DEEP DIVE ON GRAPH NEURAL NETWORKS AND LARGE LANGUAGE MODELS

July 26, 2023

Alexander Tsyplikhin

GRAPHCORE

AGENDA

GNNs

- Graphcore IPUs and PyTorch Geometric
- Case study: SchNet for molecular property prediction

LLMs

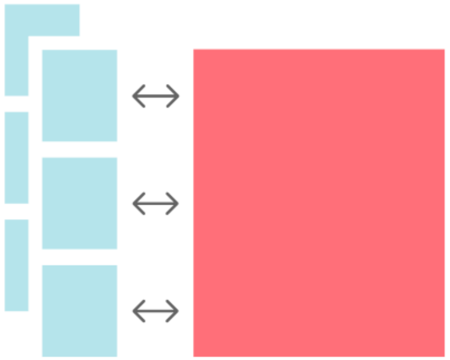
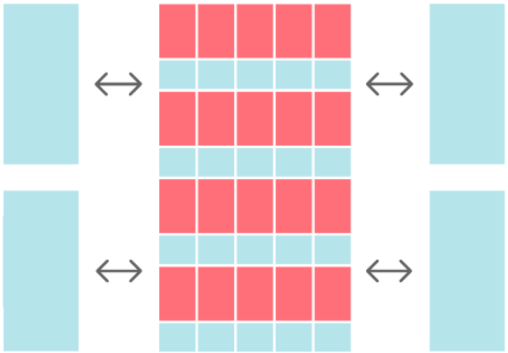
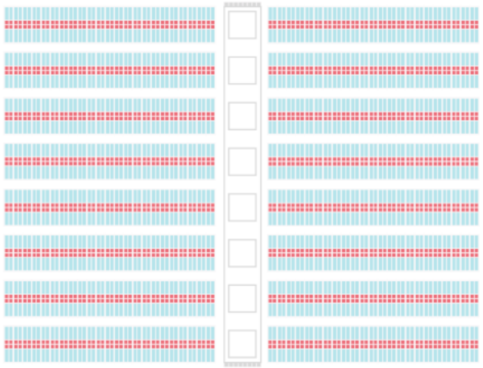
- HuggingFace Optimum
- PopART for GPT-3 175B

Q&A

Slides courtesy of AriannaS, AdamS, SteveB, JoshK

IPU – Architectured For AI

Massive parallelism with ultrafast memory access

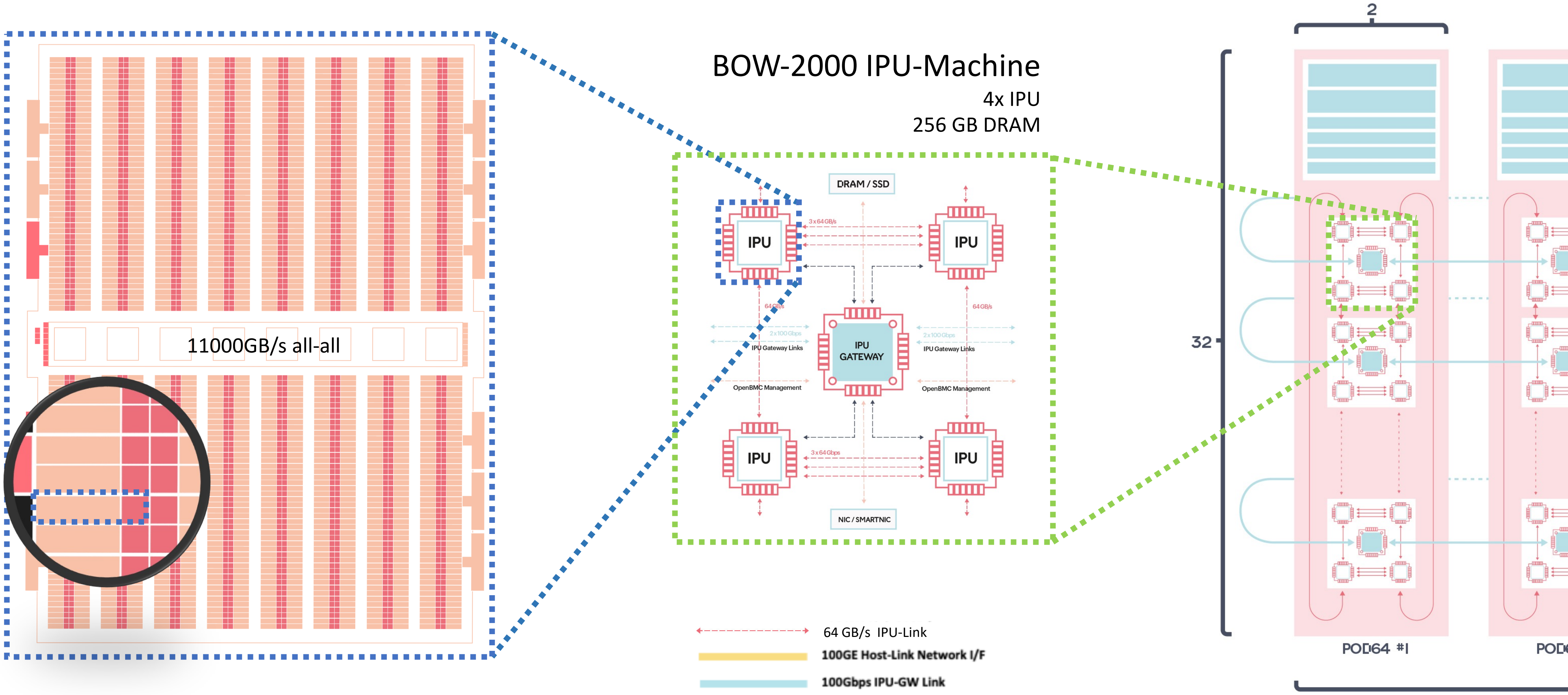
	CPU	GPU	IPU
Parallelism	Designed for scalar processes	SIMD/SIMT architecture. Designed for large blocks of dense contiguous data	Massively parallel MIMD. Designed for fine-grained, high-performance computing
Processors Memory			
Memory Access	Off-chip memory	Model and data spread across off-chip and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

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



OGB-LSC
Large – Scale Challenge

@ NeurIPS 2022

GRAPHCORE IPU ACHIEVES DOUBLE FIRST PLACE!

GRAPH-LEVEL PREDICTION	LINK-LEVEL PREDICTION
1st	1st
GRAPHCORE Valence Mila	GRAPHCORE

https://ogb.stanford.edu/neurips2022/results/

Open Graph Benchmark was established in 2020 with the aim of objectively measuring the performance of different graph models and compute systems

"As I started applying IPUs for molecular property predictions, I was shocked to see the speed improvements over traditional methods."

