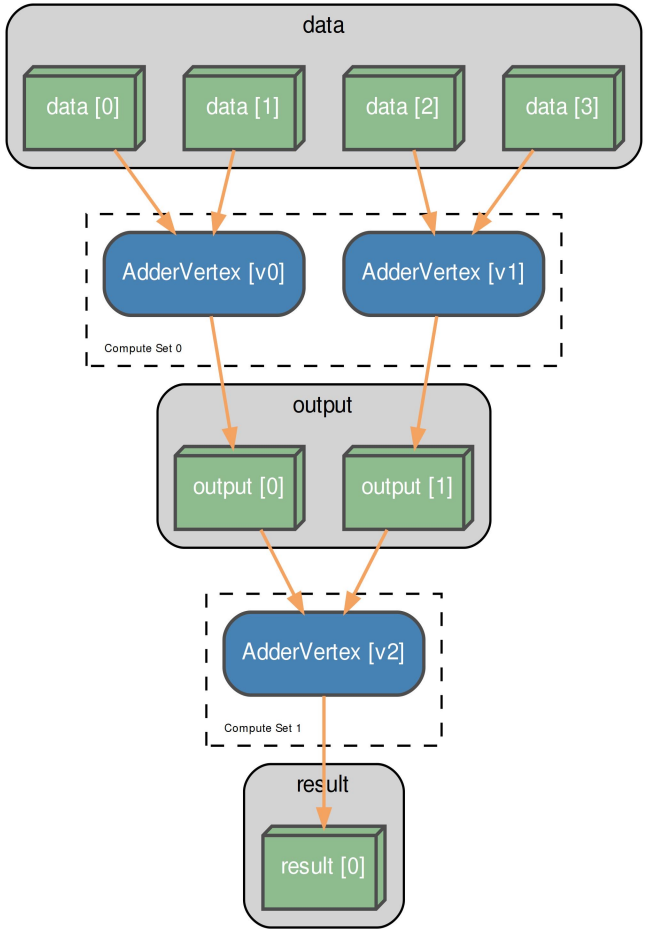
In-Processor-Memory™

900MB In-Processor-Memory™ per IPU

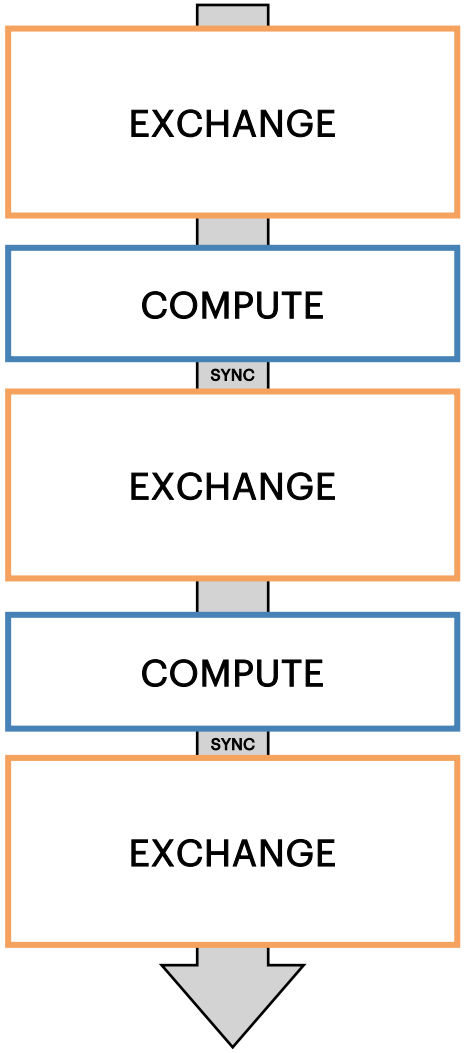
65.4TB/s memory bandwidth per IPU

EXECUTION MODEL

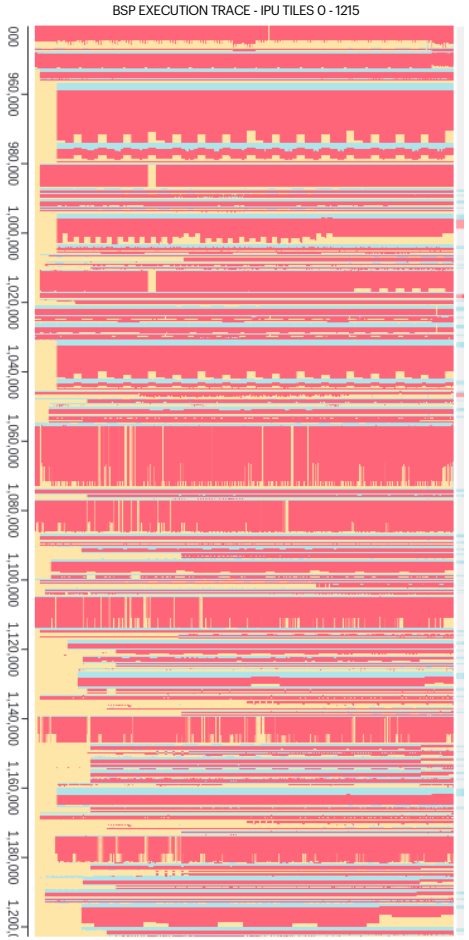
COMPUTATIONAL GRAPH



BSP SCHEDULE



OPTIMIZED IPU EXECUTION



OUTPUT FROM POPVISION GRAPH ANALYSER

GRAPHCORE

BULK SYNCHRONOUS PARALLEL (BSP)

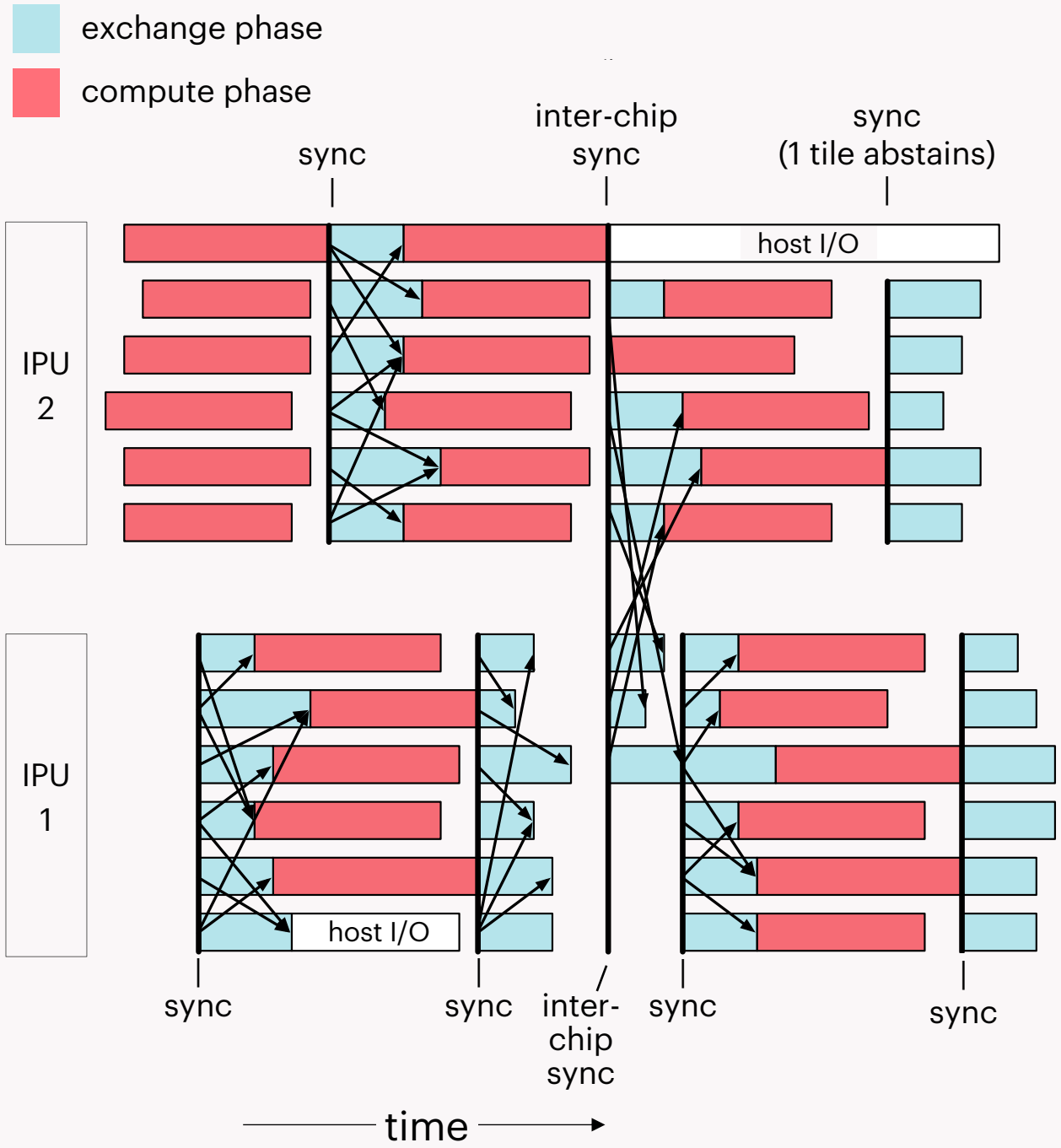
BSP software bridging model – massively parallel computing with no concurrency hazards

3 phases: compute, sync, exchange

Easy to program – no live-locks or dead-locks

Widely-used in parallel computing – Google, FB, ...

First use of BSP inside a parallel processor



BOW-2000 IPU MACHINE

1U blade form factor delivering 1.4 PetaFLOPS AI Compute

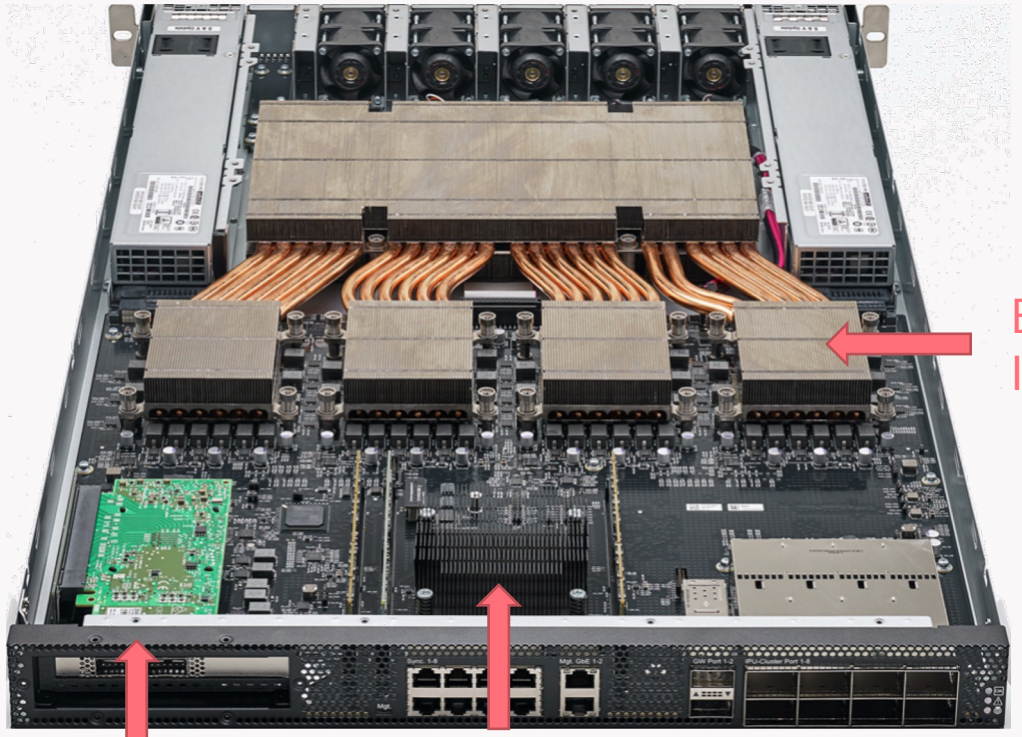
Disaggregated AI/ML accelerator platform

