GRAPHCORE

SC23
Denver, CO | i am hpc.

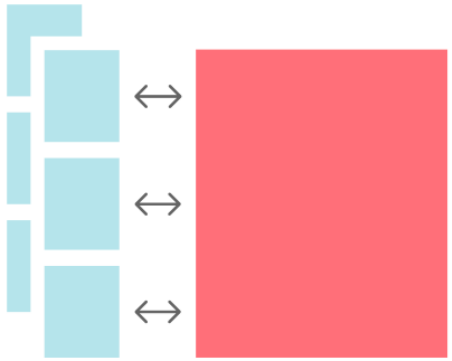
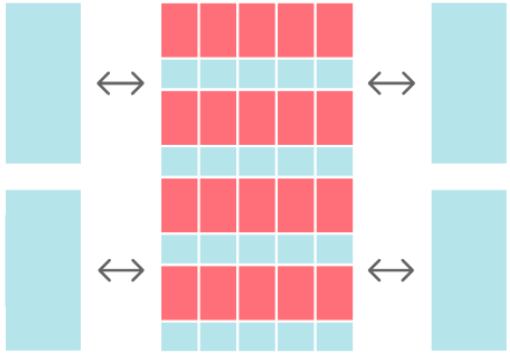
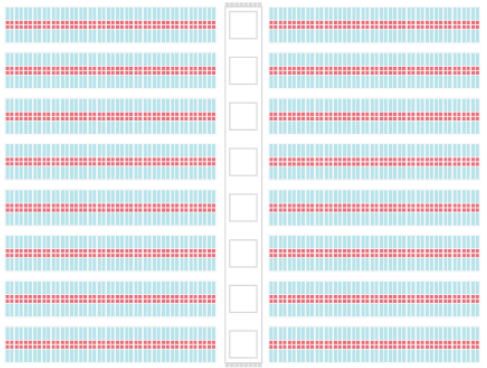
Programming IPUs for Scientific Computing