Dominique Beaini, Research Team Lead at Valence Discovery and Associate Professor at Mila

Valence Mila

OGB-LSC PCQM4MV2 CHALLENGE
THE IPU ADVANTAGE

- Simulating molecular properties using traditional methods (like DFT – Dense Functional Theory) is a very slow process

- Finding the optimal model & implementation required fast experimentation and innovation to explore combined benefits of GNN approaches with transformer-style attention

- The IPUs unique MIMD architecture and ultra-fast memory bandwidth enables :
  - Flexibility for **innovation**
  - High performance for **speed of experimentation**

- IPUs efficient scaling enabled quick experimentation on small models & efficient tuning on larger 'production' models

OGB-LSC

Large - Scale Challenge

OGB-LSC 2022
GRAPH-LEVEL
PREDICTION

PCQM4MV2

1st

GRAPHCORE

Valence Mila

https://ogb.stanford.edu/neurips2022/results/

OGB-LSC WIKIKG90MV2 CHALLENGE
THE IPU ADVANTAGE

- Knowledge graph completion challenge using WikiKG90Mv2 dataset, based on the knowledge graph consisting of pages extracted from Wikipedia

- Dataset scale presents a problem for standard techniques

- This is addressed efficiently by exploitation of the IPU systems high capacity streaming memory, supplementing the large and ultra-fast In-Processor memory & inter-processor communication via IPU-Links

- This enabled quick iteration across the hyperparameter space and experimentation with new ideas, training of hundreds of models to convergence, and in the end construction of an ensemble of models for increased predictive power

OGB-LSC

Large – Scale Challenge

OGB-LSC 2022
LINK-LEVEL
PREDICTION

WIKIKG90MV2

1st

GRAPHCORE

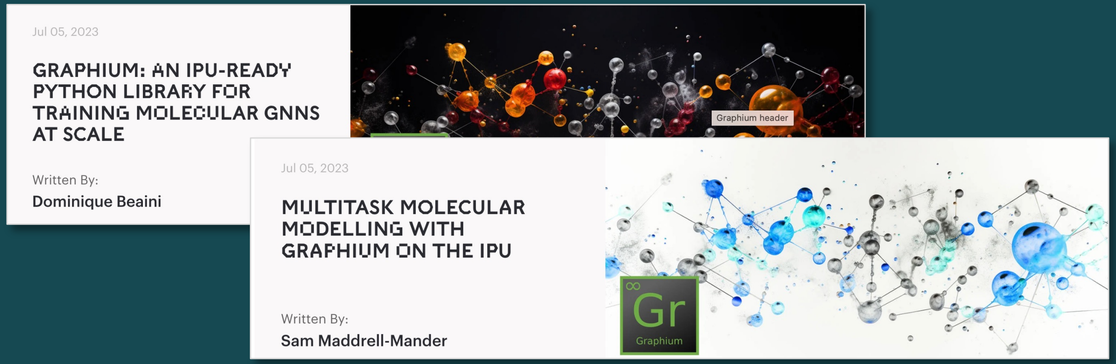
https://ogb.stanford.edu/neurips2022/results/

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

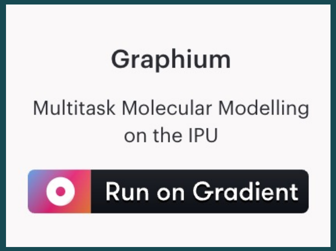
GRAPHIUM FOR IPU

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



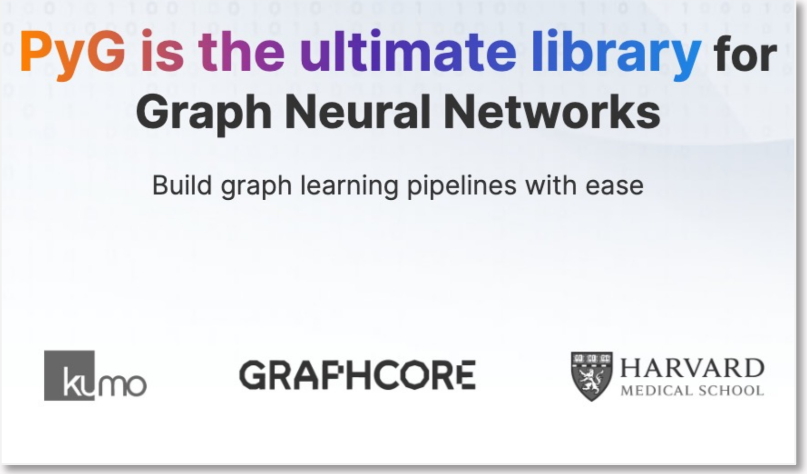
RUN GRAPHIUM ON IPU WITH PAPERSPACE JUPYTER NOTEBOOK



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

PYTORCH GEOMETRIC FOR IPU

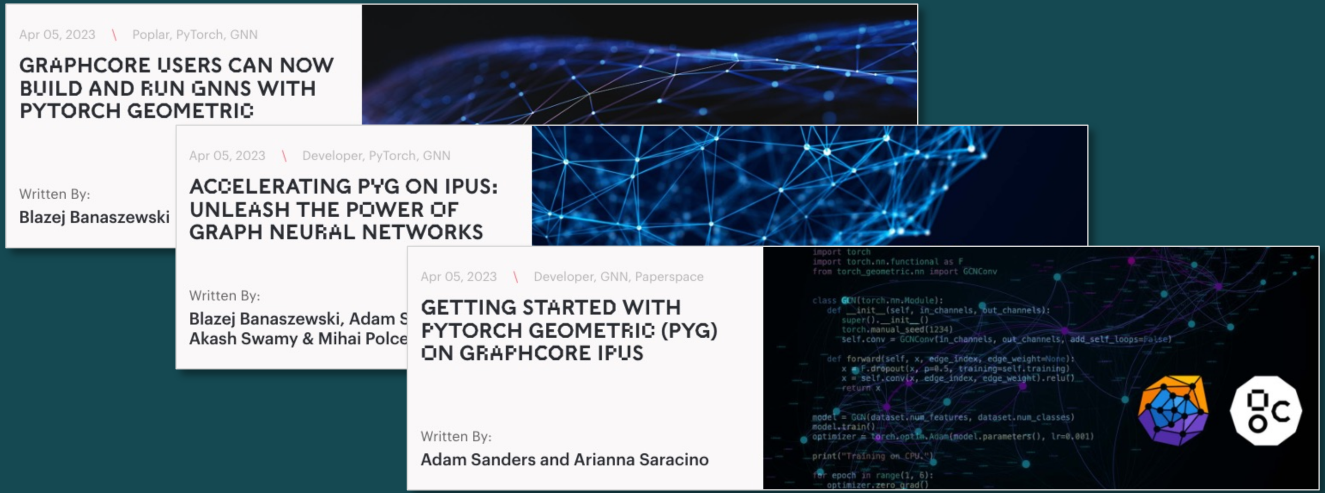
PyG



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



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

PYTORCH GEOMETRIC + IPU

- Hardware lends itself to **GNNs** - fast gather scatter operations
- Already possible to run **PyTorch** on **IPUs**
- **PyTorch Geometric** is the PyTorch library to unify deep learning on graph-structured data
- Aim to make it as easy as possible to use **PyTorch Geometric** on **IPUs** and start **accelerating** your GNNs

AHEAD OF TIME COMPILATION

What?

- The model is **compiled** into a single compute graph with forward and backward passes.

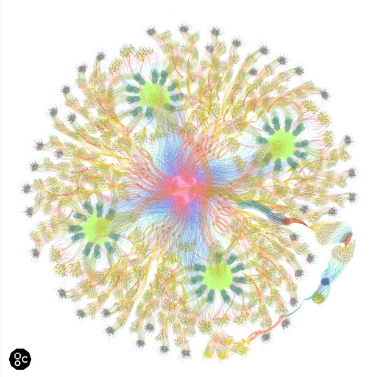
Why?

- **Efficient memory** & **communication**
- Allows **optimisations** to be applied during compilation

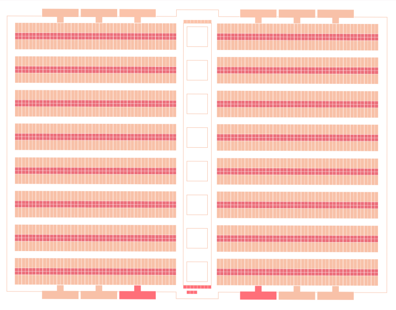
What does it mean for you?