Excellent performance & TCO leveraging In-Processor memory & IPU-Exchange

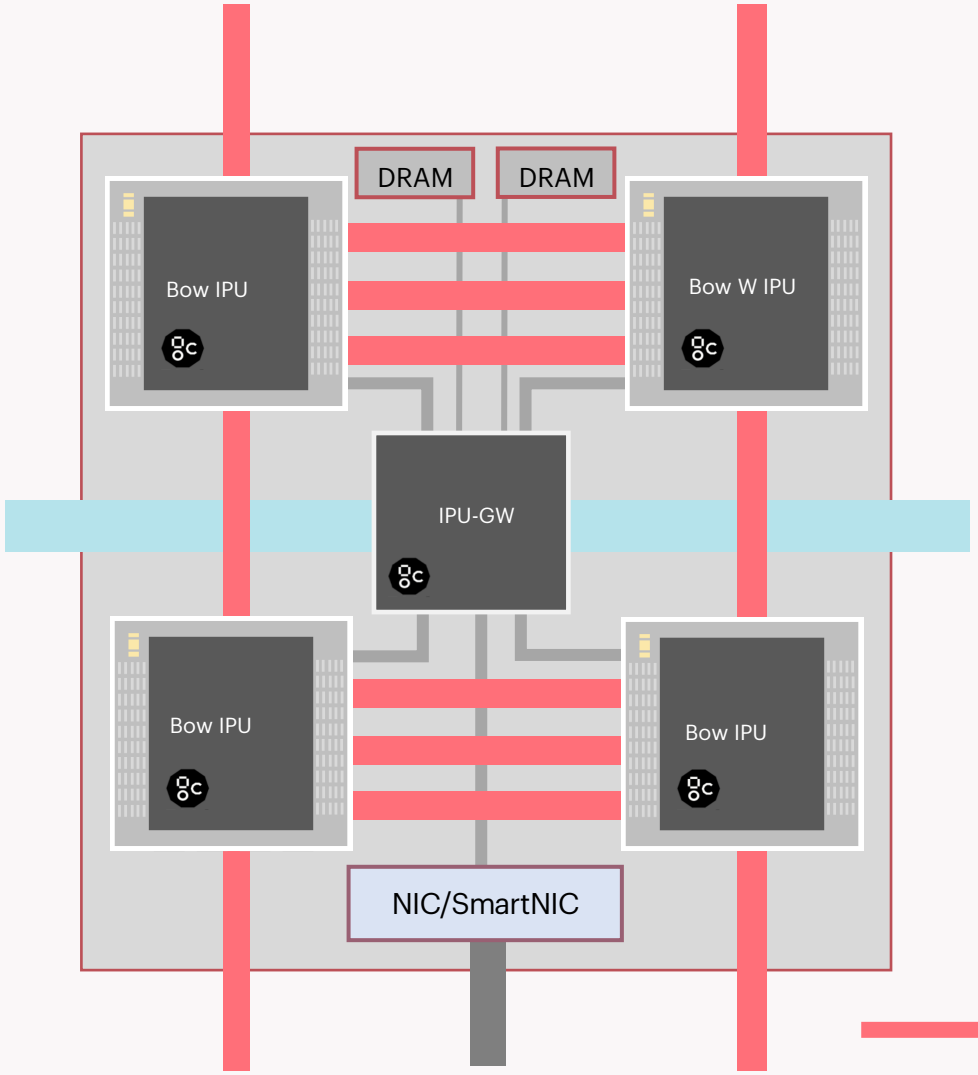
IPU-Links scale to Bow Pod64

Expansion to Bow Pod256 and beyond with IPU-GW Links

BOW IPU-2000



BOW-2000: THE BUILDING BLOCK OF LARGE PODS



COMPUTE

4x Bow IPUs

- 1.4 $PFLOP_{16}$ compute
- 5,888 processor cores
- > 35,000 independent parallel threads

DATA

Exchange Memory

- 3.6GB In-Processor-Memory @ 260 TB/s
- 128GB Streaming Memory DRAM (up to 256GB)

COMMUNICATIONS

IPU-Fabric managed by IPU-GW

- Host-Link – 100GE to Poplar Server for standard data center networking
- IPU-Link – 2D Torus for intra-POD64 communication
- GW-Link - 2x 100Gbps Gateway-Links for rack-to-rack – flexible topology

x16 IPU-Link [64GB/s]

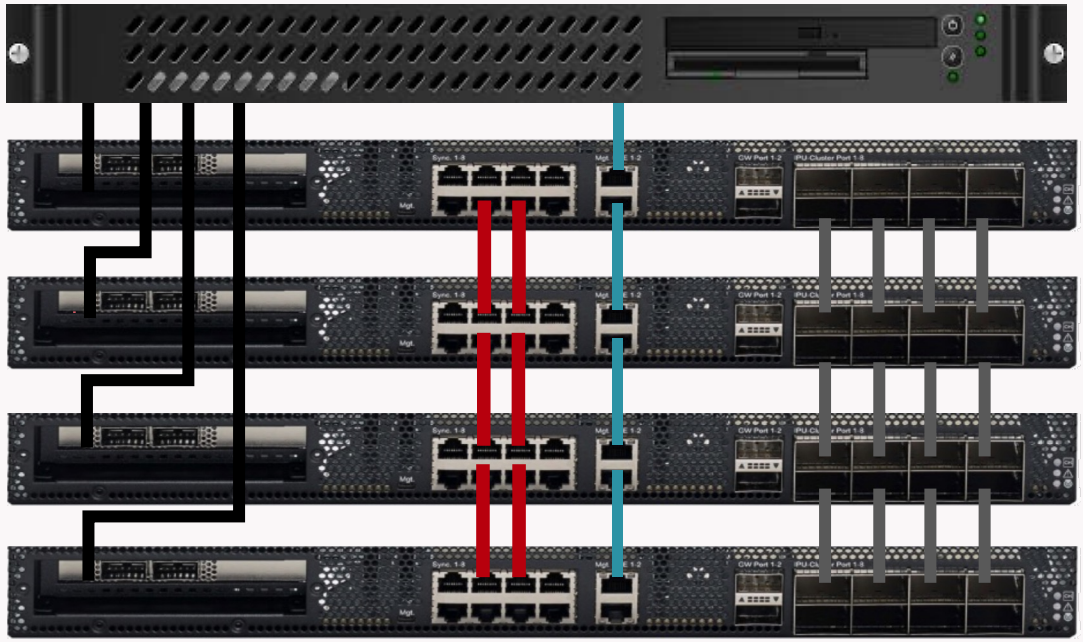
Host-Link Network I/F [100Gbps]

IPU-GW Link [100Gbps]

x8 PCIe G4 [32GB/s]

BOW POD16 DIRECT ATTACH

Simple design with server with 4 Bow-2000 and 16 Bow IPUs



- Host-Link 100GE network interface (QSFP, 1.0m)
- 1GbE Management (Cat5, 1.5m)
- Sync-Link (Cat5, 0.15m)
- IPU-Link (OSFP, 0.3m)

HANDS-ON:

GET STARTED

RUN AN EXAMPLE

HANDOUT

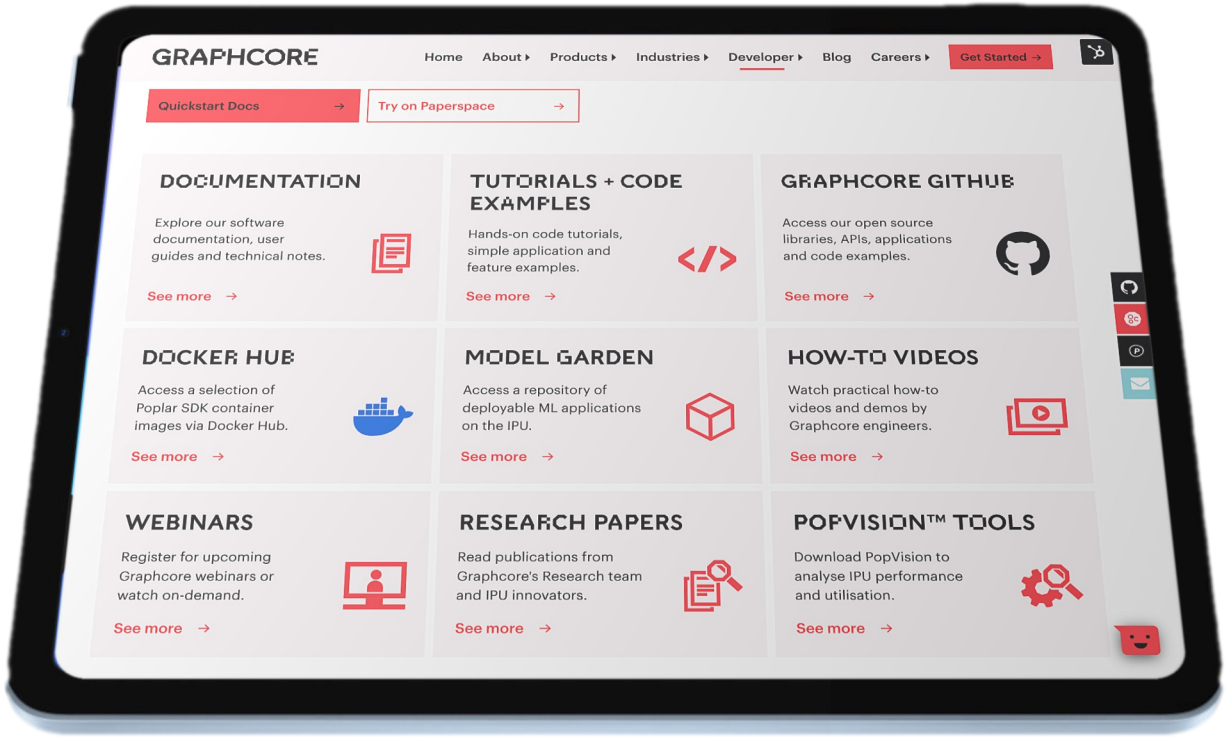
bit.ly/tamu231003

IPU DEVELOPER ECOSYSTEM

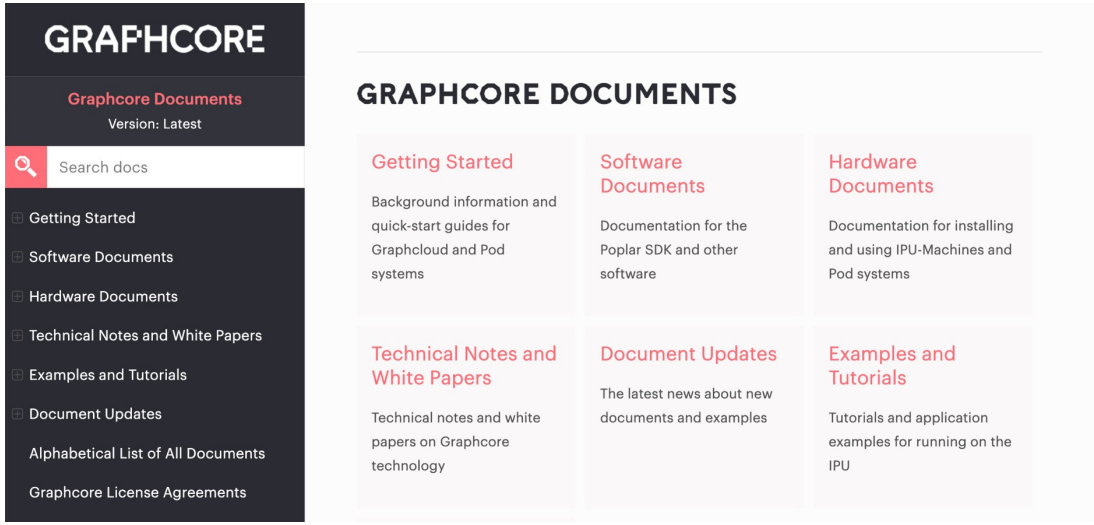
GRAPHCORE

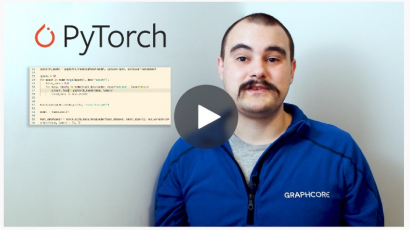
GRAPHCORE SOFTWARE ECOSYSTEM

WORLD CLASS DEVELOPER RESOURCES FOR IPU USERS



WWW.GRAPHCORE.AI/DEVELOPER





Getting started with PyTorch for the IPU

Running a basic model for training and inference

AI Customer Engineer, Chris Bogdiukiewicz introduces PyTorch for the IPU. With PopTorch™ - a simple Python wrapper for PyTorch programs, developers can easily run models, directly on Graphcore IPUs with a few lines of extra code.

In this video, Chris provides a quick demo on running a basic model for both training and inference using a MNIST based example.