Alexander Tsyplikhin, Field AI Engineering

Tutorial with Argonne National Laboratory

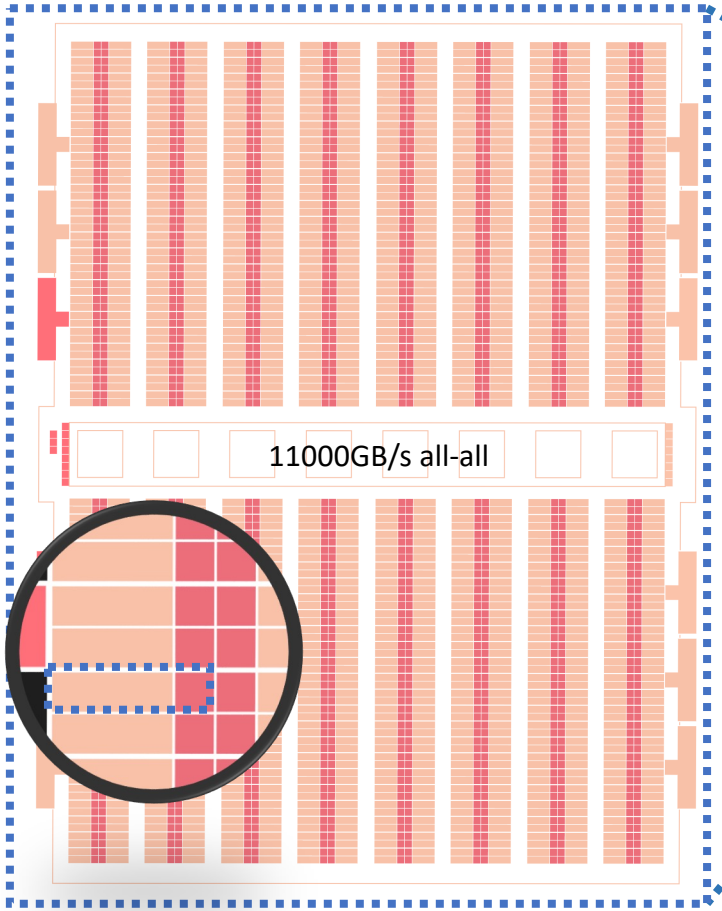
IPU – Architectured For AI

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
			
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

BOW IPU

350TFLOPS(F16)
900MB SRAM
1472 IPU-Tiles
8832 independent instruction streams

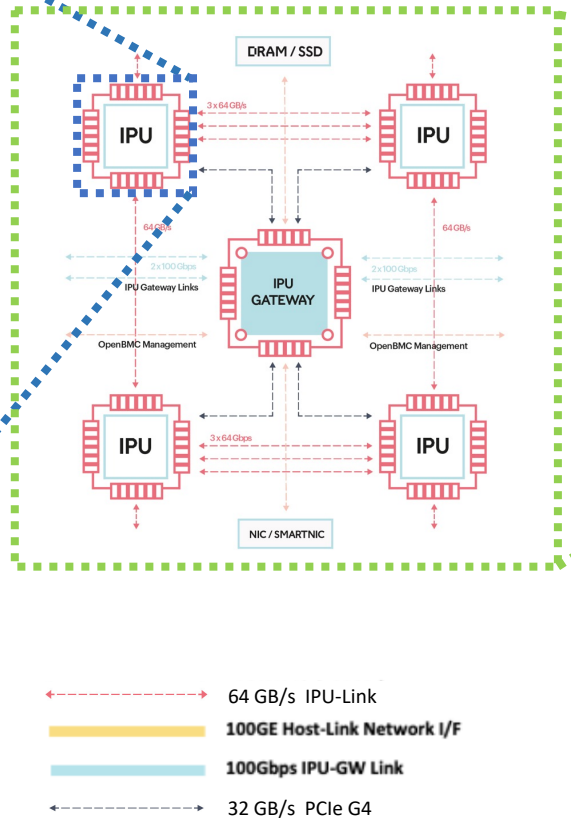


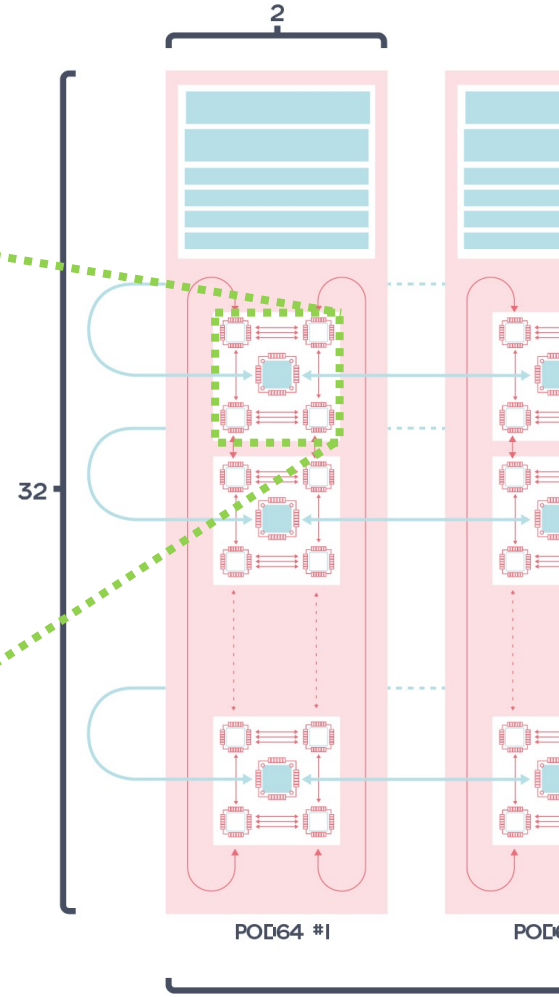
IPU-Tile

236GFLOPS(F16)
640KB SRAM
6 HW Threads
1.83GHz
128 F16Ops/Cycle

BOW-2000 IPU-Machine

4x IPU
256 GB DRAM





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

POPLAR®
PARTITIONER
POPIR
POPIT
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

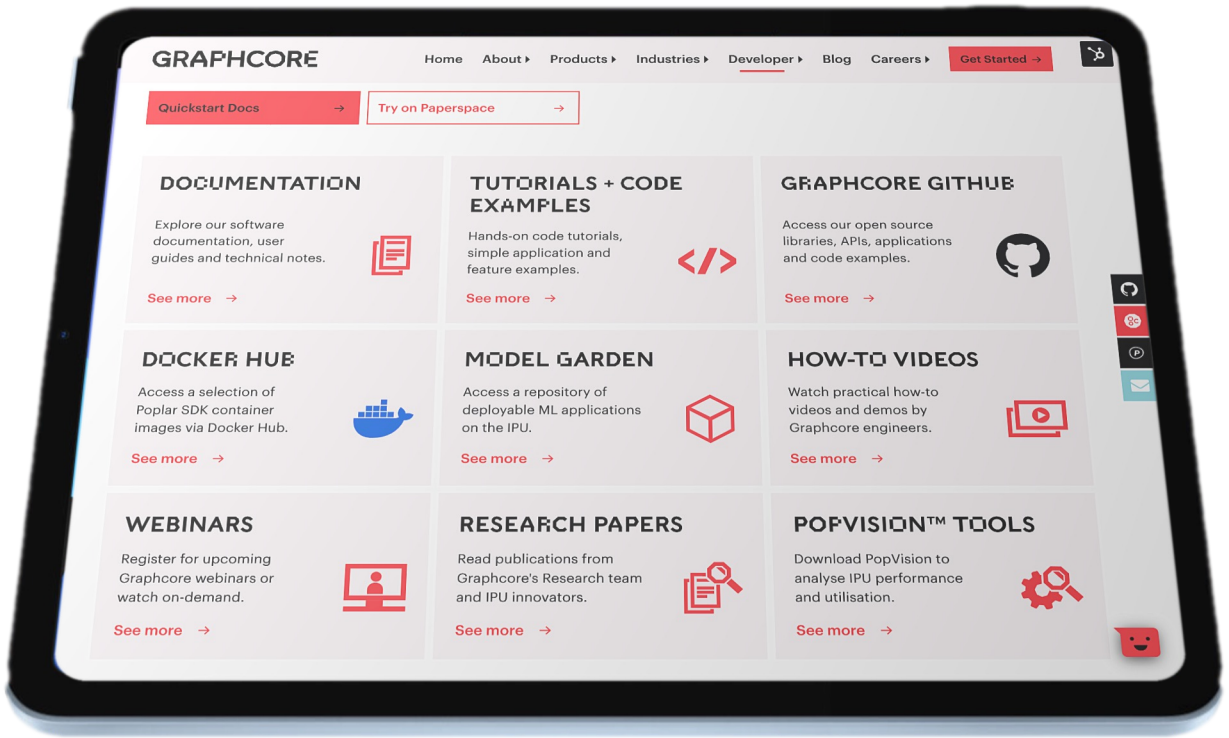
SYSTEM MONITORING
PROMETHEUS
GRAFANA

JOB DEPLOYMENT
K8S
SLURM

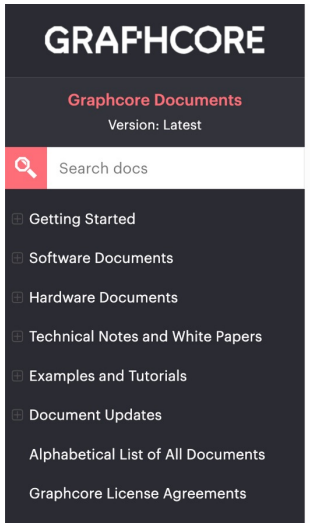
SYSTEM SOFTWARE

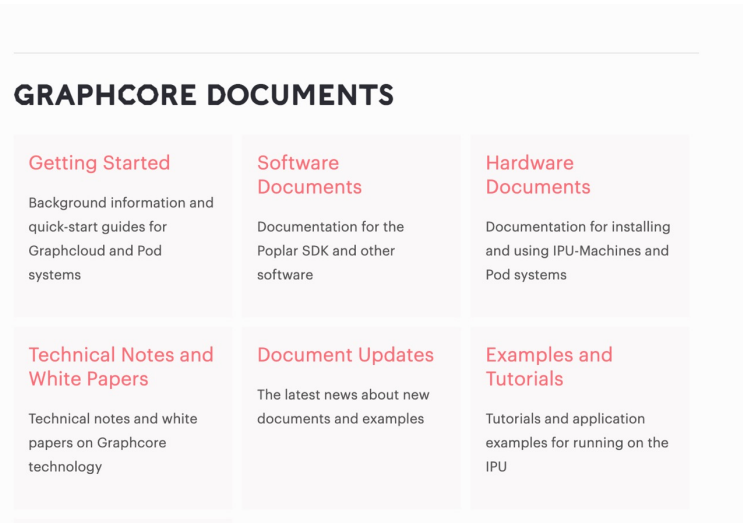
GRAPHCORE SOFTWARE ECOSYSTEM

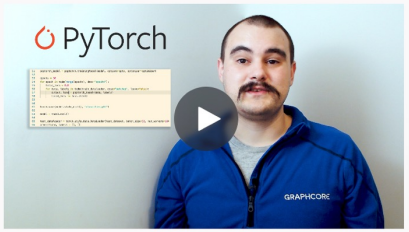
WORLD CLASS DEVELOPER RESOURCES FOR IPU USERS



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.