- All tensors in your model must be **fixed size**
- This includes the model **inputs**

PyTorch model



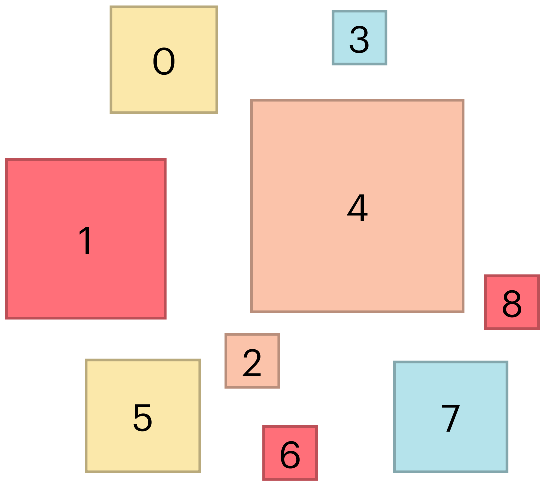
Compiled graph



IPU

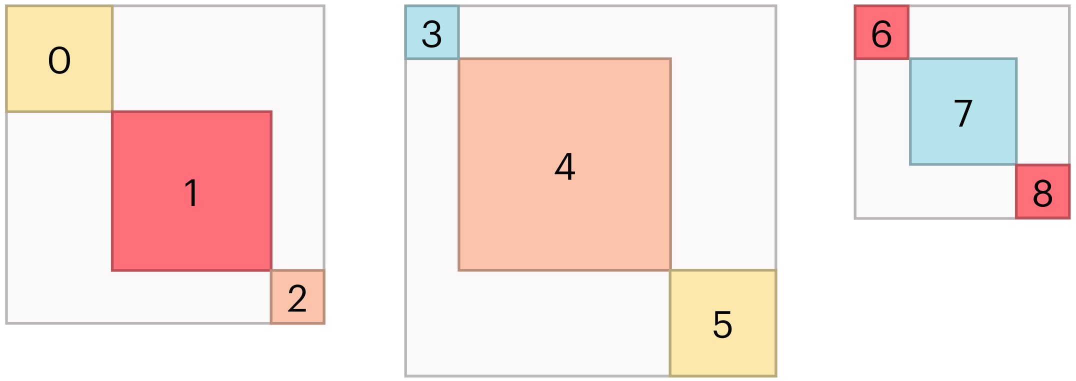
PYG MINI-BATCHING OF SMALL GRAPHS

Adjacency of samples in dataset



Mini-batching (batch size 3)

Adjacency of each mini-batch



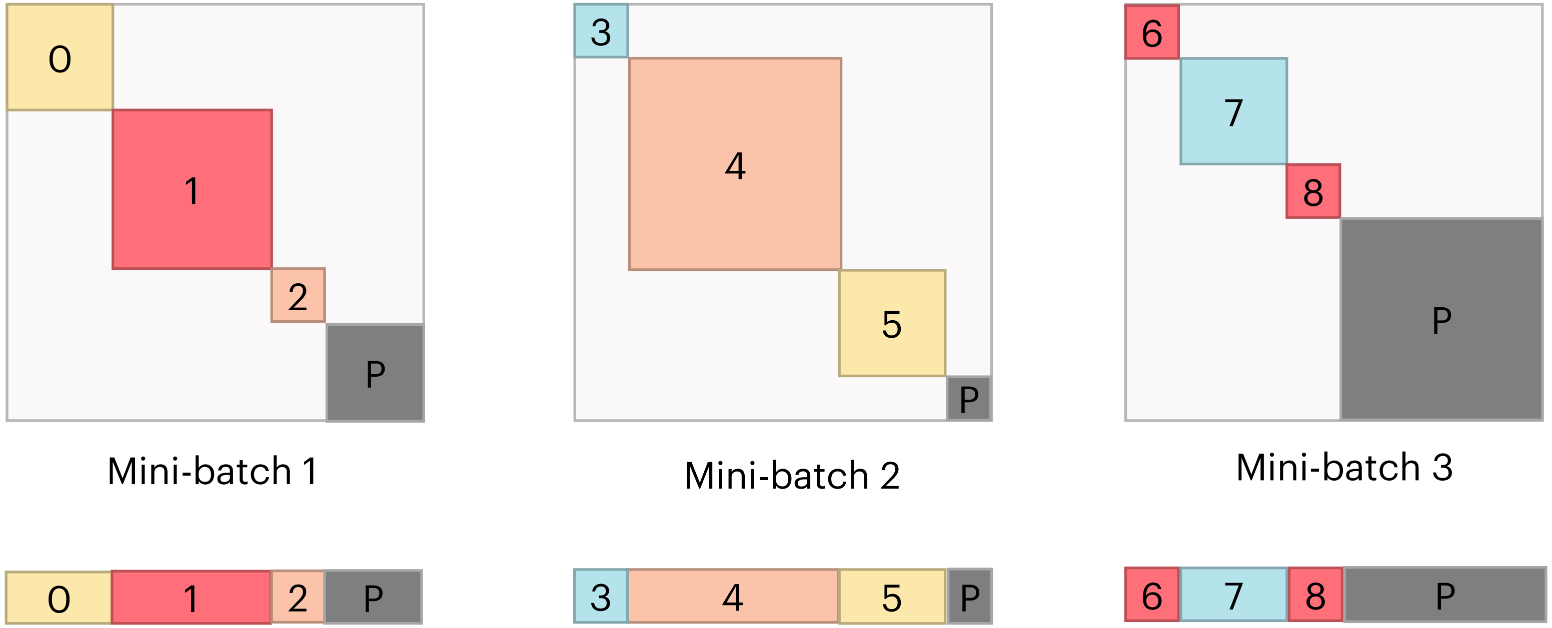
Mini-batch 1 Mini-batch 2 Mini-batch 3

Sparse representation of each mini-batch

FIXED SIZE MINI BATCHING

Adjacency of each mini-batch

Sparse representation of each mini-batch



- **Message passing** just works!
- Do we have to do any **masking**?

```python
# Ignore final padded graph
loss = F.mse_loss(x[:-1], y[:-1])
```

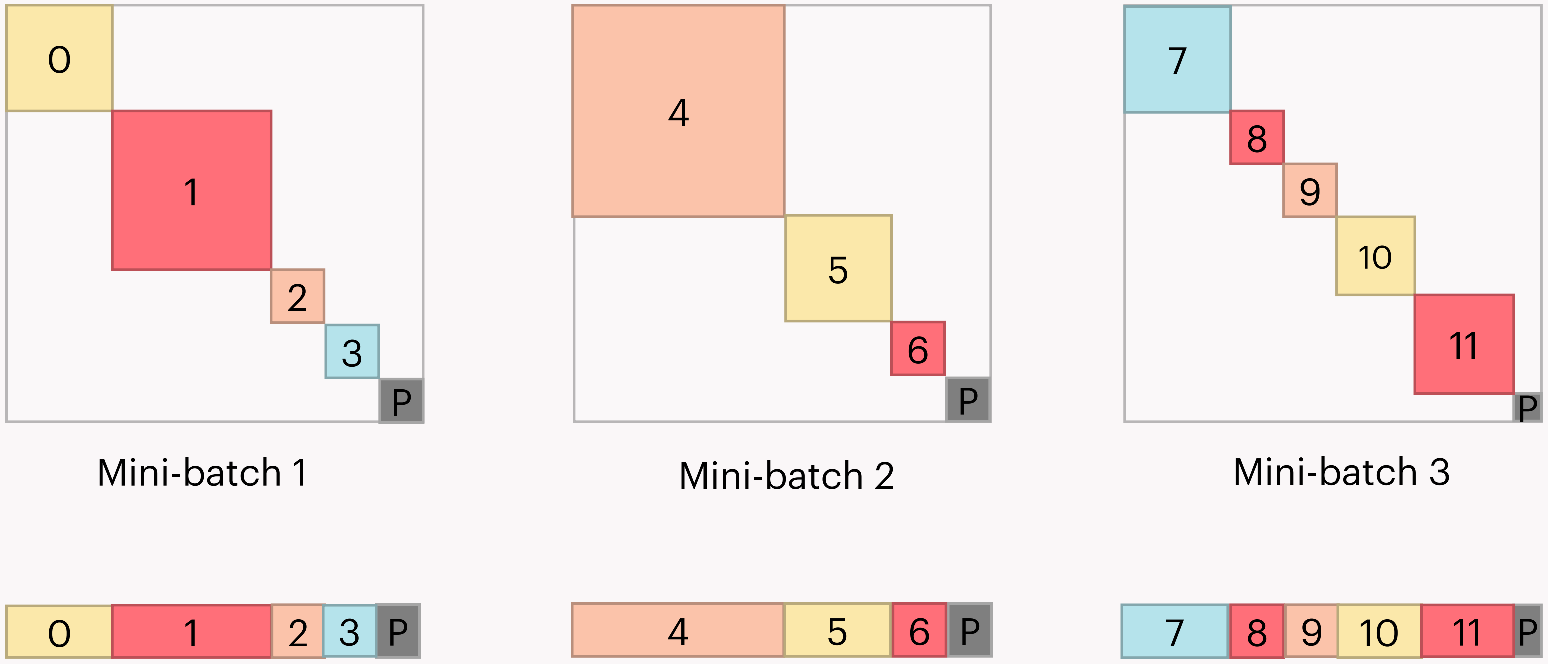
```python
dataloader = FixedSizeDataLoader(dataset,
                                 num_nodes=300,
                                 num_edges=600,
                                 batch_size=10)
```