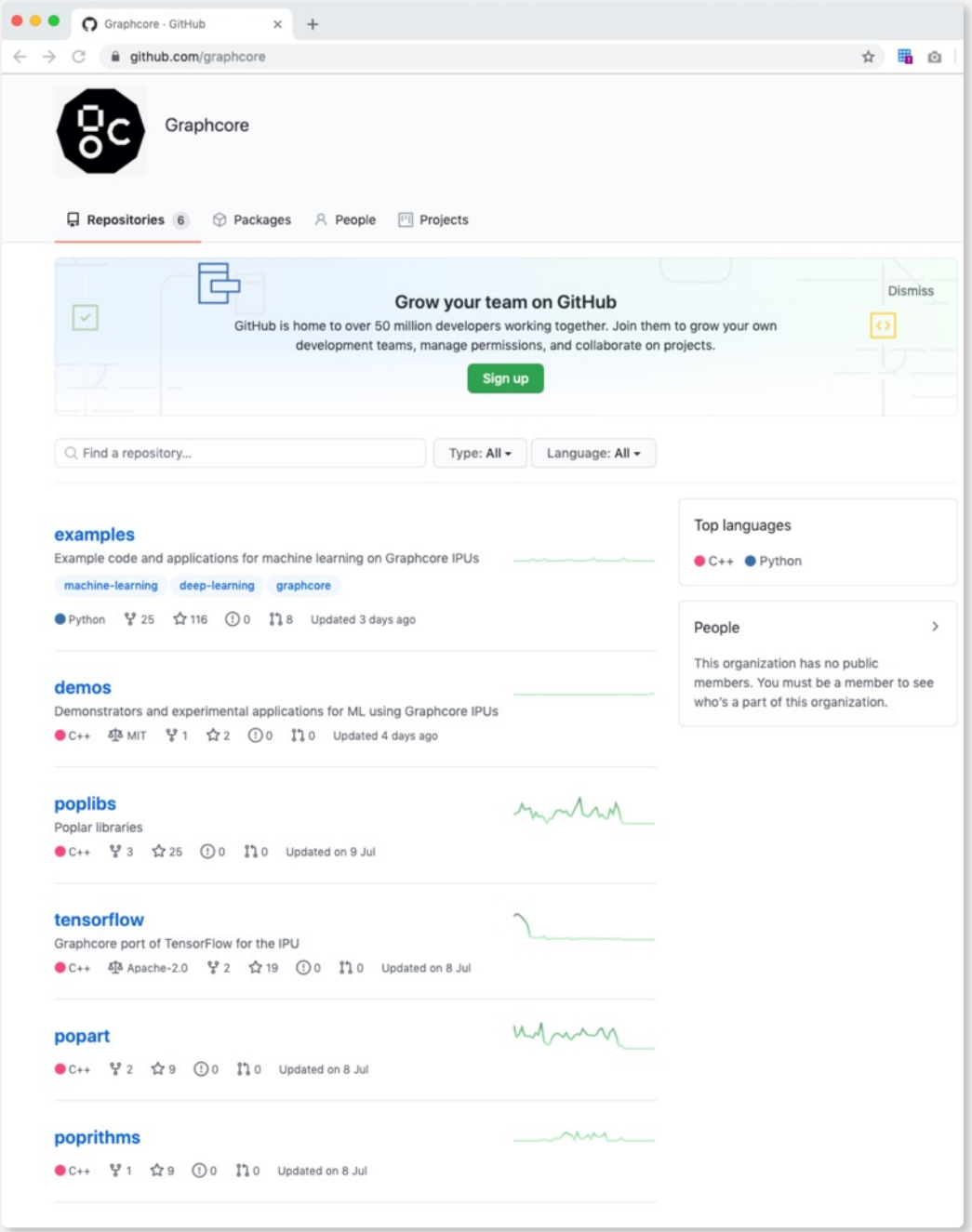
Get the Code →

Read the Guide →

OPEN SOURCE

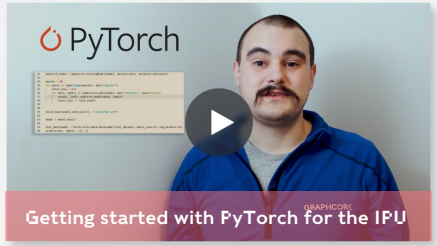
github.com/graphcore

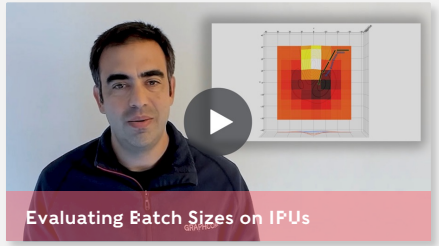
- As part of our ethos to put power in the hands of AI developers, Graphcore open sourced in 2020
- PopLibs™, PopART, PyTorch & TensorFlow for IPU fully open source and available on GitHub
- Our code is public and open for code contributions from the wider ML developer community

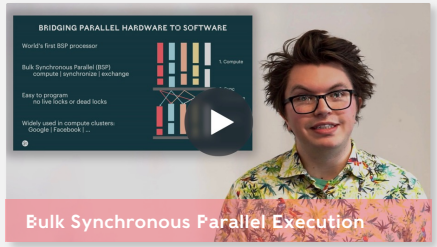


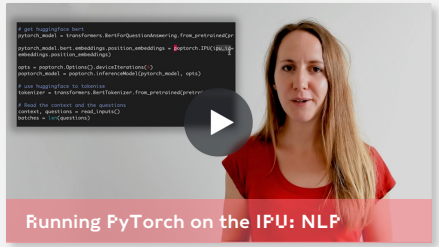
VIDEO + GITHUB TUTORIALS

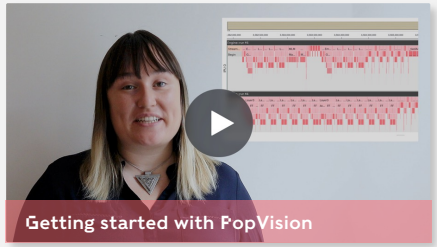
A comprehensive set of online developer training materials and educational content

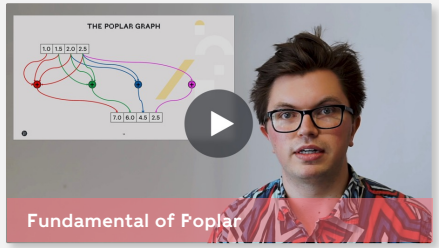


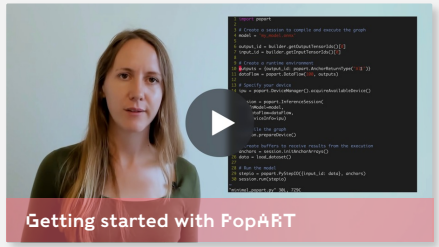


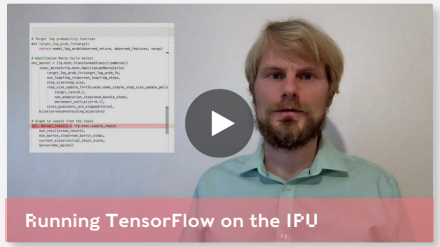












TUTORIALS

Learn how to create and run programs using Poplar and PopLibs with our hands-on programming tutorials.

- Programs and Variables
- Using PopLibs
- Writing Vertex Code
- Profiling Output
- Basic Machine Learning Example
- Matrix-Vector Multiplication
- Matrix-Vector Multiplication Optimisation
- Simple PyTorch for the IPU NEW

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 the outline 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.

The tutorials 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 to create the first variable in the program:

Tutorial 5: a basic machine learning example

This tutorial contains a complete training program that performs a logistic regression on the MNIST data set, using gradient descent. The files for the demo are in `tut5_ml`. There are no coding steps in the tutorial. The task is to understand the code, build it and run it. You can build the code using the supplied makefile.

Before you can run the code you will need to run the `get_mnist.sh` script to download the MNIST data.

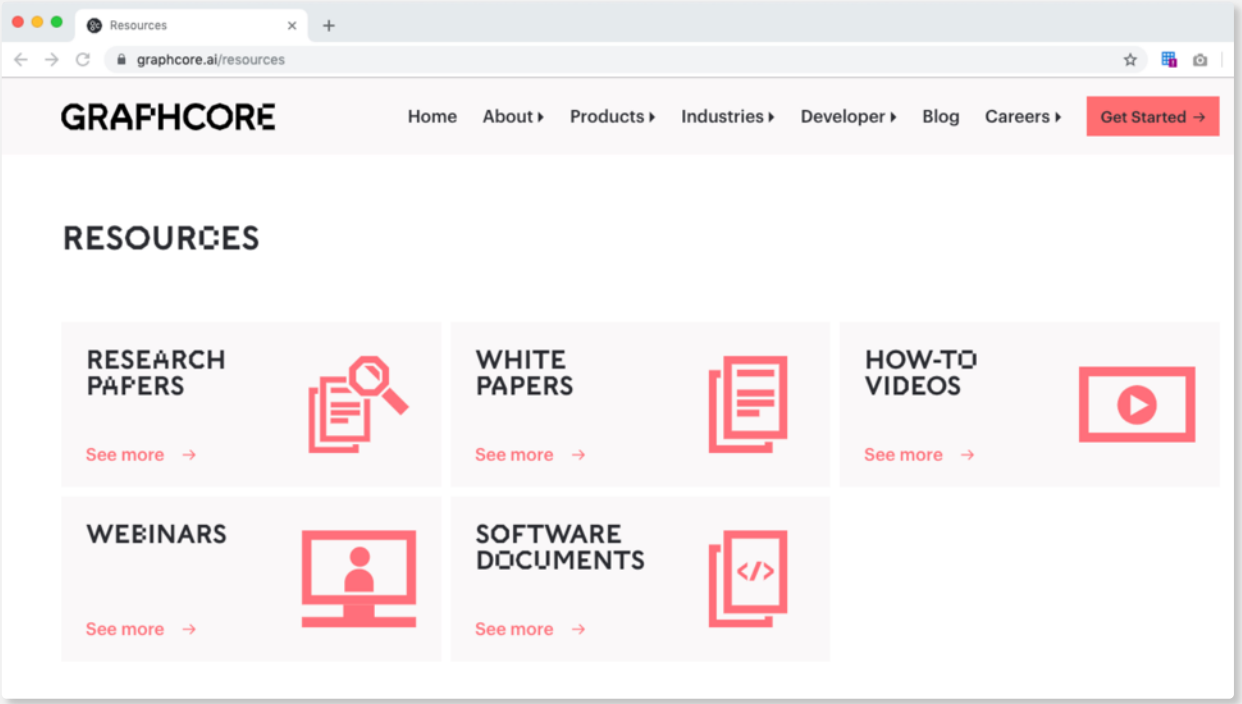
The program accepts an optional command line argument to make it use the IPU hardware instead of a simulated IPU.

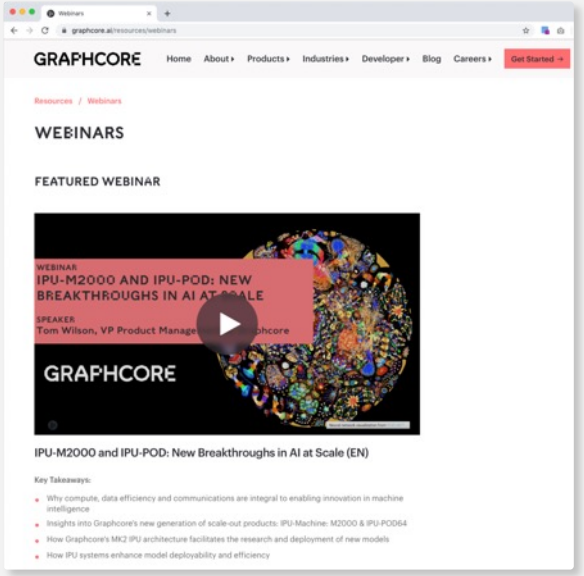
As you would expect, training is significantly faster on the IPU hardware.

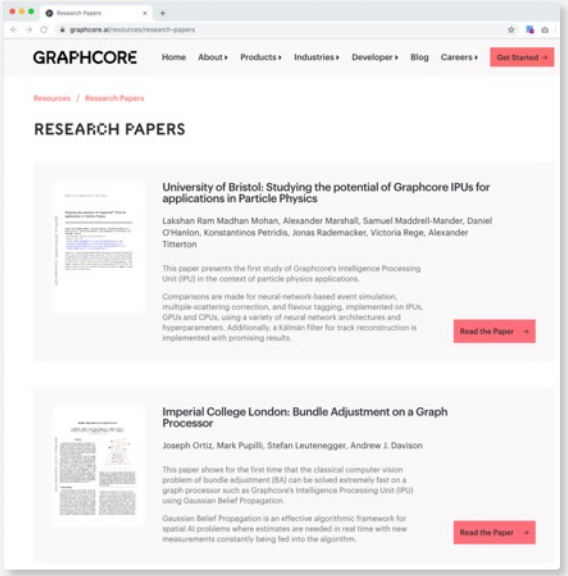
RESOURCES CENTRE

graphcore.ai/resources

- Central source of research papers, white papers, videos, on-demand webinars and documentation
- Product resources for ML Engineers & IT / Infrastructure Managers now available