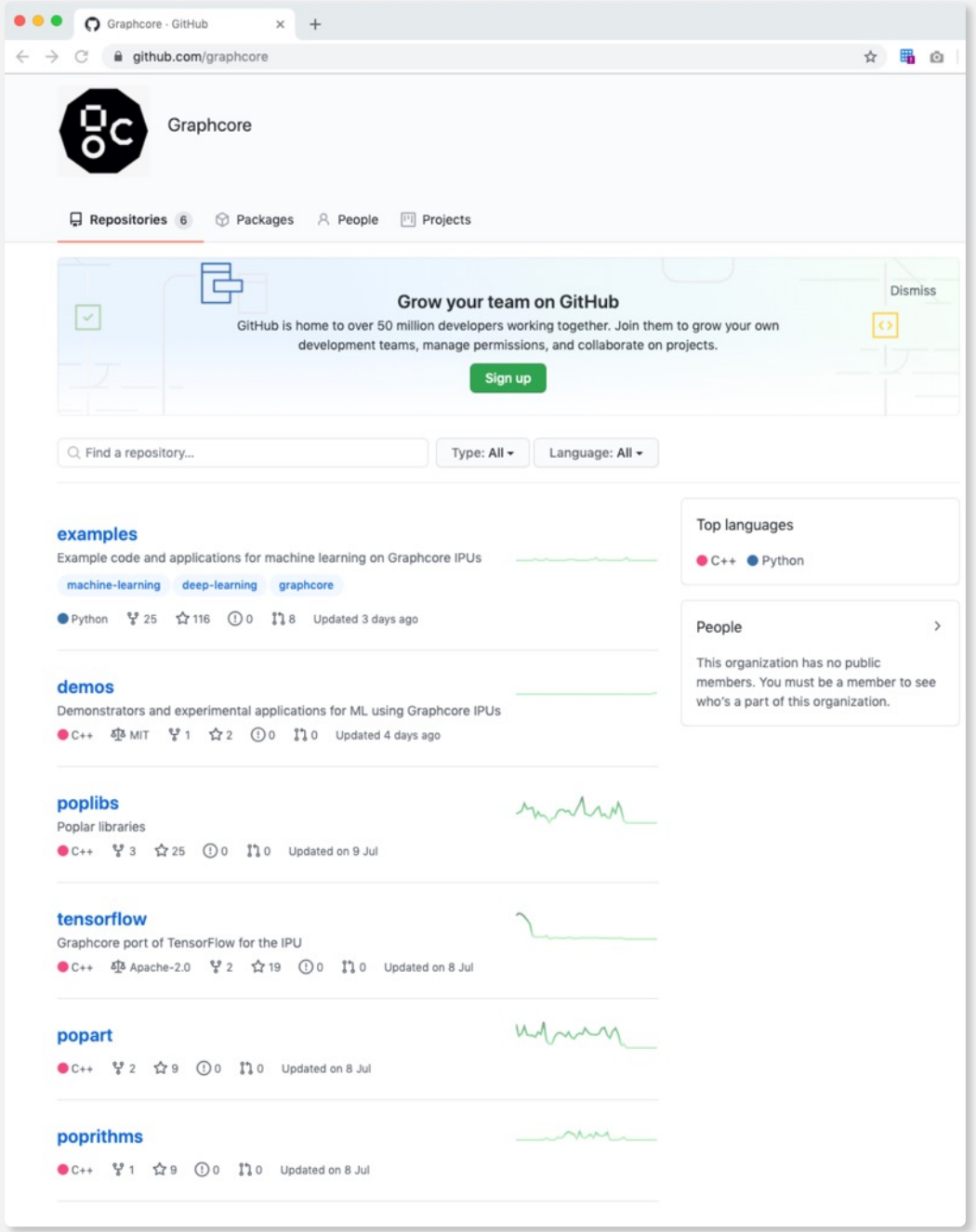
Get the Code →

Read the Guide →

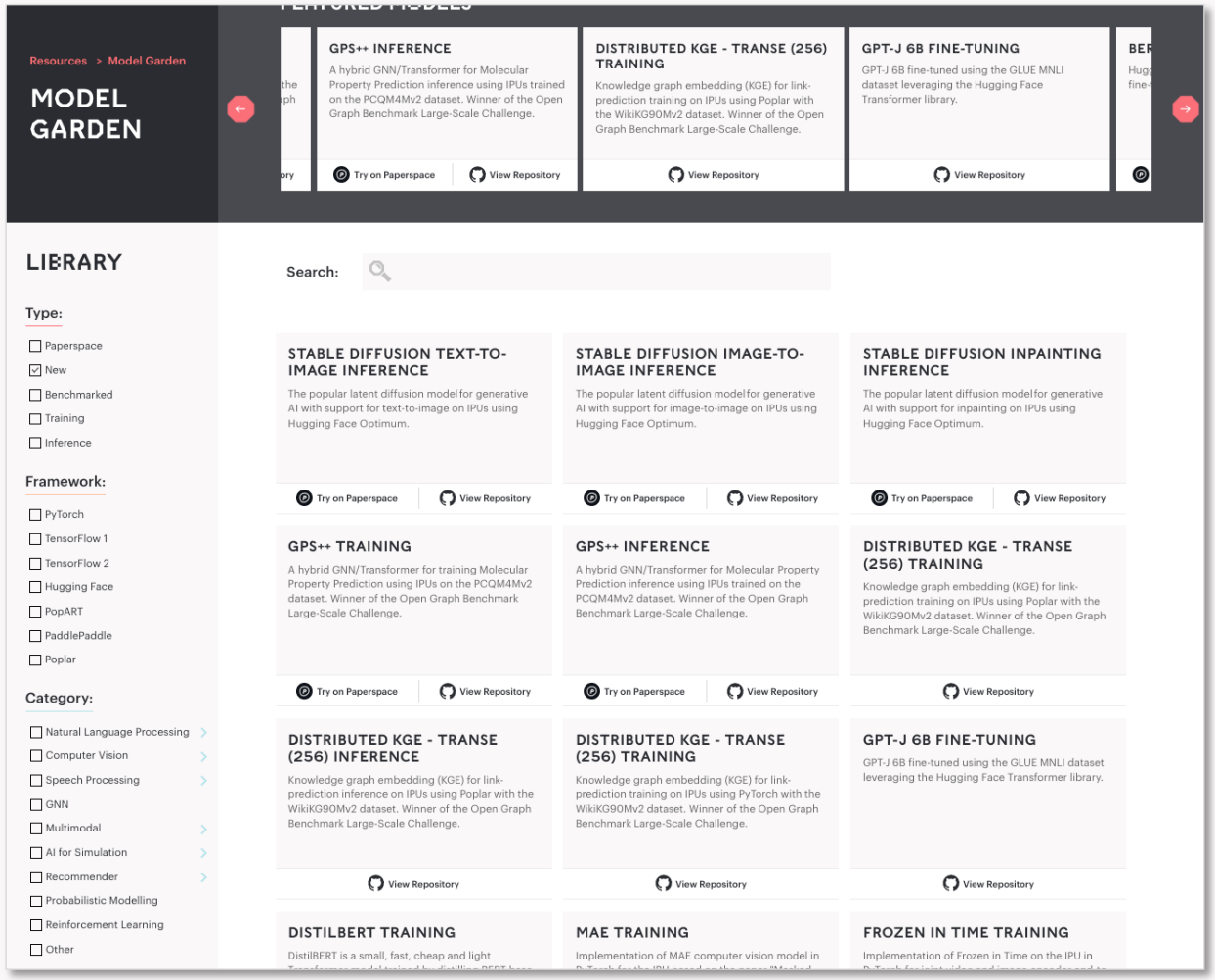
OPEN SOURCE

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



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

PyTorch

GPU

```
    _, 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.size(0) * 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)'
    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.size(0) * 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)))
```


gpu_cnn_keras.py ↔ ipu_cnn_keras.py tf_keras
Keras
GPU
IPU
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)