FIXED SIZE INPUTS WITH PACKING

Stream packing

Adjacency of each mini-batch

Sparse representation of each mini-batch



Global packing

https://arxiv.org/abs/2209.06354

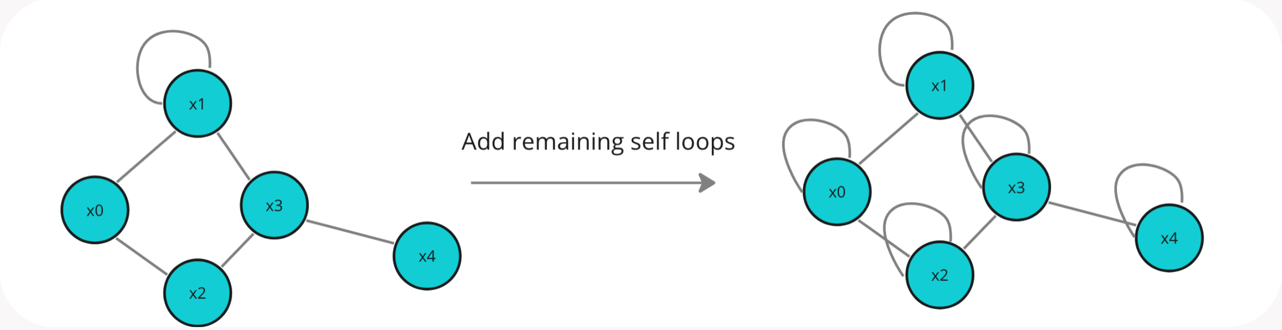
AND OTHER DYNAMIC THINGS

Other operations in your model may be **dynamic** that you wouldn't expect

Adding **self-loops**

```
# Adds remaining self loops
conv = GCNConv(10,
               10,
               add_self_loops=True)
```

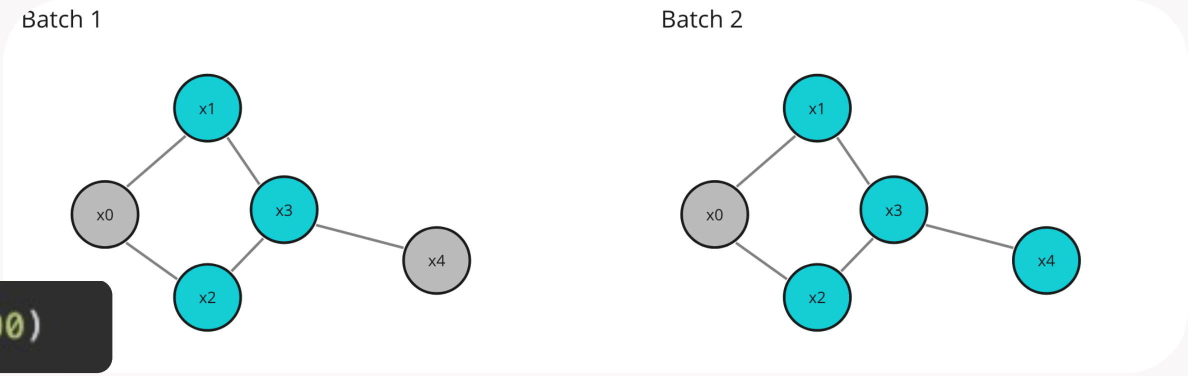
```
transform = T.AddSelfLoops()
```



Using **masks** of different sizes each batch

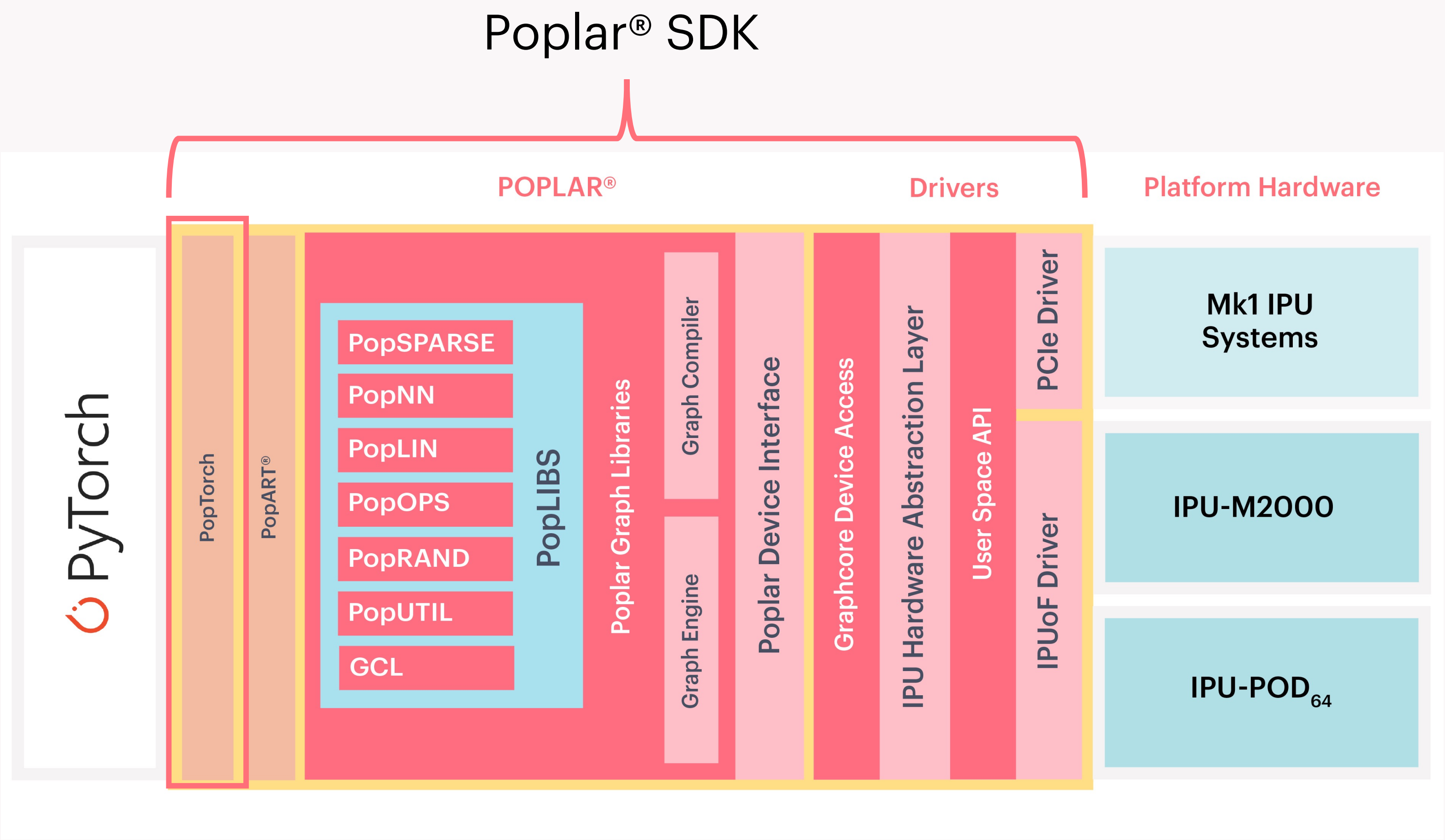
```
out = out[batch.train_mask]
```

```
out = torch.where(batch.train_mask, out, -100)
```