GRAPHCORE SOFTWARE

NLP/TRANSFORMERS
IMAGE CLASSIFICATION/CNNS
OBJECT DETECTION
LARGE MODELS
MLPERF
CONDITIONAL SPARSITY
GNNS

ML APPLICATIONS

TUTORIALS
CODE EXAMPLES
DOCUMENTATION
VIDEOS
NATIVE IPU CODERS PROGRAM
APPS PORTFOLIO

DEVELOPER ECOSYSTEM

FRONTENDS

JUPYTER NOTEBOOKS

INFERENCE DEPLOYMENT TOOLKIT

FRAMEWORKS

ONNX
HALO
Keras
PaddlePaddle

FW BACKENDS

XLA
POPART+
POPDIST
PARTITIONER
POPIR
POPIT

POPLAR®

POPLIBS
GCL
POPLAR
GRAPH ENGINE
GRAPH COMPILER

DRIVERS

GC DEVICE ACCESS LAYER
IPUOF DRIVER
PCIe DRIVER

POPLAR® SDK

GRAPH ANALYZER
SYSTEM ANALYZER
DEBUGGER
DEVELOPMENT ENVIRONMENT

POPVISION TOOLS

V-IPU

SYSTEM MONITORING

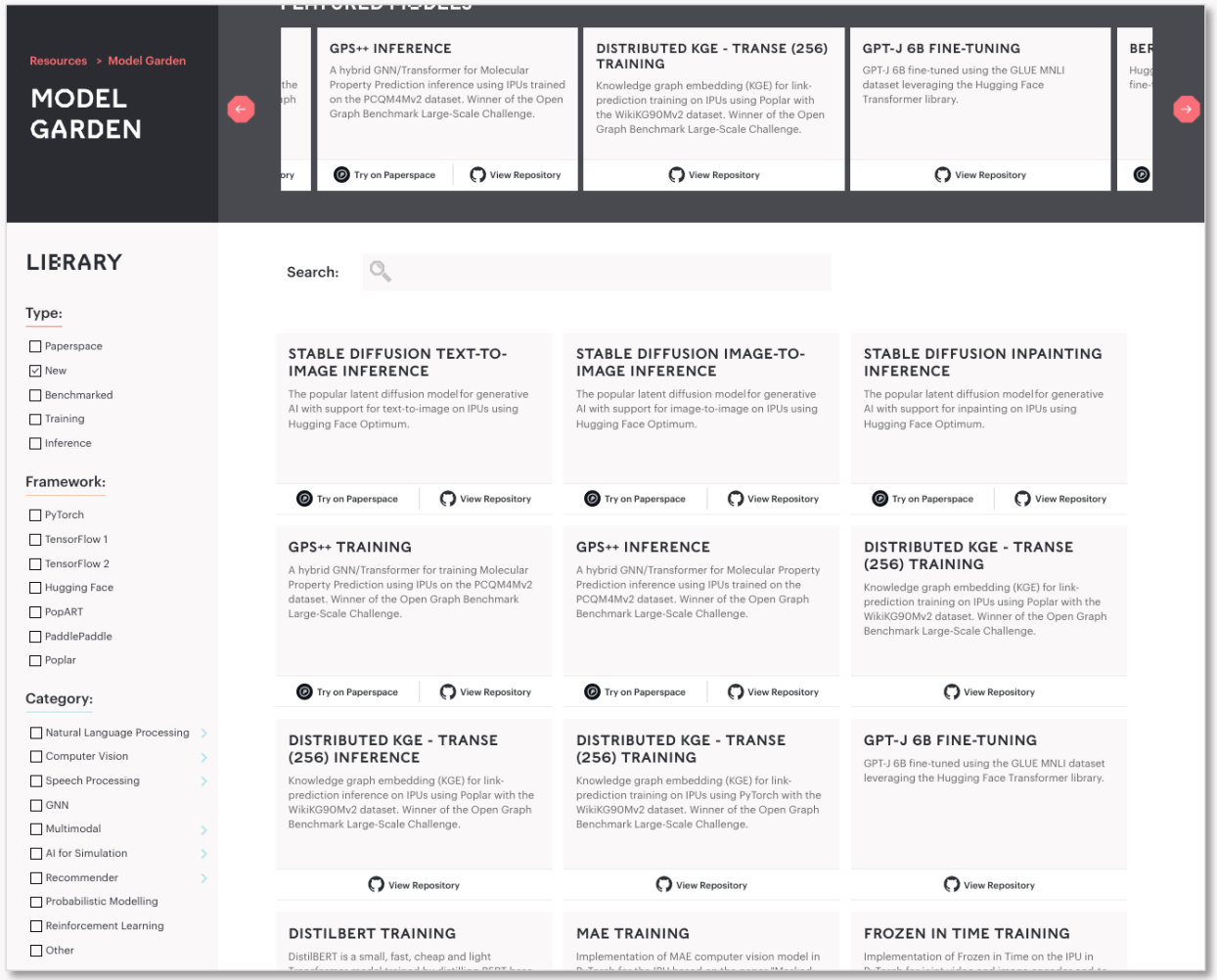
PROMETHEUS
GRAFANA

JOB DEPLOYMENT

K8S
SLURM

SYSTEM SOFTWARE

ENHANCED MODEL GARDEN



PUBLIC ACCESS TO WIDE VARIETY OF MODELS, READY TO RUN ON IPU

NEW FILTER/SEARCH CAPABILITY

DIRECT ACCESS TO GITHUB

PAPERSPACE NOTEBOOK LINKS

https://www.graphcore.ai/resources/model-garden

Access IPU-ready notebooks in seconds

Paperspace

COMPUTER VISION

Stable Diffusion 2
Text => Image Inference
Run on Gradient

Stable Diffusion
Text => Image Inference
Run on Gradient

Stable Diffusion
Image => Image Inference
Run on Gradient

Stable Diffusion
Text Guided In-Painting Inference
Run on Gradient

Training ViT HF model
ViT - Fine Tuning
Run on Gradient

Image Classification
ViT - Fine Tuning
Run on Gradient

Object Detection
YOLO v4 - Inference
Run on Gradient
NEW

NATURAL LANGUAGE PROCESSING (NLP)

BERT Fine Tuning
BERT-Large - Fine Tuning
Run on Gradient

Fast sentiment analysis BERT | RoBERTa - Inference
Run on Gradient

Named Entity Recognition
BERT - Inference
Run on Gradient

Multiple choice task
RoBERTa – Fine Tuning
Run on Gradient

Question-Answering task
RoBERTa – Fine Tuning
Run on Gradient

Instruction Tuned LLM
Dolly 2.0 - Inference
Run on Gradient
NEW

Flan-T5 is all you need
Fine-Tuning & Inference
Run on Gradient
NEW

Chatbot Open Source LLM
Llama 2 - Inference
Run on Gradient
NEW

Summarization task
T5 Small – Fine Tuning
Run on Gradient

Text Classification task
RoBERTa – Fine Tuning
Run on Gradient

Token Classification task
BERT
Run on Gradient

Translation task
BART-Base
Run on Gradient

Text Entailment
GPT-J – Fine Tuning
Run on Gradient

Text Generation
GPT-J
Run on Gradient

Chatbot using OpenAssistant
Pythia 12B - Inference
Run on Gradient
NEW

SQuAD & MNLI
DeBERTa - Inference
Run on Gradient
NEW

GNN

Training Large Graphs
Cluster-GCN - Training
Run on Gradient

Training Dynamic Graphs
TGN - Training
Run on Gradient

Predicting molecular properties
SchNet - Training
Run on Gradient

Predicting molecular properties
GIN - Training
Run on Gradient

Predicting molecular properties
GPS++ (OGB-LSC) Train | Inf
Run on Gradient

Link Prediction training
Dist KGE (OGB-LSC) - Train
Run on Gradient

Link Prediction training
NBFNet - Training
Run on Gradient

Molecular Prediction
MolFeat – FT & Inf
Run on Gradient

Molecular Modelling with Graphium
GCN/GIN – Train & Inf
Run on Gradient
NEW

SPEECH PROCESSING

Fine-Tuning a wav2vec
wav2vec – Fine Tuning
Run on Gradient

Running ASR
wav2vec - Inference
Run on Gradient

Speech Transcription on IPUs
Whisper – FT & Inf
Run on Gradient
NEW

TIME SERIES

Multi-horizon financial forecasting
DeepLOB Seq2Seq – Train & Inf
Run on Gradient
NEW

Multi-horizon financial forecasting
DeepLOB Attention – Train & Inf
Run on Gradient
NEW

https://www.graphcore.ai/ipu-jupyter-notebooks

USEFUL ENV VARIABLES

LOGGING

Logging messages can be generated when your program runs. This is controlled by the environment variables described below. For more detailed information see the docs: https://docs.graphcore.ai/projects/poplar-user-guide/en/latest/env-vars.html

POPLAR_LOG_LEVEL: Enable logging for Poplar

POPLAR_LOG_DEST: Specify the destination for Poplar logging (“stdout”, “stderr” or a file name)

“OFF”	No logging information. The default.
“ERR”	Only error conditions will be reported.
“WARN”	Warnings when, for example, the software cannot achieve what was requested (for example, if the convolution planner can’t keep to the memory budget, or Poplar has determined that the model won’t fit in memory but the debug.allowOutOfMemory option is enabled).
“INFO”	Very high level information, such as PopLibs function calls.
“DEBUG”	Useful per-graph information.
“TRACE”	The most verbose level. All useful per-tile information.

CREATE EXECUTION PROFILE

POPLAR_ENGINE_OPTIONS='{"autoReport.all":"true", "autoReport.directory":"./report"}’

- The PopVision Graph Analyser uses report files generated during compilation and execution by the Poplar SDK.
- These files can be created using POPLAR_ENGINE_OPTIONS.
- In order to capture the reports needed for the PopVision Graph Analyser you only need to set POPLAR_ENGINE_OPTIONS='{"autoReport.all":"true"}' before you run a program. By default this will enable instrumentation and capture all the required reports to the current working directory.

EXECUTABLE CACHE

If you often run the **same models** you might want to enable executable caching to save time:

POPTORCH:

- You can do this by either setting the POPTORCH_CACHE_DIR environment variable or by calling poptorch.Options.enableExecutableCaching.

TENSORFLOW:

- You can use the flag --executable_cache_path to specify a directory where compiled files will be placed. Fused XLA/HLO graphs are hashed with a 64-bit hash and stored in this directory.

Warning
The cache directory might grow large quickly. Poplar doesn't evict old models from the cache and, depending on the number and size of your models and the number of IPUs used, the executables might be quite large.
It is the your responsibility to delete the unwanted cache files.

SYNTHETIC-DATA

TF_POPLAR_FLAGS= "--use_synthetic_data --synthetic_data_initializer=random"

Used for measuring the IPU-only throughput and disregards any host/CPU activity.

GRAPHCORE COMMAND LINE TOOLS

gc-docker Allows you to use IPU devices in Docker containers using the Docker container engine.

gc-flops Allows you to benchmark the number of floating point operations per second on one or more IPU processors.

gc-info Determines what IPU cards are present in the system.

gc-inventory Lists device IDs, physical parameters and firmware version numbers.

gc-links Displays the status and connectivity of each of the IPU-Links that connect IPUs. See also *IPU-Link channel mapping* for connectivity in an IPU Server containing C2 cards.

gc-monitor Monitors IPU activity on shared systems.

gc-reset Resets IPU devices.

gc-exchangetest Allows you to test the internal exchange fabric in an IPU.

gc-exchangewritetest Tests direct writes to the IPU's tile memory via the host.

gc-gwlinkstraffictest Tests GW-Links on multi-rack IPU-POD systems.

gc-hostsynclatencytest Reports the latency of transfers between the host machine and the IPUs (in both directions).

gc-hosttraffictest Allows you to test the data transfer between the host machine and the IPUs (in both directions).

gc-iputraffictest Allows you to test the data transfer between IPUS.

gc-memorytest Tests all the memory in an IPU, reporting any tiles that fail.

gc-podman Allows you to use IPU devices in Docker containers using the Podman container engine.

gc-powertest Tests power consumption and temperature of the IPU processors.

https://docs.graphcore.ai/projects/command-line-tools/en/latest/index.html

TF2/KERAS ON IPU

KERAS ON IPU

IPU optimized Keras `Model` and `Sequential` with the following features:

- On-device training loop for reduction of communication overhead.
- Gradient accumulation for simulating larger batch sizes.
- Automatic data-parallelisation of the model when placed on a multi-IPU device.

gpu_cnn_keras.py ↔ ipu_cnn_keras.py tf_keras

Keras

GPU

```
as tf
ras.layers import *

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
x_train = x_train.astype('float32') / 255.0
y_train = tf.keras.utils.to_categorical(y_train, 10)
ds_train = tf.data.Dataset.from_tensor_slices((x_train, y_train)).batch(64, drop_remainde

model = tf.keras.Sequential([
    Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:]),
    Activation('relu'),
    Conv2D(32, (3, 3)),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
    Conv2D(64, (3, 3), padding='same'),
    Activation('relu'),
    Conv2D(32, (3, 3)),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
    Flatten(),
    Dense(512),
    Activation('relu'),
    Dropout(0.5),
    Dense(10),
    Activation('softmax')
    ])

model.compile(loss='categorical_crossentropy',
    optimizer=tf.optimizers.SGD(learning_rate=0.016),
    metrics=['accuracy'])

model.fit(ds_train, epochs=40)
```

IPU

```
import tensorflow as tf
from tensorflow.keras.layers import *
+ from tensorflow.python import ipu

+ cfg = ipu.config.IPUConfig()
+ cfg.auto_select_ipus = 1
+ cfg.configure_ipu_system()
+ with ipu.ipu_strategy.IPUStrategy().scope():
    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
    x_train = x_train.astype('float32') / 255.0
    y_train = tf.keras.utils.to_categorical(y_train, 10)
    ds_train = tf.data.Dataset.from_tensor_slices((x_train, y_train)).batch(64, drop_rema

    model = tf.keras.Sequential([
        Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:]),
        Activation('relu'),
        Conv2D(32, (3, 3)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),
        Conv2D(64, (3, 3), padding='same'),
        Activation('relu'),
        Conv2D(32, (3, 3)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),
        Flatten(),
        Dense(512),
        Activation('relu'),
        Dropout(0.5),
        Dense(10),
        Activation('softmax')
        ])

    model.compile(loss='categorical_crossentropy',
        optimizer=tf.optimizers.SGD(learning_rate=0.016),
        metrics=['accuracy'])

    model.fit(ds_train, epochs=40)
```

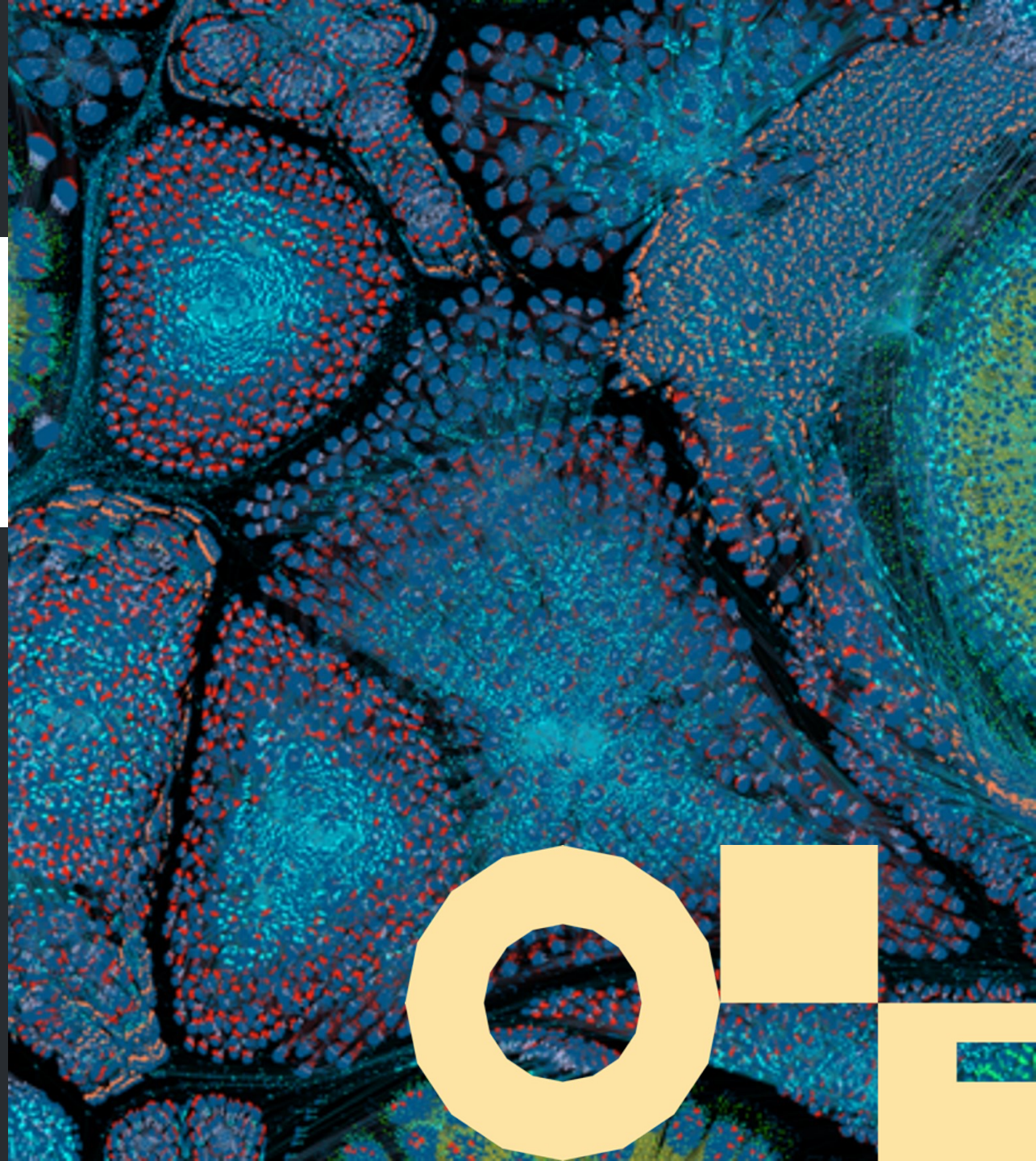
TF2/KERAS TUTORIALS

Continued in the repositories below (follow the READMEs)