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)

CODELET DEFINITIONS

Each codelet is defined as a C++ class that inherits from the **Vertex** class.

The fields of the vertex specify its inputs, outputs and internal data.

```
class AdderVertex : public Vertex {
public:
 Input<float> x;
 Input<float> y;
 Output<float> z;
 float bias;

 bool compute() {
   *z = x + y + bias;
   return true;
 }
}
```

The compute method specifies the vertex execution behaviour.

BULK SYNCHRONOUS PARALLEL (BSP)

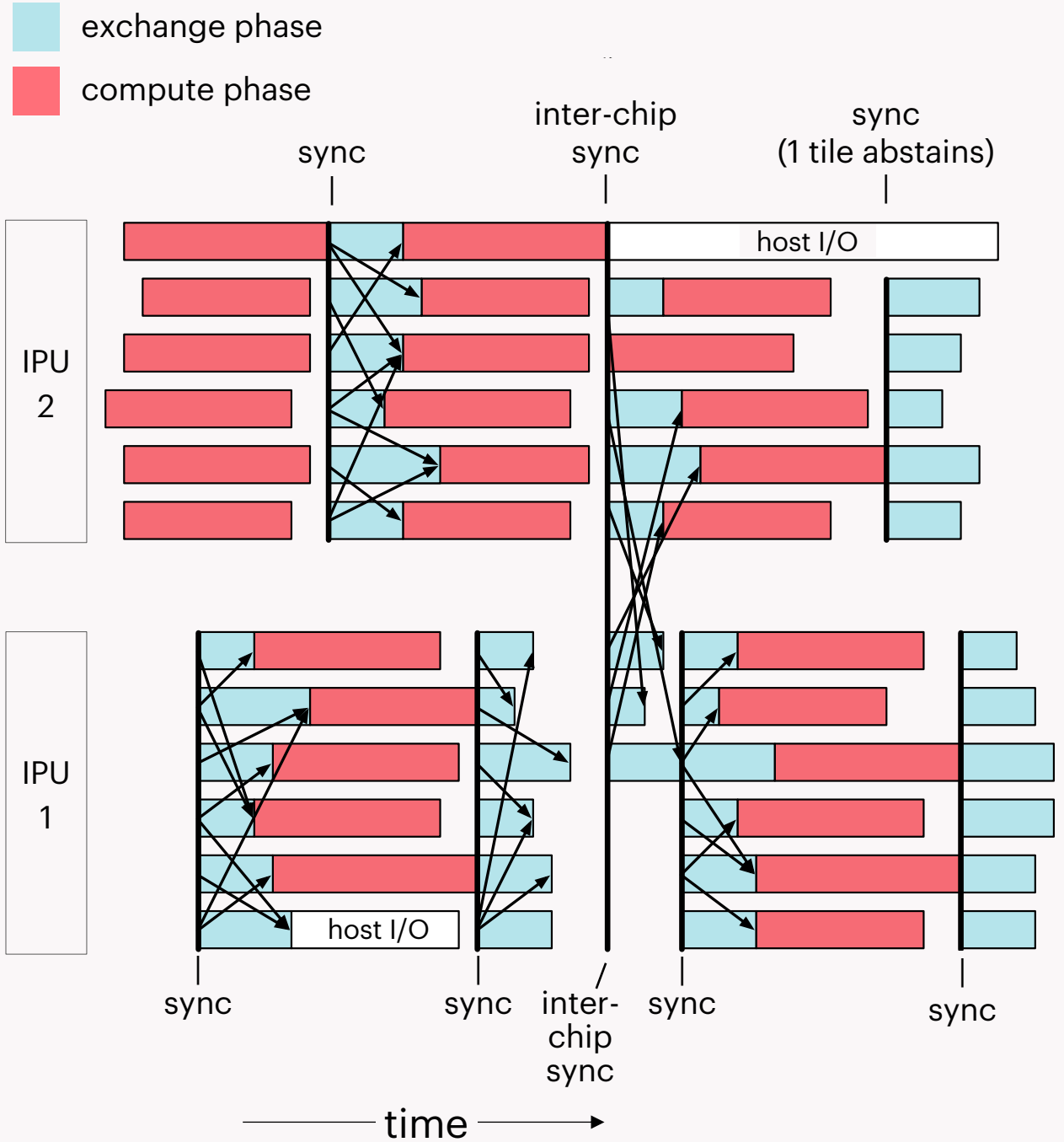
BSP software bridging model – massively parallel computing with no concurrency hazards