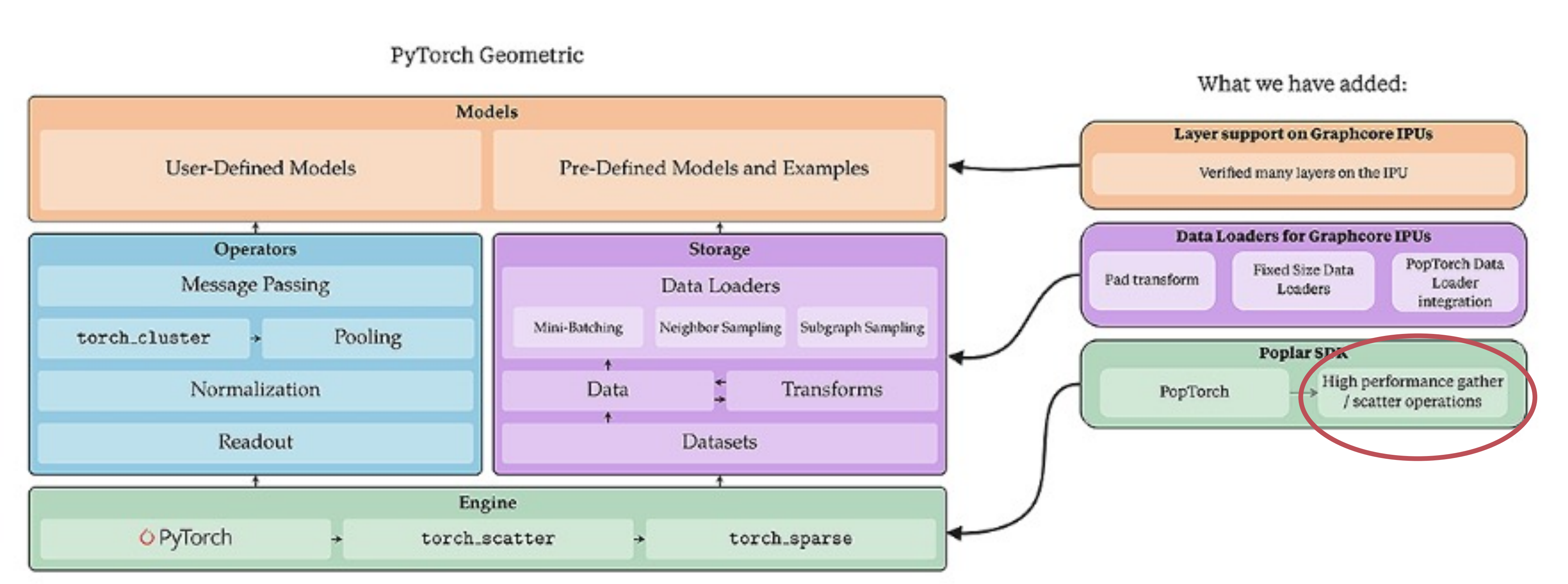
RUNNING PYG ON IPUS: POPTORCH

PopTorch compiles PyTorch models into Poplar executables

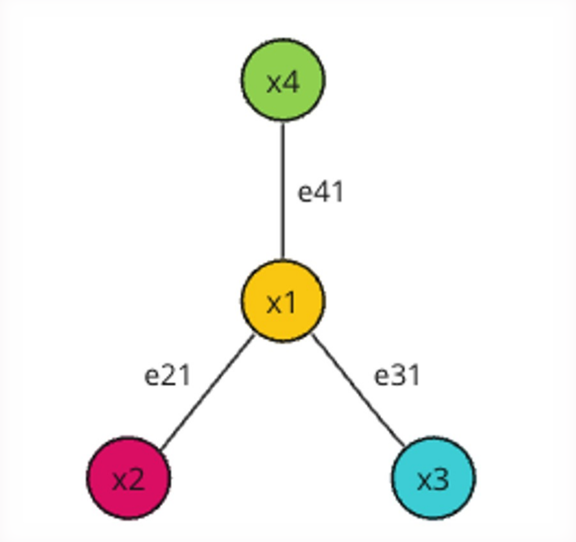


RUNNING PYG ON IPUS: POPTORCH GEOMETRIC

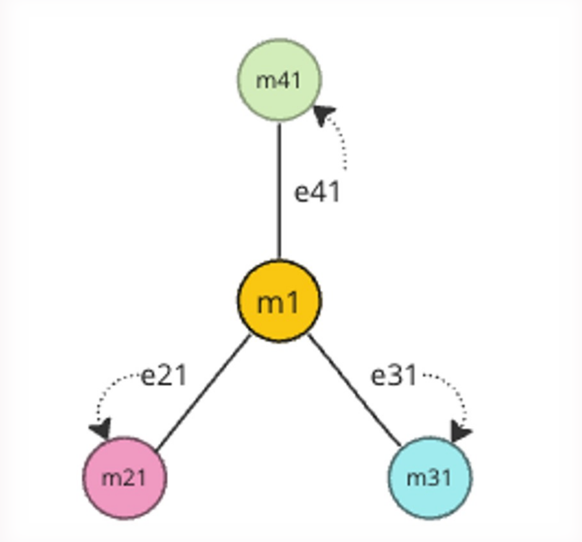
PopTorch Geometric enables GNN models to be run on Graphcore IPUs