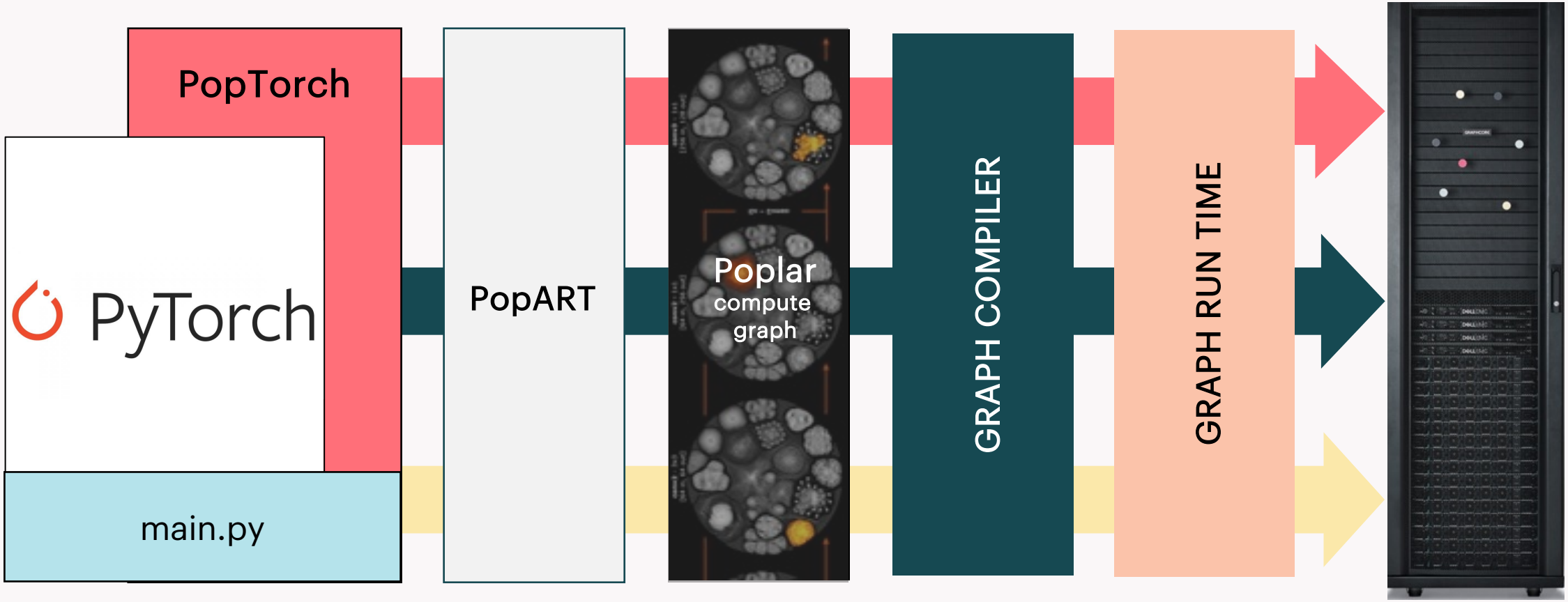
github.com/graphcore/examples/tree/master/tutorials/tutorials/tensorflow2/keras

INTRO TO POPTORCH

GRAPHCORE



WHAT IS POPTORCH?



WHAT IS POPTORCH?

- PopTorch is a set of extensions for PyTorch to enable PyTorch models to run on Graphcore's IPU hardware.
- PopTorch supports both inference and training. To run a model on the IPU you wrap your existing PyTorch model in either a PopTorch inference wrapper or a PopTorch training wrapper.
- You can provide further annotations to partition the model across multiple IPUs. Using the user-provided annotations, PopTorch will use PopART to parallelise the model over the given number of IPUs.
- Additional parallelism can be expressed via a replication factor which enables you to data-parallelise the model over more IPUs.

PYTORCH FOR IPU

PyTorch

Define a model within PyTorch

Create an IPU execution wrapper around the model and run as normal

PopTorch uses PyTorch dispatcher to trace the model

Compile the graph in PopART and then run on one or more IPUs

GETTING STARTED: TRAINING A MODEL

TRAINING A MODEL

1. Import packages

PopTorch is a separate package from PyTorch, and must be imported.

2. Load dataset using torchvision.datasets and poptorch.DataLoader

In order to make data loading easier and more efficient, PopTorch offers an extension of `torch.utils.data.DataLoader` class:
`poptorch.DataLoader` class is specialised for the way the underlying PopART framework handles batching of data.

3. Define model and loss function using torch API

The only difference here from pure PyTorch is the loss computation, which has to be part of the forward function. This is to ensure the loss is computed on the IPU and not on the CPU, and to give us as much flexibility as possible when designing more complex loss functions.

TRAINING A MODEL

4. Prepare training

Instantiate compilation and execution options, these are used by PopTorch's wrappers such as `poptorch.DataLoader` and `poptorch.trainingModel`.

5. Train the model

Define the optimizer using PyTorch's API.

Use poptorch.trainingModel wrapper, to wrap your PyTorch model. This wrapper will trigger the compilation of our model, using TorchScript, and manage its translation to a program the IPU can run. Then run your training loop.

PyTorch

GPU

```
    _, ind = torch.max(predictions, 1)
    # provide labels only for samples, where prediction is available (during the training, not
    ions.size()[0]:]
    rch.eq(ind, labels)).item() / labels.size(  0

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='MNIST training in PopTorch')
    parser.add_argument('--batch-size', type=int, default=8, help='batch size for training (de
    parser.add_argument('--test-batch-size', type=int, default=8, help='batch size for testing
    parser.add_argument('--epochs', type=int, default=10, help='number of epochs to train (def
    parser.add_argument('--lr', type=float, default=0.05, help='learning rate (default: 0.05)

    args = parser.parse_args()

    training_data = torch.utils.data.DataLoader(
                        torchvision.datasets.MNIST('mnist_data/', train=True, download=True, t
                        batch_size=args.batch_size, shuffle=True, drop_last=True)
    test_data = torch.utils.data.DataLoader(
                        torchvision.datasets.MNIST('mnist_data/', train=False, download=True,

    model = Network()
    training_model = TrainingModelWithLoss(model)
    optimizer=optim.SGD(model.parameters(), lr=args.lr)

    # Run training
    for _ in range(args.epochs):
        for data, labels in training_data:
            preds, losses = training_model(data, labels)
            optimizer.zero_grad()
            losses.backward()
            optimizer.step()

    # Run validation
    sum_acc = 0.0
    with torch.no_grad():
        for data, labels in test_data:
            output = model(data)
            sum_acc += accuracy(output, labels)
    print("Accuracy on test set: {:0.2f}%".format(sum_acc / len(test_data)))
```

IPU

```
    _, ind = torch.max(predictions, 1)
    # provide labels only for samples, where prediction is available (during the training, not
    labels = labels[-predictions.size()[0]:]
    accuracy = torch.sum(torch.eq(ind, labels)).item() / labels.si  100.0
    return accuracy

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='MNIST training in PopTorch')
    parser.add_argument('--batch-size', type=int, default=8, help='batch size for training (de
    parser.add_argument('--test-batch-size', type=int, default=8, help='batch size for testing
    parser.add_argument('--epochs', type=int, default=10, help='number of epochs to train (def
    parser.add_argument('--lr', type=float, default=0.05, help='learning rate (default: 0.05)'
    parser.add_argument('--device-iterations', type=int, default=50, help='device iterations (
    args = parser.parse_args()

    opts = poptorch.Options().deviceIterations(args.device_iterations)
    training_data = poptorch.DataLoader(opts,
                        torchvision.datasets.MNIST('mnist_data/', train=True, download=True, trans
                        batch_size=args.batch_size, shuffle=True, drop_last=True)
    test_data = poptorch.DataLoader(opts,
                        torchvision.datasets.MNIST('mnist_data/', train=False, download=True, tran

    model = Network()
    training_model = TrainingModelWithLoss(model)
    optimizer=optim.SGD(model.parameters(), lr=args.lr)
    training_model = poptorch.trainingModel(training_model, opts, optimizer=optimizer)
    inference_model = poptorch.inferenceModel(model)

    # Run training
    for _ in range(args.epochs):
        for data, labels in training_data:
            preds, losses = training_model(data, labels)

    # Detach the training model so that the same IPU could be used for validation
    training_model.detachFromDevice()

    # Run validation
    sum_acc = 0.0
    with torch.no_grad():
        for data, labels in test_data:
            output = inference_model(data)
            sum_acc += accuracy(output, labels)
    print("Accuracy on test set: {:0.2f}%".format(sum_acc / len(test_data)))
```

POPTORCH.OPTIONS

- The compilation and execution on the IPU can be controlled using `poptorch.Options`

- Full list of options available here: https://docs.graphcore.ai/projects/poptorch-user-guide/en/latest/overview.html#options

- Some examples:

  (i) **`deviceIterations`**
  This option specifies the number of batches that is prepared by the host (CPU) for the IPU. The higher this number, the less the IPU has to interact with the CPU, for example to request and wait for data, so that the IPU can loop faster. However, the user will have to wait for the IPU to go over all the iterations before getting the results back. The maximum is the total number of batches in your dataset, and the default value is 1.

  (ii) **`replicationFactor`**
  This is the number of replicas of a model. We use replicas as an implementation of data parallelism. To achieve the same behavior in pure PyTorch, you'd wrap your model with torch.nn.DataParallel, but with PopTorch, this is an option.

POPTORCH TUTORIALS

Continued in the repositories below (follow the READMEs)

github.com/graphcore/examples/tree/master/tutorials/tutorials/pytorch/basics

github.com/graphcore/examples/tree/master/tutorials/tutorials/pytorch/mixed_precision

github.com/graphcore/examples/tree/master/tutorials/tutorials/pytorch/efficient_data_loading

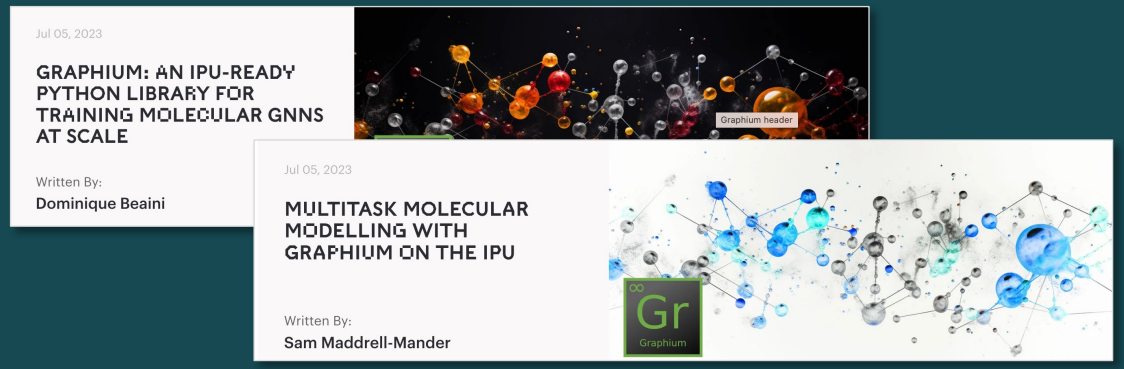
github.com/graphcore/examples/tree/master/tutorials/tutorials/pytorch/pipelining

GRAPHIUM FOR IPU

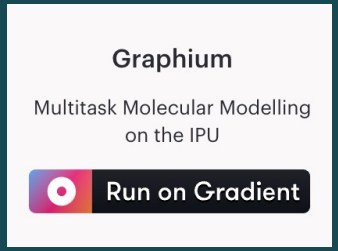
A POWERFUL AND FLEXIBLE OPEN-SOURCE PYTHON LIBRARY FOR TRAINING MOLECULAR GNNS AT SCALE

Graphium integrates state-of-the-art Graph Neural Network (GNN) architectures and a user-friendly API, enabling the easy construction and training of custom GNN models.