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



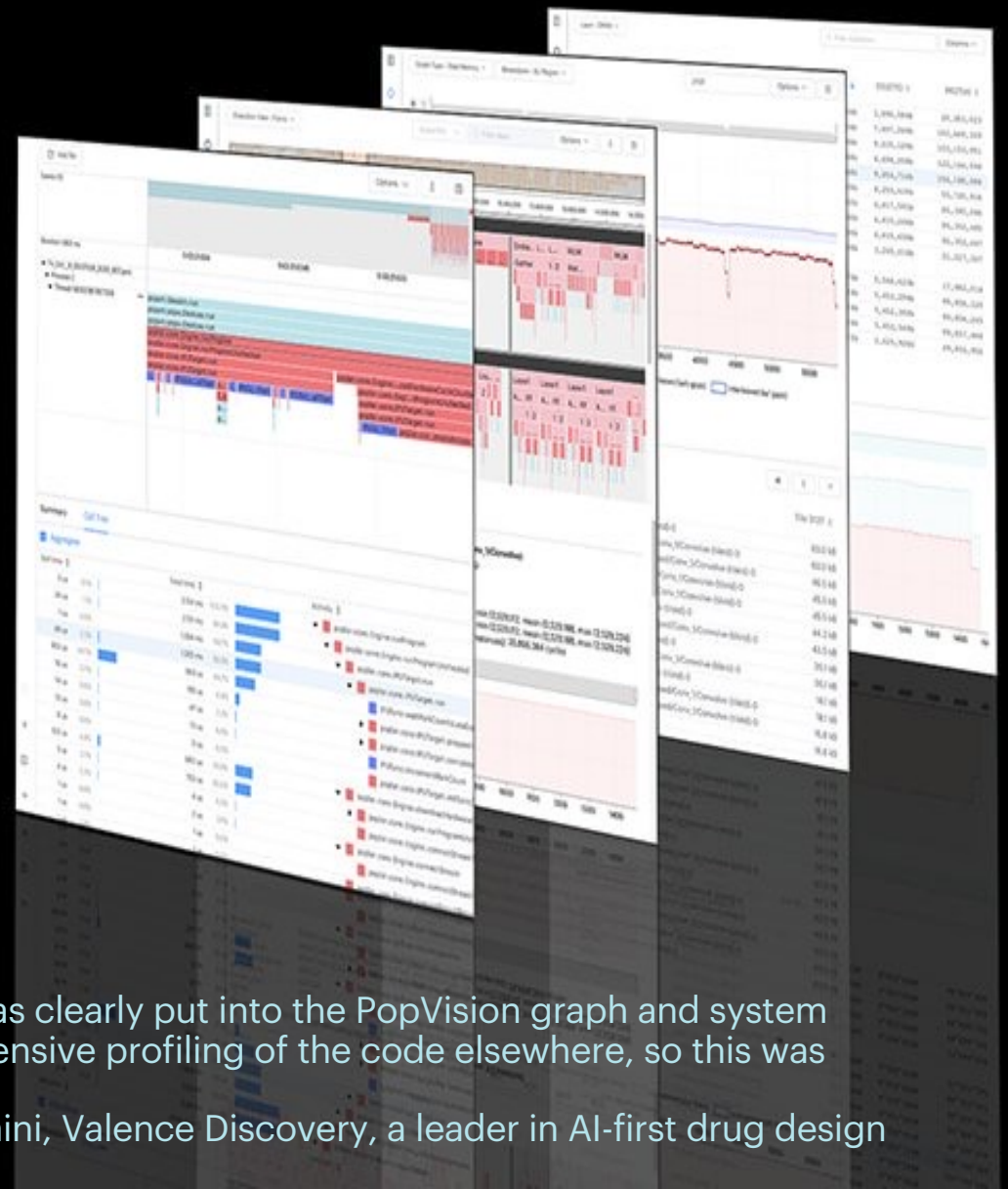
POPVISION™ TOOLS

GRAPH ANALYSER

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

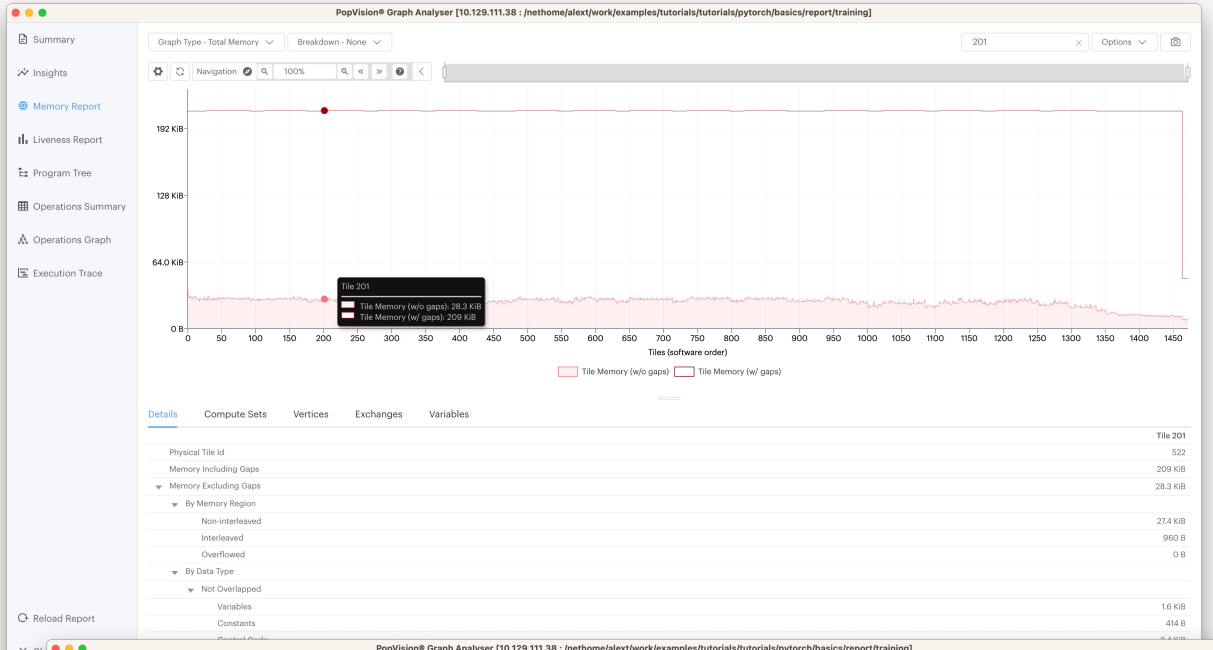
SYSTEM ANALYSER

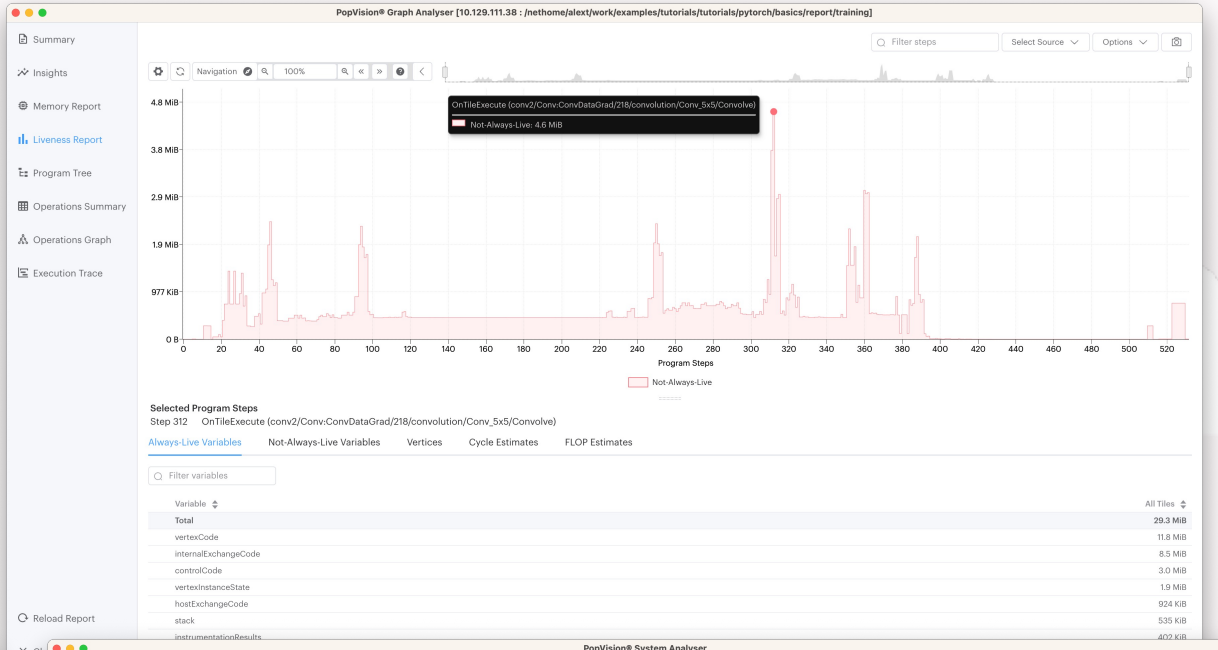
Graphical view of the timeline of host-side application execution steps