BENCHMARKING MESSAGE PASSING AS GATHER/SCATTER OPERATIONS



Original graph
x1 target node



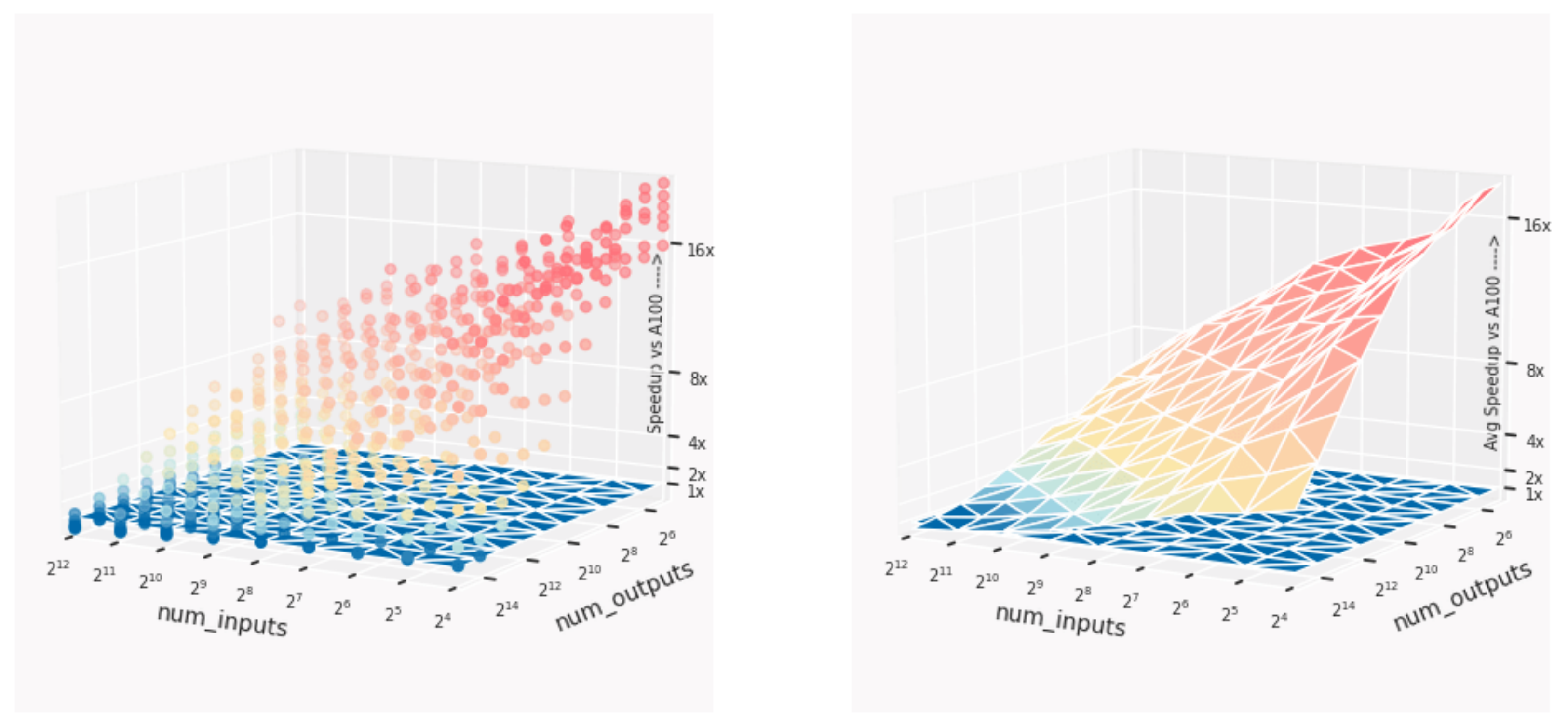
Gather
Messages are collected from neighboring nodes

Scatter(-add)
Messages are aggregated along outbound edges

$$\vec{x}_i' = \overset{\text{Update}}{\gamma}\left(\vec{x}_i, \underbrace{\underset{j \in \mathcal{N}(i)}{\text{aggregate}}}_{\text{Scatter-reduce}} \underbrace{\left[\overset{\text{Message}}{\phi}\left(\vec{x}_i, \vec{x}_j, \vec{e}_{j,i}\right)\right]}_{\text{Gather}}\right)$$

HIGH PERFORMANCE SCATTER-ADD ON IPUS

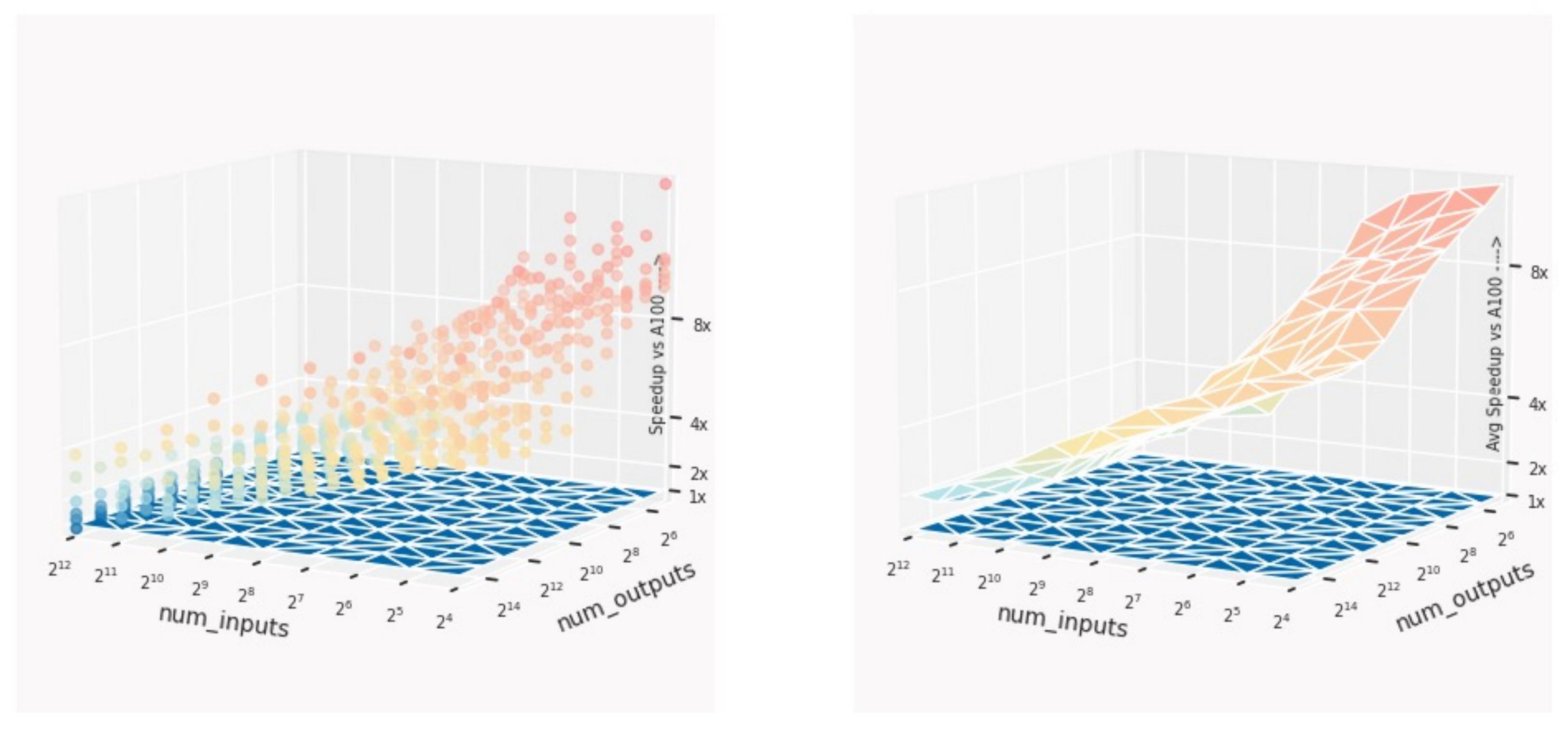
For small scatter input size, IPU achieves >16x speedups vs GPU



Graphcore BOW-M2000 vs NVIDIA A100 (1x, blue plane)