ANNOUNCEMENT | TECHNICAL BLOG | GETTING STARTED



RUN GRAPHIUM ON IPU WITH PAPERSPACE JUPYTER NOTEBOOK



Domain: Molecules
Tasks: Multitask
Model: GCN/GIN/GINE
Datasets: QM9, Zinc, Tox21
Workflow: Training, validation, inference
Execution time: 20 mins

PyG

PYTORCH GEOMETRIC FOR IPU

PyG is the ultimate library for Graph Neural Networks

Build graph learning pipelines with ease

kumo GRAPHCORE HARVARD MEDICAL SCHOOL

pyg.org

"The suitability of IPUs for running GNNs and the kind of performance advantage that Graphcore and its customers have demonstrated is really helping to accelerate the uptake of this exciting model class"
Matthias Fey – PyG creator & founder of Kumo.ai

ANNOUNCEMENT | TECHNICAL BLOG | GETTING STARTED

Apr 05, 2023 \ Poplar, PyTorch, GNN

GRAPHCORE USERS CAN NOW BUILD AND RUN GNNS WITH PYTORCH GEOMETRIC

Written By:
Blazej Banaszewski

Apr 05, 2023 \ Developer, PyTorch, GNN

ACCELERATING PYG ON IPUS: UNLEASH THE POWER OF GRAPH NEURAL NETWORKS

Written By:
Blazej Banaszewski, Adam S
Akash Swamy & Mihai Polce

Apr 05, 2023 \ Developer, GNN, Paperspace

GETTING STARTED WITH PYTORCH GEOMETRIC (PYG) ON GRAPHCORE IPUS

Written By:
Adam Sanders and Arianna Saracino

RUN GNN MODELS IN PYG ON PAPERSPACE JUPYTER NOTEBOOKS

Training Dynamic Graphs
TGN
Training
Run on Gradient

Training Large Graphs
Cluster-GCN
Training
Run on Gradient

Predicting molecular properties
SchNet
Training
Run on Gradient

Predicting molecular properties
GIN
Training
Run on Gradient

Link Prediction training
NBFNet
Training
Run on Gradient

Molecular Modelling with Graphium
GCN/GIN
Training & Inf
Run on Gradient
NEW

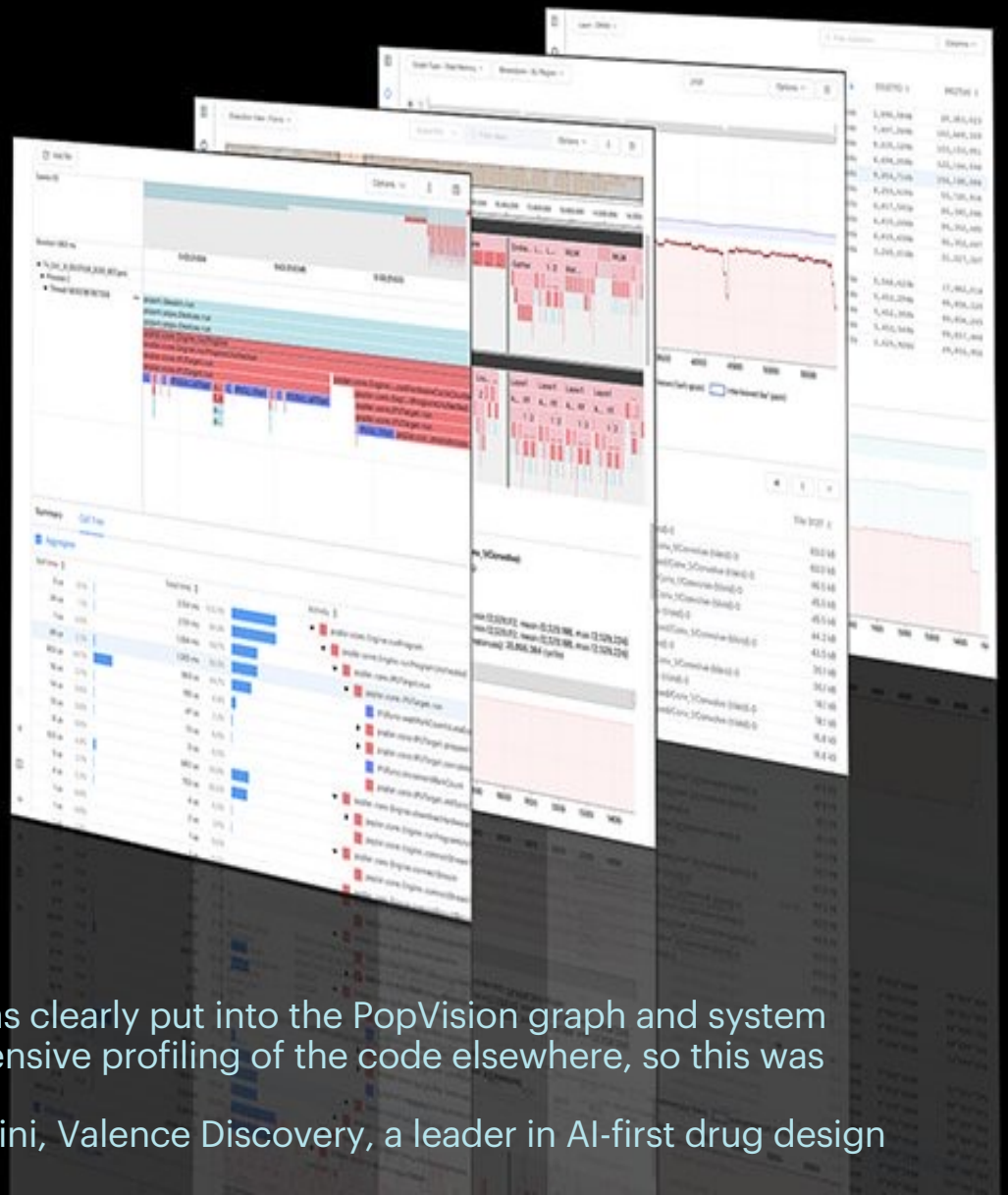
POPVISION™ TOOLS

GRAPH ANALYSER

Useful for analysing and optimising the memory use and execution performance of ML models on the IPU

SYSTEM ANALYSER

Graphical view of the timeline of host-side application execution steps



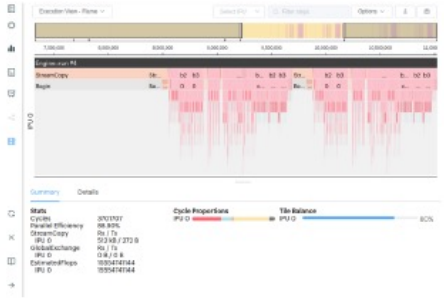
"Our team was very impressed by the care and effort Graphcore has clearly put into the PopVision graph and system analysers. It's hard to imagine getting such a helpful and comprehensive profiling of the code elsewhere, so this was really a standout feature in our IPU experience."

Dominique Beaini, Valence Discovery, a leader in AI-first drug design

POPVISION TOOLS

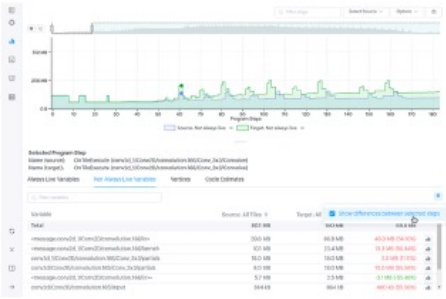
IPU MEMORY ANALYSIS

Capture memory information from your ML models when executed on IPUs. Inspect variable placement, size and liveness throughout the execution.



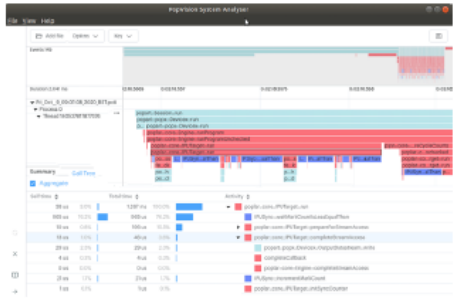
EXECUTION TRACE REPORT

View the output of instrumenting a Poplar program, capturing cycle counts for each step. See execution statistics, tile balance, cycle proportions and compute-set details.



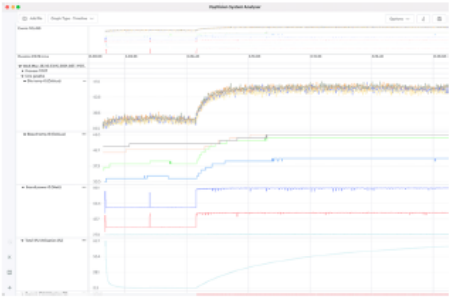
REPORT COMPARISONS

Open two reports at once to compare their memory, execution, liveness and operations. Visualise where efficiencies can be made with different model parameters.



HOST EXECUTION ANALYSIS

Understand the execution of IPU-targeted software on your host system processors. Identify any bottlenecks between CPUs and IPUs across a visual interactive timeline.



GRAPH DATA

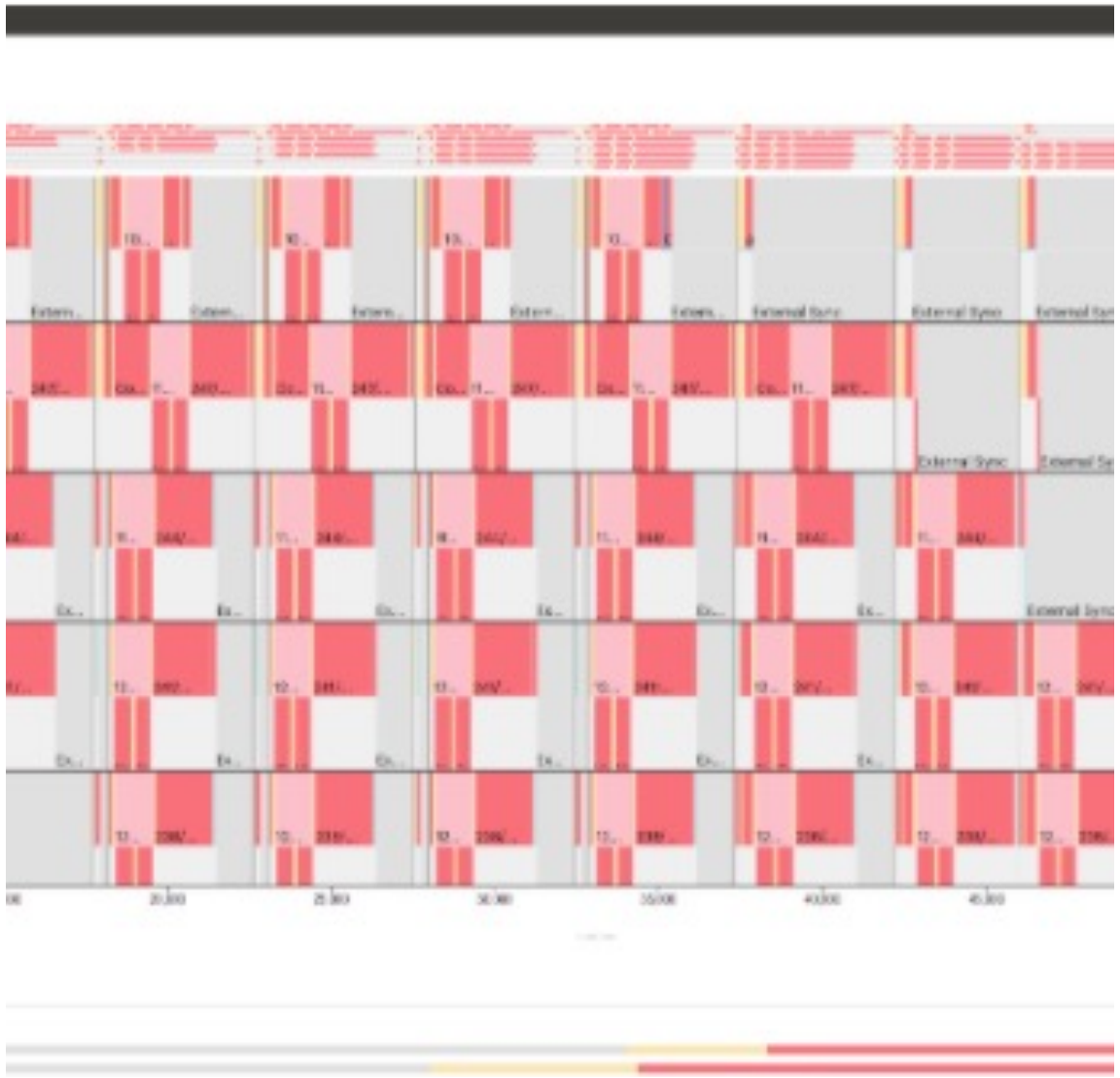
Plot graph data of any numerical data points from the host or IPU processor systems, such as board temperature, power consumption and IPU utilisation.

LOCAL + REMOTE REPORTS

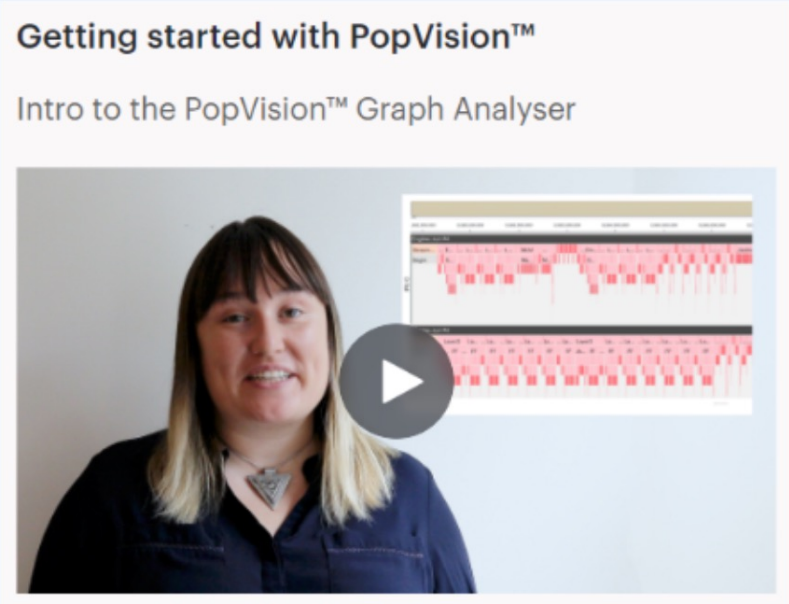
Ability to open reports either on your local machine, or remotely on the host machine. The Graph Analyser also supports local and remote report access.