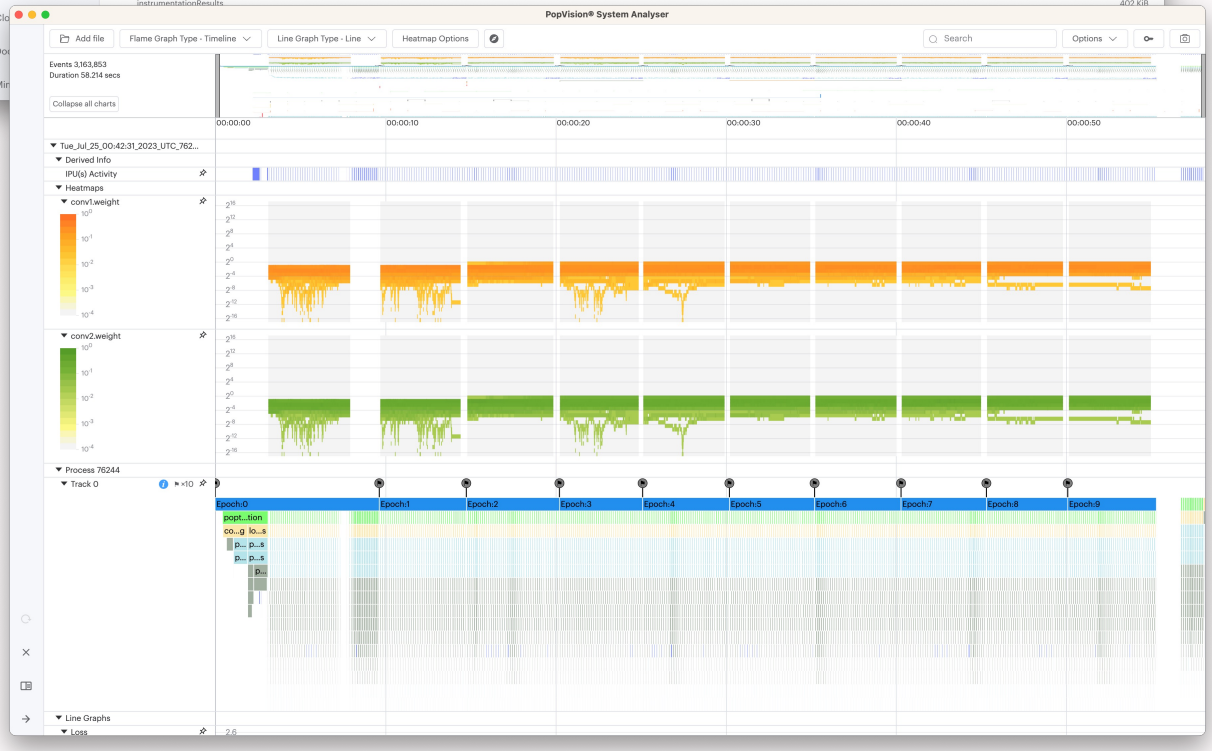
"Our team was very impressed by the care and effort Graphcore has clearly put into the PopVision graph and system analysers. It's hard to imagine getting such a helpful and comprehensive profiling of the code elsewhere, so this was really a standout feature in our IPU experience."