HIGH PERFORMANCE GATHER ON IPUS

For small gather input size, IPU achieves >8x speedups vs GPU

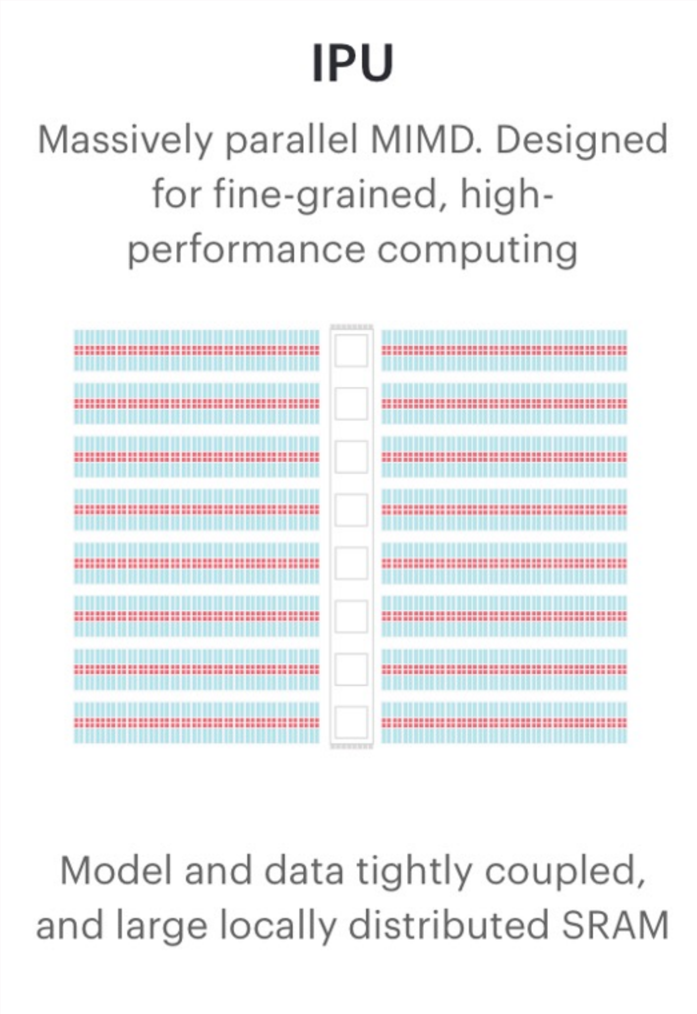


Graphcore BOW-M2000 vs NVIDIA A100 (1x, blue plane)

HIGH PERFORMANCE GATHER-SCATTER OPS ON IPUS

Why faster on IPUs?

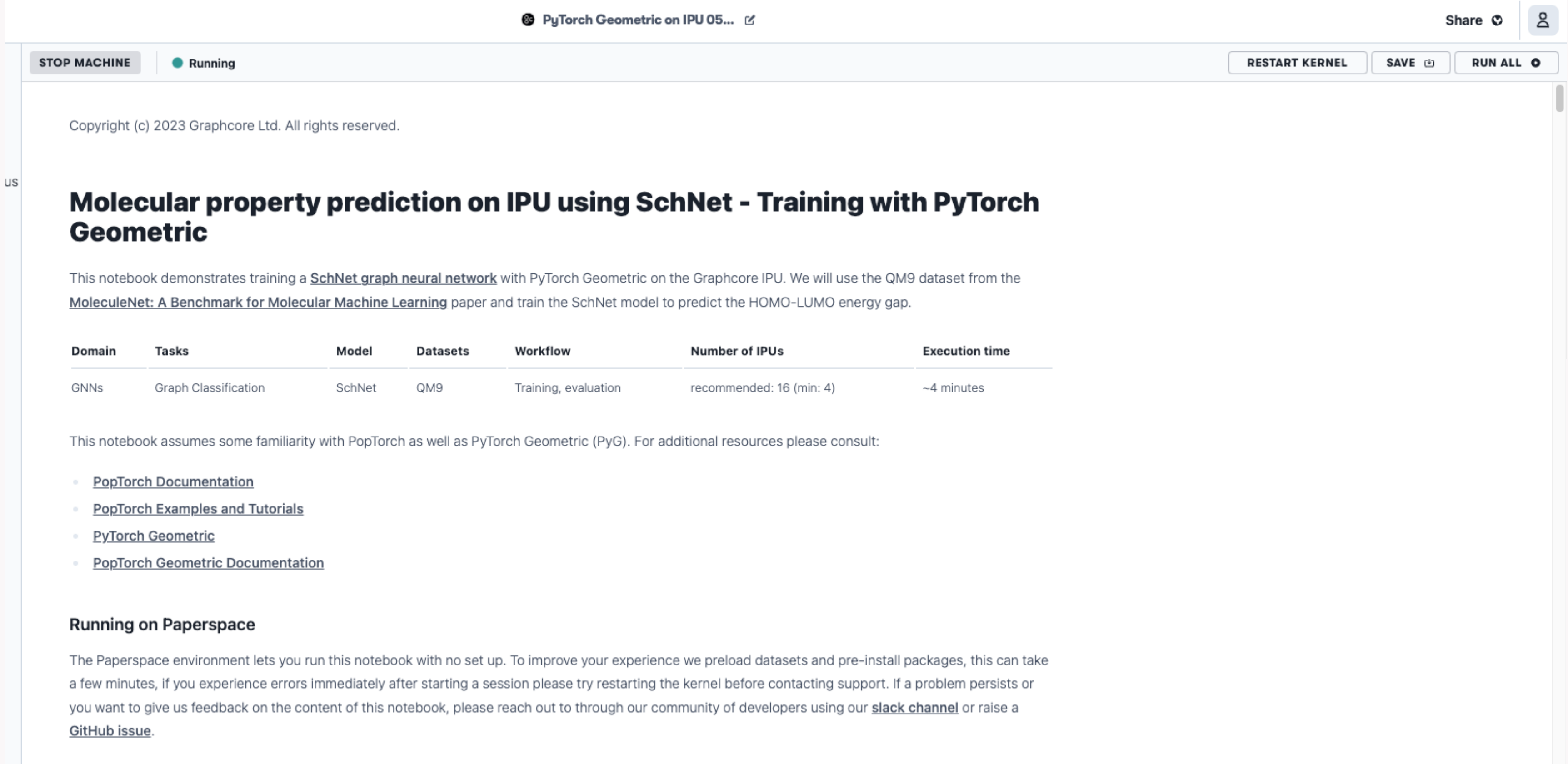
- Large, high bandwidth on-chip SRAM.
- Support for fine-grained parallelism.
- Fast all-to-all communication links.



CASE STUDY: SCHNET FOR MOLECULAR PROPERTY PREDICTION

Training on Graphcore IPUs with PyG

Use the **QM9** dataset from MoleculeNet to train the SchNet model to predict a **graph-level property**, the HOMO-LUMO energy gap


PyTorch Geometric on IPU 05...

Share

STOP MACHINE · Running

RESTART KERNEL · SAVE · RUN ALL

us

Molecular property prediction on IPU using SchNet - Training with PyTorch Geometric

This notebook demonstrates training a [SchNet graph neural network] with PyTorch Geometric on the Graphcore IPU. We will use the QM9 dataset from the [MoleculeNet: A Benchmark for Molecular Machine Learning] paper and train the SchNet model to predict the HOMO-LUMO energy gap.

Domain	Tasks	Model	Datasets	Workflow	Number of IPUs	Execution time
GNNs	Graph Classification	SchNet	QM9	Training, evaluation	recommended: 16 (min: 4)	~4 minutes

This notebook assumes some familiarity with PopTorch as well as PyTorch Geometric (PyG). For additional resources please consult:

- PopTorch Documentation
- PopTorch Examples and Tutorials
- PyTorch Geometric
- PopTorch Geometric Documentation

Running on Paperspace

The Paperspace environment lets you run this notebook with no set up. To improve your experience we preload datasets and pre-install packages, this can take a few minutes, if you experience errors immediately after starting a session please try restarting the kernel before contacting support. If a problem persists or you want to give us feedback on the content of this notebook, please reach out to through our community of developers using our slack channel or raise a GitHub issue.