POPVISION GRAPH ANALYSER

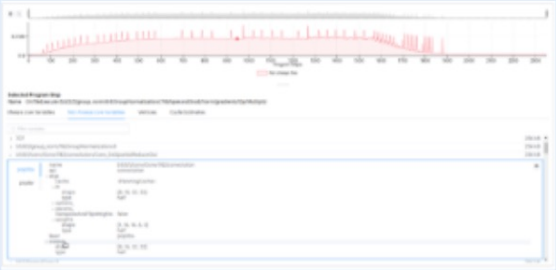
- You can use the PopVision Graph Analyser tool to debug IPU programs and generate reports on compilation and execution of the program.
- This tool can be downloaded from graphcore.ai/downloads
- There is a built-in help system within the tool for any questions you might have about producing and analysing reports.



PopVision Graph Analyser



Getting started video available on the developers portal



Several new features including:

- A new file format for the graph and execution profile, resulting in a 50% file size reduction
- Enhanced PopLibs debug information

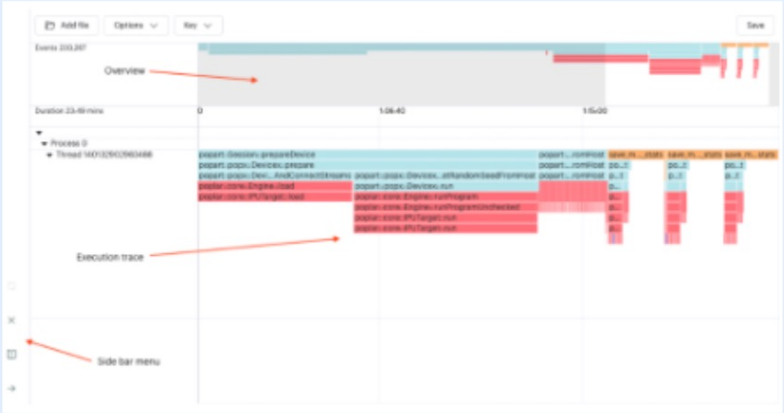
Liveness Report

The debug information shown for a variable now displays enhanced information. For each variable that has debug information, you can now see the PopLibs API that created it, its arguments and its outputs.

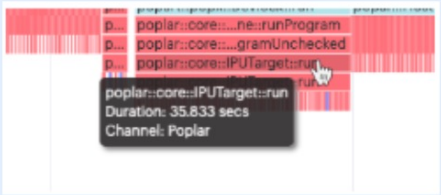
Enhanced debug information has been added to program steps. Program steps show Poplar and PopLibs debug information such as which PopLibs API created that program step, its arguments and its outputs.

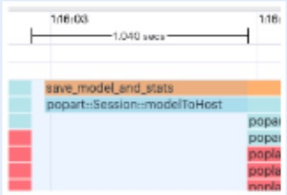
Check out the integrated help or visit our developer portal for more information

PopVision System Analyser



Show the execution of the software on the host processor enabling users to identify bottlenecks in execution between CPU & IPU(s).





Provide profile insights as you scale models to multiple CPUs / IPUs.

The PopVision System Analyser allows developers to understand the execution of programs running on the host processor which control the IPU(s). The System Analyser shows the interaction between the host and the IPU(s) so that developers can understand where the bottlenecks are in the execution of their applications.

The PopVision System Analyser visualises the information collected by the PopVision Trace Instrumentation Library which is part of the Poplar SDK.

Visit our developer portal for more information and the latest documentation:

https://www.graphcore.ai/developer

CREATE PROFILE

```
(3.3.0+1403_poptorch) alext@bpod16:~/work/examples/tutorials/tutorials/pytorch/basics$ POPLAR_ENGINE_OPTIONS='{"autoReport.all":"true", "autoReport.directory":"./
report", "debug.allowOutOfMemory":"true", "profiler.includeFlopEstimates":"true"}' python walkthrough.py
epochs:   0%|                                                                                                                    | 0/5 [00:00<?, ?it/s[
22:25:58.232] [poptorch:cpp] [warning] [DISPATCHER] Type coerced from Long to Int for tensor id 12                                | 0/3750 [00:00<?, ?it/s]
Graph compilation: 100%|                                                                                                          | 100/100 [00:39<00:00]
epochs: 100%|                                                                                                                     | 5/5 [01:21<00:00, 16.21s/it]
/nethome/alext/venvs/poplar_sdk-ubuntu_20_04-3.3.0+1403-208993bbb7/3.3.0+1403_poptorch/lib/python3.8/site-packages/torch/nn/modules/module.py:1802: UserWarning: P
ositional args are being deprecated, use kwargs instead. Refer to https://pytorch.org/docs/master/generated/torch.nn.Module.html#torch.nn.Module.state_dict for de
tails.
  warnings.warn(
Graph compilation: 100%|                                                                                                          | 100/100 [00:19<00:00]
Eval accuracy: 89.25%
Graph compilation: 100%|                                                                                                          | 100/100 [00:09<00:00]
tensor([[-4.4754, -0.9084, -3.2791, -5.8905, -3.4738, -1.4554, -1.8643, -6.6169,
         -2.1170, -5.5300]])
IPU predicted class: Trouser
CPU predicted class: Trouser
```

PopVision® Graph Analyser [10.129.111.38 : /nethome/alext/work/examples/tutorials/tutorials/pytorch/basics/report/training]

- Summary
- Insights
- Memory Report
- Liveness Report
- Program Tree
- Operations Summary
- Operations Graph
- Execution Trace
- Reload Report
- Close Report
- Documentation
- Minimise

Graph Type - Total Memory | Breakdown - None | 201 | Options

Navigation 100%

Details | Compute Sets | Vertices | Exchanges | Variables

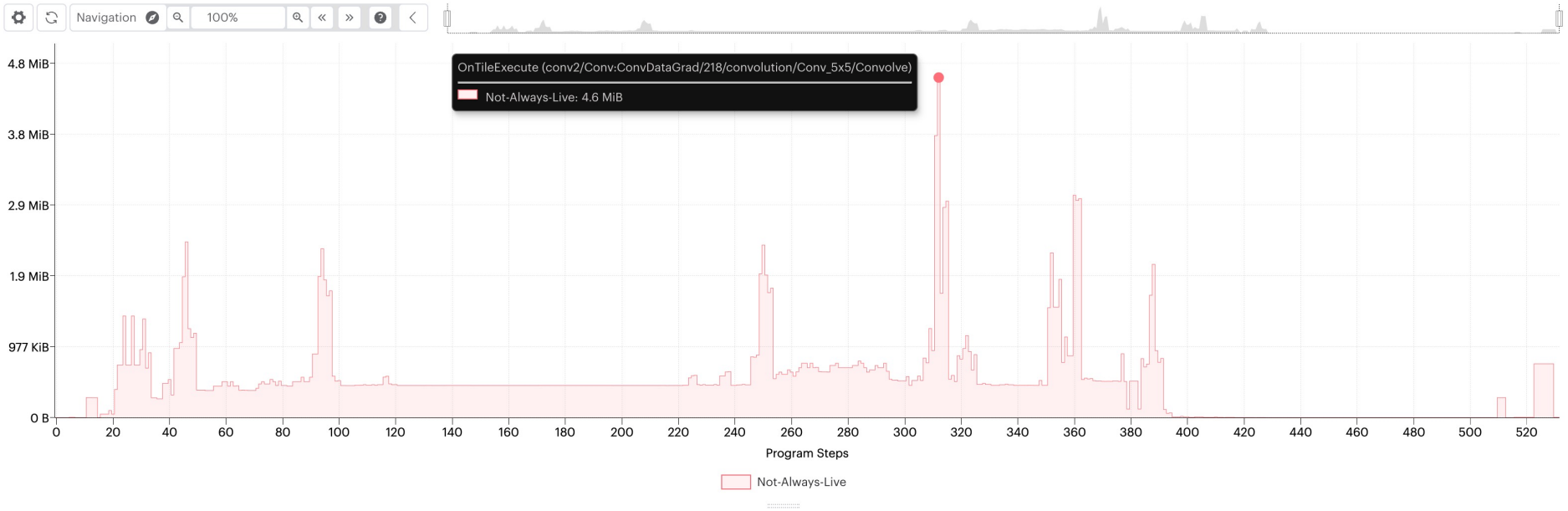
	Tile 201
Physical Tile Id	522
Memory Including Gaps	209 KiB
Memory Excluding Gaps	28.3 KiB
By Memory Region	
Non-interleaved	27.4 KiB
Interleaved	960 B
Overflowed	0 B
By Data Type	
Not Overlapped	
Variables	1.6 KiB
Constants	414 B
Control Code	2.4 KiB
Vertex Code	8.8 KiB
Internal Exchange Code	7.2 KiB
Host Exchange Code	640 B
Global Exchange Code	0 B
Vertex Instances	1.4 KiB

PopVision® Graph Analyser [10.129.111.38 : /nethome/alext/work/examples/tutorials/tutorials/pytorch/basics/report/training]

- Summary
- Insights
- Memory Report
- Liveness Report
- Program Tree
- Operations Summary
- Operations Graph
- Execution Trace
- Reload Report
- Close Report
- Documentation
- Minimise

Filter steps | Select Source | Options

Navigation 100%



Selected Program Steps

Step 312 OnTileExecute (conv2/Conv:ConvDataGrad/218/convolution/Conv_5x5/Convolve)

Always-Live Variables | Not-Always-Live Variables | Vertices | Cycle Estimates | FLOP Estimates

Filter variables

Variable	All Tiles
Total	**29.3 MiB**
vertexCode	11.8 MiB
internalExchangeCode	8.5 MiB
controlCode	3.0 MiB
vertexInstanceState	1.9 MiB
hostExchangeCode	924 KiB
stack	535 KiB
instrumentationResults	402 KiB
Accl1__fc1.weight	380 KiB
fc1.weight	380 KiB
vertexFieldData	241 KiB
copyDescriptor	223 KiB
hostExchangePacketHeader	198 KiB

PopVision® Graph Analyser [10.129.111.38 : /nethome/alext/work/examples/tutorials/tutorials/pytorch/basics/report/training]

- Summary
- Insights
- Memory Report
- Liveness Report
- Program Tree
- Operations Summary
- Operations Graph
- Execution Trace
- Reload Report
- Close Report
- Documentation
- Minimise

Execution View - Flame & BSP | Navigation | 58957% | All IPUs | Search | Options

480,000 | 1,490,000 | 1,500,000 | 1,510,000 | 1,520,000 | 1,530,000 | 1,540,000 | 1,550,000 | 1,560,000 | 1,570,000

IPU 0

Engine::run #0

StreamCopy | con...x3 | p... | co...-0 | c...5 | norm/G...n/107 | fc1/Gemm | fc...mm | log...ax | loss/NllloSS | log_s.../138 | fc2 | f...1 | fc1...v_1 | norm/Group...zation/180 | conv2/...v_5x5 | poo...ol | 1...d | 1... | fc1 | con...tas | _... | co...as | StreamCopy

End | gr...cs | gr...rm | 110/...nv_1 | 116..._1 | 121 | 136 | no...ax | w... | operandGrad | s...s | Co...ve | P...0 | G... | C...e | Begin

no...d | W... | NonDi...ients | J...

N... | N...s

J...

100%

Summary | Details

Total Cycles 94,526 Total FLOPs 31,844,509

IPU	Proportions	Tile Utilisation	Stream Copy (Rx/Tx)	Global Exchange (Rx/Tx)
0		59.16%	0 B / 0 B	0 B / 0 B

One or more step types are hidden. These step types are excluded from the tile balance, tile utilisation, and cycle proportions shown and so the statistics may not be representative. Step types can be shown or hidden using the view options dropdown.

POPVISION TUTORIALS

Continued in the repositories below (follow the READMEs)

tutorials/tensorflow2/infeed_outfeed

tutorials/pytorch/pipelining

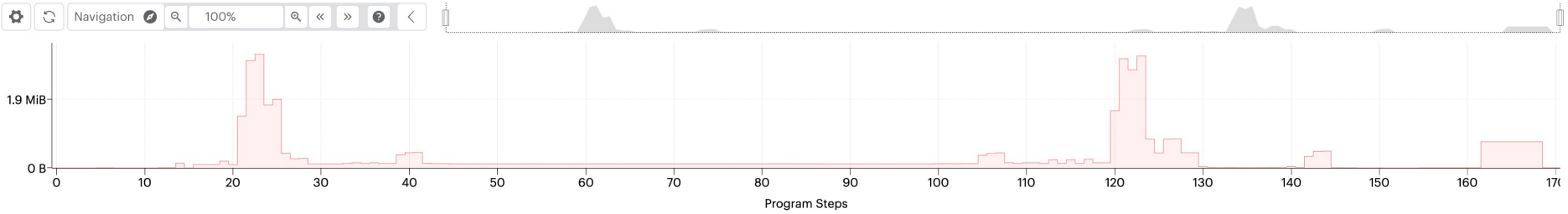
tutorials/popvision/system_analyser_instrumentation

PopVision® Graph Analyser [10.129.111.38 : /nethome/alext/work/examples/tutorials/tutorials/tensorflow2/infeed_outfeed/completed_code/report/tf_report__2023-07-25__01-03-33.553__93433 vs 10.129.111.38 : /nethome/alext/work/examples/tutorials/tutorials/tensor...