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


PopVision® 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
Graph Type - Total Memory
Breakdown - None
Navigation
Tile 201
Tile Memory (w/o gaps): 28.3 KiB
Tile Memory (w/ gaps): 209 KiB
Tiles (software order)
Tile Memory (w/o gaps)
Tile Memory (w/ gaps)
Details
Compute Sets
Vertices
Exchanges
Variables
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
Reload Report
Close Report
Documentation
Minimise


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
Navigation
OnTileExecute (conv2/Conv:ConvDataGrad/218/convolution/Conv_5x5/Convolve)
Not-Always-Live: 4.6 MiB
Program Steps
Not-Always-Live
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
Reload Report


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
Execution View - Flame & BSP
Navigation
All IPUs
Search
IPU 0
Engine::run #0
Summary
Details
Total Cycles 94,526 Total FLOPs 31,844,509
IPU
Proportions
Tile Utilisation
Stream Copy (Rx/Tx)
Global Exchange (Rx/Tx)
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.
Reload Report
Close Report
Documentation
Minimise


PopVision® System Analyser
Add file
Flame Graph Type - Timeline
Line Graph Type - Line
Heatmap Options
Search
Events 3,163,853
Duration 58.214 secs
Collapse all charts
Derived Info
IPU(s) Activity
Heatmaps
conv1.weight
conv2.weight
Process 76244
Track 0
Epoch:0
Epoch:1
Epoch:2
Epoch:3
Epoch:4
Epoch:5
Epoch:6
Epoch:7
Epoch:8
Epoch:9
Line Graphs
Loss

TUTORIALS

PyTorch

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

TensorFlow/Keras

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

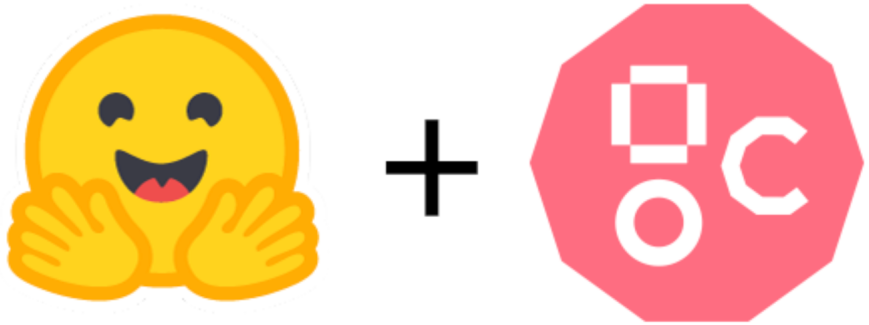
PopVision

tutorials/tensorflow2/infeed_outfeed

tutorials/pytorch/pipelining

tutorials/popvision/system_analyser_instrumentation

README.md



Optimum Graphcore

🤗 Optimum Graphcore is the interface between the 🤗 Transformers library and Graphcore IPUs. It provides a set of tools enabling model parallelization and loading on IPUs, training, fine-tuning and inference on all the tasks already supported by 🤗 Transformers while being compatible with the 🤗 Hub and every model available on it out of the box.

What is an Intelligence Processing Unit (IPU)?

Quote from the Hugging Face blog post:

> IPUs are the processors that power Graphcore's IPU-POD datacenter compute systems. This new type of processor is designed to support the very specific computational requirements of AI and machine learning. Characteristics such as fine-grained parallelism, low precision arithmetic, and the ability to handle sparsity have been built into our silicon.

> Instead of adopting a SIMD/SIMT architecture like GPUs, Graphcore's IPU uses a massively parallel, MIMD architecture, with ultra-high bandwidth memory placed adjacent to the processor cores, right on the silicon die.

> This design delivers high performance and new levels of efficiency, whether running today's most popular models, such as BERT and EfficientNet, or exploring next-generation AI applications.

README.md

How to use Optimum Graphcore

To immediately use a model on a given input (text, image, audio, ...), we support the `pipeline` API:

```
->>> from transformers import pipeline
+>>> from optimum.graphcore import pipeline

# Allocate a pipeline for sentiment-analysis
->>> classifier = pipeline('sentiment-analysis', model="distilbert-base-uncased-finetuned-sst-2-engl
+>>> classifier = pipeline('sentiment-analysis', model="distilbert-base-uncased-finetuned-sst-2-engl
>>> classifier('We are very happy to introduce pipeline to the transformers repository.')
[{'label': 'POSITIVE', 'score': 0.9996947050094604}]
```

It is also super easy to use the `Trainer` API:

```
-from transformers import Trainer, TrainingArguments
+from optimum.graphcore import IPUConfig, IPUTrainer, IPUTrainingArguments

-training_args = TrainingArguments(
+training_args = IPUTrainingArguments(
     per_device_train_batch_size=4,
     learning_rate=1e-4,
+    # Any IPUConfig on the Hub or stored locally
+    ipu_config_name="Graphcore/bert-base-ipu",
+)
+
+# Loading the IPUConfig needed by the IPUTrainer to compile and train the model on IPUs
+ipu_config = IPUConfig.from_pretrained(
+    training_args.ipu_config_name,
 )

 # Initialize our Trainer
-trainer = Trainer(
+trainer = IPUTrainer(
     model=model,
+    ipu_config=ipu_config,
     args=training_args,
```

README.md

Supported models

The following model architectures and tasks are currently supported by 🤗 Optimum Graphcore:

	Pre-Training	Masked LM	Causal LM	Seq2Seq LM (Summarization, Translation, etc)	Sequence Classification	Token Classification	Ques Answ
BART	✅		❌	✅	✅		❌
BERT	✅	✅	❌		✅	✅	✅
ConvNeXt	✅						
DeBERTa	✅	✅			✅	✅	✅
DistilBERT	❌	✅			✅	✅	✅
GPT-2	✅		✅		✅	✅	
GroupBERT	✅	✅	❌		✅	✅	✅
HuBERT	❌				✅		
LXMERT	❌						✅
RoBERTa	✅	✅	❌		✅	✅	✅
T5	✅			✅			
ViT	❌						
Wav2Vec2	✅						
Whisper	❌			✅			

If you find any issue while using those, please open an issue or a pull request.