- Summary
- Memory Report
- Liveness Report
- Program Tree
- Operations Summary
- Execution Trace
- Swap Reports
- Reload Report
- Close Report
- Documentation
- Minimise

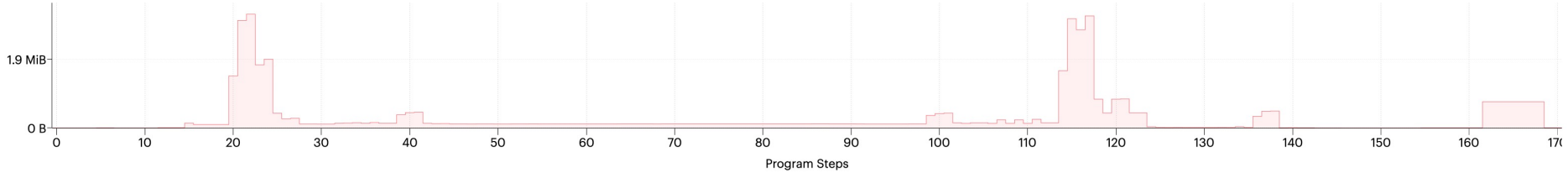
Filter steps | Select Source | Options

Navigation 100%



Source: Not-Always-Live

Navigation 100%



Target: Not-Always-Live

Selected Program Step

Name (source): None selected. Click on the graph to select a source program step

Name (target): None selected. Click on the graph to select a target program step

Always-Live Variables | Not-Always-Live Variables | Vertices | Cycle Estimates

Filter variables

Variable	Source: All Tiles	Target: All Tiles	Diff: All Tiles
Total	**11.3 MiB**	**11.2 MiB**	**-113 KiB**
vertexCode	6.0 MiB	6.0 MiB	-16 B (-0.00%)
controlCode	1.5 MiB	1.5 MiB	-37.7 KiB (-2.41%)
internalExchangeCode	1.2 MiB	1.1 MiB	-47.6 KiB (-3.99%)
hostExchangeCode	595 KiB	556 KiB	-39.2 KiB (-6.58%)
stack	540 KiB	529 KiB	-11.0 KiB (-2.04%)
vertexInstanceState	527 KiB	527 KiB	20 B (0.00%)
while/sequential/dense/MatMul/dot.35/rhs	392 KiB	—	-392 KiB (-100.00%)
sequential/dense/MatMul/dot/rhs	—	392 KiB	392 KiB (100.00%)
instrumentationResults	180 KiB	180 KiB	4 B (0.00%)
hostExchangePacketHeader	114 KiB	137 KiB	23.7 KiB (17.25%)

PopVision® Graph Analyser [10.129.111.38 : /nethome/alext/work/examples/tutorials/tutorials/pytorch/pipelining/report/training]

- Summary
- Insights
- Memory Report
- Liveness Report
- Program Tree
- Operations Summary
- Operations Graph
- Execution Trace
- Reload Report
- Close Report
- Documentation
- Minimise

Execution View - Flame | Navigation | 260775% | All IPUs | Search | Options

2,600,000 2,800,000 3,000,000 3,200,000 3,400,000 3,600,000 3,800,000 4,000,000 4,200,000

IPU 0

Engine::run #0

St...y StreamCopy StreamCopy StreamCopy StreamCopy StreamCopy Ex...c StreamCopy StreamCopy StreamCopy Stre...Copy E...

Mid Mid Mid Mid Mid Mid Mid Mid Mid Mid

IPU 1

Engine::run #0

St...y StreamCopy StreamCopy StreamCopy StreamCopy StreamCopy StreamCopy Stre...Copy StreamCopy Stre...Copy E...

End End End End End End End End End End

IPU 2

Engine::run #0

St...y StreamCopy StreamCopy StreamCopy StreamCopy StreamCopy Ex...c StreamCopy Ex...c Stre...Copy Ex...c StreamCopy Ex...c Stre...Copy Ex...c Ex...c E... E... E... Ex...c E...

End End End End End End End End End End

IPU 3

Engine::run #0

St...y StreamCopy StreamCopy StreamCopy StreamCopy StreamCopy E...c StreamCopy E...c StreamCopy E...c StreamCopy E...c StreamCopy E...c E...c S... Ext...nc Ext...nc E...c

End End End B... End End End End End End End End

Summary Details

Total Cycles 1,700,636 Total FLOPs 2,227,345,013

IPU	Proportions	Tile Utilisation	Stream Copy (Rx/Tx)	Global Exchange (Rx/Tx)
0		56.02%	1.1 MiB / 172 KiB	4.1 MiB / 4.2 MiB
1		54.73%	0 B / 0 B	5.0 MiB / 5.3 MiB
2		26.01%	0 B / 0 B	1.8 MiB / 1.4 MiB
3		18.01%	768 B / 320 B	640 KiB / 640 KiB

PopVision® System Analyser

Add file | Flame Graph Type - Aggregated | Line Graph Type - Line | Heatmap Options | Navigation | 100% | Search | Options

Events 405
Duration 17.561 secs

Collapse all charts

00:00:00 00:00:02 00:00:04 00:00:06 00:00:08 00:00:10 00:00:12 00:00:14 00:00:16

Tue_Jul_25_00:06:49_2023_UTC_57...

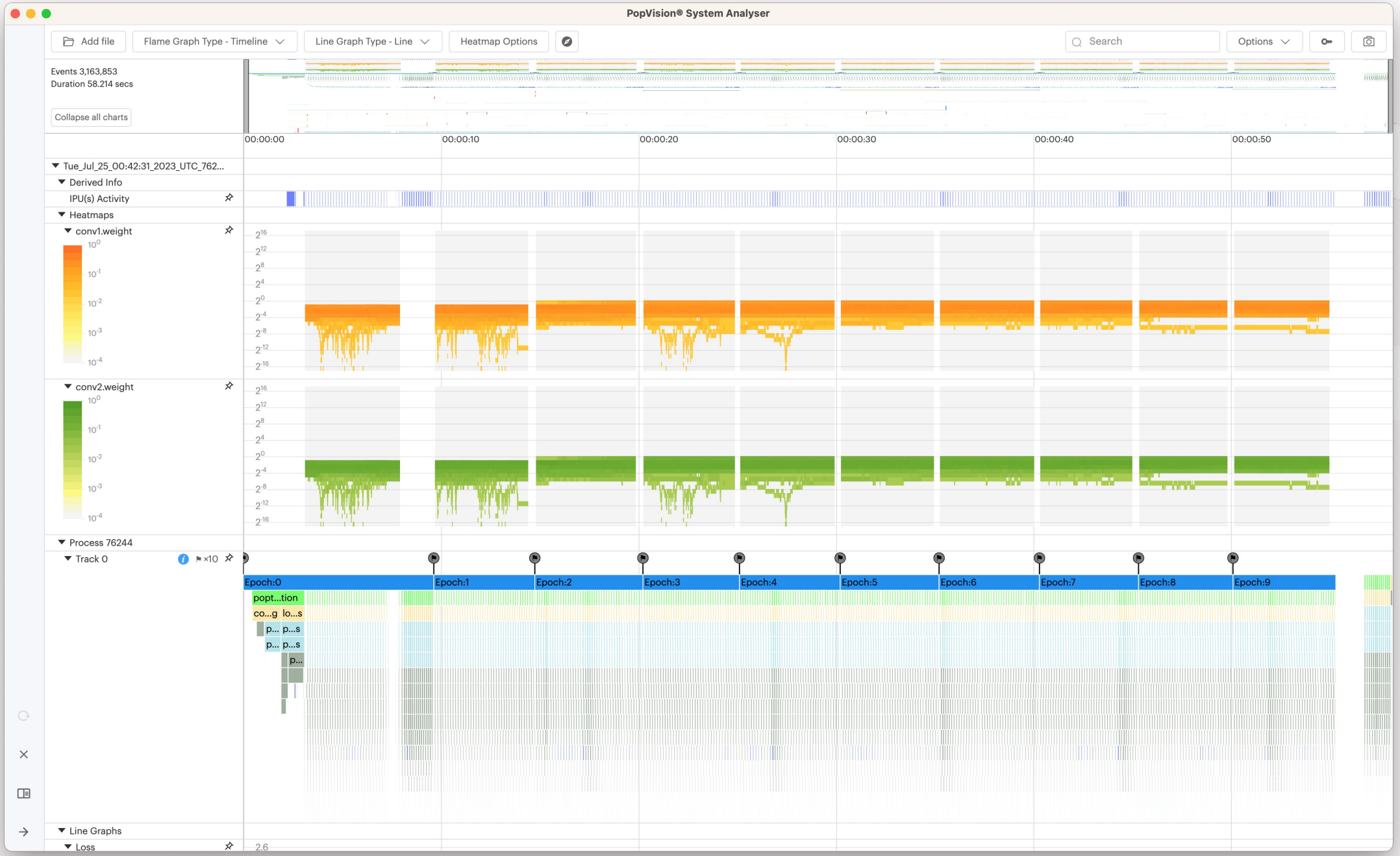
Aggregated

po...ad
pop...ch
pop...ack p...
poplar::cor...::operator po...r
IPUSync...alThan poplar::core:...repareAccess pop...ss IP...t
poplar::co...aitForAny poplar::core::IPUTarget::processHspSync po...h
poplar::core::IPUTarget::run
p... poplar::core::IPUTarget::run
P...e poplar::core::Engine::runProgramUnchecked
p...e pop...oad poplar::core::Engine::runProgram
p...e popl...oad popart::popx::Devicex::run
popa...ble popart:...treams popart::popx::Devicex::run
p...h popa...ice popart:...treams popart::Session::run
compileWi...ITracing loadEn...treams execute
poptorch.training...modelCompilation poptorch.trainingModel.modelExecution
popl...ach popar...vice popart::Ses...ectStreams
popar...able popart::pop...ectStreams
compileWith...nualTracing loadEngineA...ectStreams
poptorch.inferenceModel.modelCompilation
po...et
Prep...ble
popl...are poplar...::load
popl...are poplar:...::load
p...
po...n
po...n
po...d
p...m
po...n
po...n
po...n
ex...te
po...on

Summary | Call Tree

Aggregate

Self time		Total time		Activity
3.421 secs	38.0%	8.996 secs	100.0%	poplar::core::IPUTarget::run
565.996 ms	6.3%	3.615 secs	40.2%	poplar::core::IPUTarget::processHspSync
366.950 ms	4.1%	1.945 secs	21.6%	poplar::core::SyncGroupFlow::prepareAccess
352.631 ms	3.9%	1.578 secs	17.5%	poplar::core::Fetch::operator
58.174 ms	0.6%	658.362 ms	7.3%	poplar::core::Fetch::doCallback
43.665 ms	0.5%	600.188 ms	6.7%	popart::popx::Devicex::PrefetchCallback::fetch
556.523 ms	6.2%	556.523 ms	6.2%	popart::popx::Devicex::InputDatastream::read
306.832 ms	3.4%	306.832 ms	3.4%	poplar::core::Fetch::doHostGatewaySync
192.610 ms	2.1%	192.610 ms	2.1%	poplar::core::Fetch::doHostGatewayMirror
36.681 ms	0.4%	67.629 ms	0.8%	popart::popx::Devicex::PrefetchCallback::complete
147.590 ms	1.6%	599.345 ms	6.7%	poplar::core::SyncGroupFlow::completeAccess
331.532 ms	3.7%	331.532 ms	3.7%	IPUSync::incrementMarkCount
96.082 ms	1.1%	173.335 ms	1.9%	poplar::core::SyncGroupFlow::releaseAccess
313.409 ms	3.5%	1.503 secs	16.7%	poplar::core::IPUDevice::waitForAny


PopVision® System Analyser
Add file
Flame Graph Type - Timeline
Line Graph Type - Line
Heatmap Options
Search
Options
Events 3,163,853
Duration 58.214 secs
Collapse all charts
00:00:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:50
Tue_Jul_25_00:42:31_2023_UTC_762...
Derived Info
IPU(s) Activity
Heatmaps
conv1.weight
conv2.weight
Process 76244
Track 0
×10
Epoch:0
Epoch:1
Epoch:2
Epoch:3
Epoch:4
Epoch:5
Epoch:6
Epoch:7
Epoch:8
Epoch:9
popt...tion
co...g
lo...s
Line Graphs
Loss
2.6

ACCESS Allocations: Prepare

allocations.access-ci.org/prepare-requests-overview

ALLOCATIONS SUPPORT OPERATIONS METRICS Login

ACCESS Allocations

Request an IPU allocation for your research today!

allocations.access-ci.org/prepare-requests-overview

Get started | Manage allocations | Prepare requests | Use credits | Updates | Policies | FAQs

Prepare requests: overview

We've organized your options for requesting access to advanced research computing resources into a set of opportunities designed to support needs ranging from new or entry-level exploration to the largest-scale computational experiments. We welcome you to find the opportunity that aligns with your best estimate of your resource needs. And don't worry about starting too small. As you clarify your needs, you can upgrade to a larger-scale opportunity when you're ready.

There are four opportunities, which are described below and **compared side-by-side in this table**:

- Explore ACCESS
- Discover ACCESS

Prepare requests: overview

- Explore ACCESS
- Discover ACCESS
- Accelerate ACCESS
- Maximize ACCESS
- Comparison Table

THANK YOU

Alexander Tsyplikhin
alext@graphcore.ai

Request an IPU allocation for your research today!

allocations.access-ci.org/prepare-requests-overview