Code

main

Go to file t

- .github
- docs
- examples
  - audio-classification
  - image-classification
  - language-modeling
  - multiple-choice
  - question-answering
  - speech-pretraining
  - speech-recognition
  - summarization
  - text-classification
  - token-classification
  - translation
  - README.md
- notebooks
  - images
  - packed_bert

optimum-graphcore / examples /

Add file

jimypbr Update examples requirements for sdk3.3 (#434) ✓ 5c4a5c6 · last week History

Name	Last commit message	Last commit date
..		
audio-classification	Update examples requirements for sdk3.3 (#434)	last week
image-classification	Update examples requirements for sdk3.3 (#434)	last week
language-modeling	Bump transformers to 4.29.2 (#389)	last month
multiple-choice	Bump transformers to 4.29.2 (#389)	last month
question-answering	Bump transformers to 4.29.2 (#389)	last month
speech-pretraining	Update examples requirements for sdk3.3 (#434)	last week
speech-recognition	Update examples requirements for sdk3.3 (#434)	last week
summarization	Bump transformers to 4.29.2 (#389)	last month
text-classification	Bump transformers to 4.29.2 (#389)	last month
token-classification	Bump transformers to 4.29.2 (#389)	last month
translation	Bump transformers to 4.29.2 (#389)	last month
README.md	Update README.md	last year

GPT2 training and inference

```python
class GPT2Wrapper(nn.Module):
    def __init__(self, args, model_config):
        super().__init__()
        self.args = args
        if args.checkpoint_input_dir:  # load pretrained model checkpoint
            self.model = GPT2LMHeadModel.from_pretrained(args.checkpoint_input_dir)
        else:  # init model
            self.config = model_config
            self.model = GPT2LMHeadModel(config=self.config)

        for layer in self.model.transformer.h:
            gpt2_attn = OptimizedGPT2Attention(self.model.config, layer_idx=layer.attn.layer_idx)
            gpt2_attn.load_state_dict(layer.attn.state_dict())
            layer.attn = gpt2_attn

        if args.embedding_serialization_factor > 1:
            serialized_lmhead = SerializedLinear(
                self.model.config.n_embd,
                self.model.config.vocab_size,
                args.embedding_serialization_factor,
                bias=False,
                mode=poptorch.MatMulSerializationMode.OutputChannels,
            )
            serialized_lmhead.load_state_dict(self.model.lm_head.state_dict())
            self.model.lm_head = serialized_lmhead
            self.model.tie_weights()

        logger("-------------------- Device Allocation --------------------")
        logger("Embedding  --> IPU 0")
        self.model.transformer.wte = poptorch.BeginBlock(self.model.transformer.wte, "wte", ipu_id=0)
        self.model.transformer.wpe = poptorch.BeginBlock(self.model.transformer.wpe, "wpe", ipu_id=1)
        outline_attribute(self.model.transformer.ln_f, "LayerNorm")

        layer_ipu = _get_layer_ipu(args.layers_per_ipu)
        for index, layer in enumerate(self.model.transformer.h):
            ipu = layer_ipu[index]
            self.model.transformer.h[index] = poptorch.BeginBlock(layer, f"Encoder{index}", ipu_id=ipu)
            logger(f"Layer {index:<2} --> IPU {ipu}")

        logger(f"LM_head --> IPU 0")
        self.model.lm_head = poptorch.BeginBlock(self.model.lm_head, ipu_id=0)

    def forward(self, input_ids):
        transformer_outputs = self.model.transformer(input_ids=input_ids)
        hidden_states = transformer_outputs[0]
        lm_logits = self.model.lm_head(hidden_states)
```

Optimized attention

Serialized embedding

Pipeline-parallel

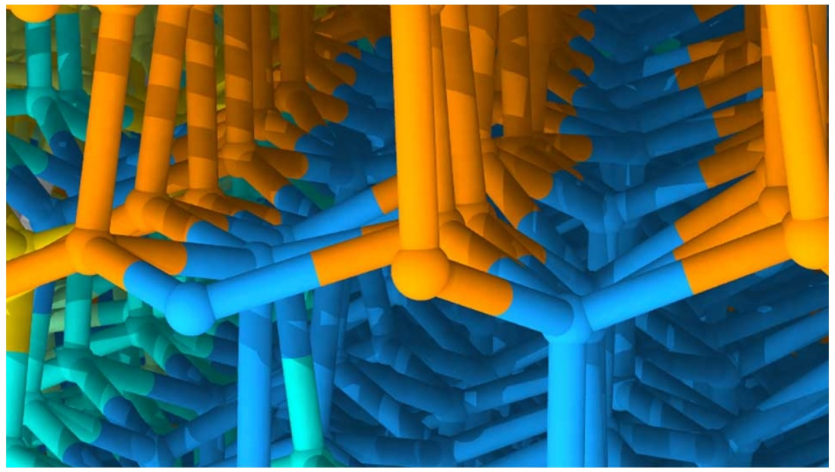
APPLY AND JOIN TODAY

Argonne Leadership Computing Facility

HOME / SCIENCE

Director's Discretionary Allocation Program

The ALCF Director's Discretionary program provides "start up" awards to researchers working to achieve computational readiness for for a major allocation award.



Molecular dynamics simulations based on machine learning help scientists learn about the movement of the boundary between ice grains (yellow/green/cyan) and the stacking disorder that occurs when hexagonal (orange) and cubic (blue) pieces of ice freeze together. Image: Henry Chan and Subramanian Sankaranarayanan, Argonne National Laboratory

Apply at alcf.anl.gov/science/directors-discretionary-allocation-program

general

charlieb 6:05 AM

Pleased to share with you all some new work from the Graphcore research team!

Our paper *Unit Scaling* introduces a new method for low-precision number formats, making FP16
We've managed to train BERT in these formats for the first time without loss scaling.

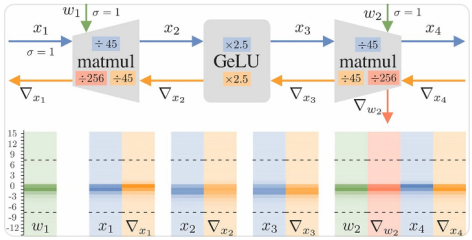
- You can find our blog post here: https://www.graphcore.ai/posts/simple-fp16-and-fp8-trainin
- Paperspace notebook (try it yourself!): https://ipu.dev/qXfm2a
- Arxiv paper: https://arxiv.org/abs/2303.11257

(& we were also featured on Davis Blalock's popular ML newsletter this week) (edited)

graphcore.ai

Simple FP16 and FP8 training with unit scaling

Unit Scaling is a new low-precision machine learning method able to train language models in FP16 and FP8 without loss scaling. (69 kB)



arXiv.org

Unit Scaling: Out-of-the-Box Low-Precision Training

We present unit scaling, a paradigm for designing deep learning models that simplifies the use of low-precision number formats. Training in FP16 or the recently proposed FP8 formats offers substantial efficiency gains, but can lack sufficient range for out-of-the-box training. Unit scaling addresses this by introducing a principled approach to model numerics: seeking unit variance of

Show more

8 8 7 1 1 1

Join at graphcore.ai/join-community


THANK YOU!