Molecular property prediction on IPU using SchNet - Training

Run on Gradient

CASE STUDY: SCHNET FOR MOLECULAR PROPERTY PREDICTION

Notebook walkthrough

QM9 dataset

Molecular properties of interest to train SchNet are:

- z atomic number for each atom in the molecule
- pos contains the 3D structure of the molecule
- y contains the 19 regression targets: we slice it `y[:,4]` where the HOMO-LUMO gap is stored

Run 6

```python
datum = dataset[123244]
datum, datum.z, datum.pos, datum.y[:, 4]
```

```
(Data(x=[13, 11], edge_index=[2, 28], edge_attr=[28, 4], y=[1, 19], pos=[13, 3], idx=[1], name='gdb_125563', z=[13]),
 tensor([6, 6, 7, 7, 6, 7, 7, 7, 7, 1, 1, 1, 1]),
 tensor([[-2.7500e-02,  1.4963e+00,  5.2800e-02],
         [-9.1000e-03,  1.2800e-02,  2.6000e-03],
         [-4.2200e-02, -7.5060e-01, -1.0686e+00],
         [-9.3000e-03, -2.1018e+00, -7.2150e-01],
         [ 4.3600e-02, -2.0859e+00,  5.8330e-01],
         [ 1.0010e-01, -2.8658e+00,  1.7010e+00],
         [ 1.3480e-01, -2.0775e+00,  2.8101e+00],
         [ 1.0380e-01, -8.4570e-01,  2.4693e+00],
         [ 4.7300e-02, -8.2400e-01,  1.1011e+00],
         [ 8.7460e-01,  1.8923e+00,  5.3110e-01],
         [-8.0800e-02,  1.8742e+00, -9.6890e-01],
         [-8.9120e-01,  1.8675e+00,  6.1440e-01],
         [ 1.1880e-01, -3.8660e+00,  1.7999e+00]]),
 tensor([5.2708]))
```

CASE STUDY: SCHNET FOR MOLECULAR PROPERTY PREDICTION

Notebook walkthrough

Data loading and mini-batching

Run 12

```python
loader = DataLoader(dataset, batch_size=4)

it = iter(loader)
next(it), next(it)
```

```
(DataBatch(y=[4], pos=[16, 3], z=[16], batch=[16], ptr=[5]),
 DataBatch(y=[4], pos=[21, 3], z=[21], batch=[21], ptr=[5]))
```

AOT compilation requirement on IPU
The mini-batches will need to be adapted to be fixed size

Padding individual dataset samples

Run 17

```python
data = Batch.from_data_list([dataset[0]])
pad_transform = Pad(32, node_pad_value=AttrNamePadding({"z": 0, "pos": 0, "batch": 1}))
padded_batch = pad_transform(data)
padded_batch
```

```
DataBatch(y=[1], pos=[32, 3], z=[32], batch=[32], ptr=[2], num_nodes=32)
```

CASE STUDY: SCHNET FOR MOLECULAR PROPERTY PREDICTION

Notebook walkthrough

Efficient data loading: padding the mini-batch

Run 23

```
batch_size = 8

dataloader = CustomFixedSizeDataLoader(
    dataset, batch_size=batch_size, num_nodes=32 * (batch_size - 1)
)
```

```
# slice off the padding molecule and calculate the mse loss
prediction = prediction[0:-1]
target = target[0:-1]
loss = F.mse_loss(prediction, target)
return prediction, loss
```

The mini-batches have now the same sizes 🎉

Run 24

```
dataloader_iter = iter(dataloader)
first_batch = next(dataloader_iter)
second_batch = next(dataloader_iter)
print(first_batch)
print(second_batch)
```

```
DataBatch(y=[8], pos=[224, 3], batch=[224], ptr=[9], z=[224], num_nodes=224, num_edges=0)
DataBatch(y=[8], pos=[224, 3], batch=[224], ptr=[9], z=[224], num_nodes=224, num_edges=0)
```

CASE STUDY: SCHNET FOR MOLECULAR PROPERTY PREDICTION

Notebook walkthrough

Train SchNet on IPU

Select your hyperparameters and PopTorch options:

Run 27

```
replication_factor = int(num_ipus)
device_iterations = 32
gradient_accumulation = max(1, 16 // replication_factor)
learning_rate = 1e-4
num_epochs = 5
```

Run 28

```
options = poptorch.Options()
options.enableExecutableCaching(executable_cache_dir)
options.outputMode(poptorch.OutputMode.All)
options.deviceIterations(device_iterations)
options.replicationFactor(replication_factor)
options.Training.gradientAccumulation(gradient_accumulation)
```

Recreate the data loader to pass it the selected hyperparameters and options, define the model and compile it on IPU:

Run 32

```
torch.manual_seed(0)
knn_graph = KNNInteractionGraph(cutoff=cutoff, k=28)
model = SchNet(cutoff=cutoff, interaction_graph=knn_graph)
model.train()
model = TrainingModule(
    model, batch_size=batch_size, replace_softplus=additional_optimizations
)
optimizer = poptorch.optim.AdamW(model.parameters(), lr=learning_rate)
training_model = poptorch.trainingModel(model, options, optimizer)

data = next(iter(train_loader))
training_model.compile(data.z, data.pos, data.batch, data.y)
```

```
Graph compilation: 100%|██████████| 100/100 [00:05<00:00]
```

CASE STUDY: SCHNET FOR MOLECULAR PROPERTY PREDICTION

Notebook walkthrough

Train SchNet on IPU

Define the training loop and finally plot the mean of the loss

Run 33

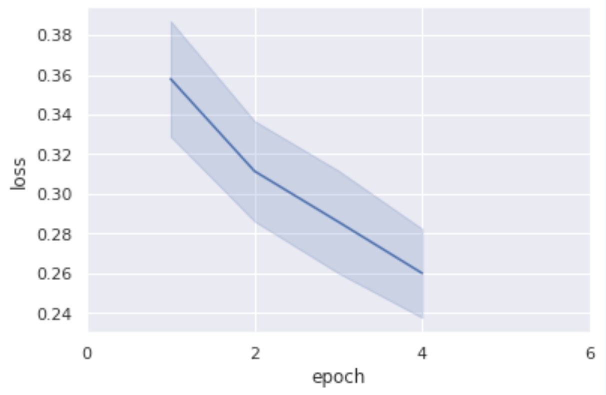
```python
train = []

for epoch in range(num_epochs):
    bar = tqdm(train_loader)
    for i, data in enumerate(bar):
        _, mini_batch_loss = training_model(data.z, data.pos, data.batch, data.y)
        loss = float(mini_batch_loss.mean())
        train.append({"epoch": epoch, "step": i, "loss": loss})
        bar.set_description(f"Epoch {epoch} loss: {loss:0.6f}")
```

```
Epoch 0 loss: 0.375165: 100%|██████████| 30/30 [00:24<00:00,  1.21it/s]
Epoch 1 loss: 0.292912: 100%|██████████| 30/30 [00:24<00:00,  1.21it/s]
Epoch 2 loss: 0.270268: 100%|██████████| 30/30 [00:25<00:00,  1.20it/s]
Epoch 3 loss: 0.255817: 100%|██████████| 30/30 [00:24<00:00,  1.21it/s]
Epoch 4 loss: 0.233512: 100%|██████████| 30/30 [00:24<00:00,  1.21it/s]
```

Run 35

```python
df = pd.DataFrame(train)
g = sns.lineplot(data=df[df.epoch > 0], x="epoch", y="loss", errorbar="sd")
g.set_xticks(range(0, num_epochs + 2, 2))
g.figure.show()
```



TRY OUR GNN NOTEBOOKS IN THE CLOUD

Training dynamic graphs on IPUs using Temporal Graph Networks (TGN)

Run on Gradient

Molecular property prediction on IPU using SchNet - Training

Run on Gradient

Node Classification on IPU using Cluster-GCN - Training

Run on Gradient

Molecular property prediction on IPU using GIN - Training

Run on Gradient

Training NBFnet for inductive knowledge graph link prediction

Run on Gradient

Molecular property prediction using GPS++ (OGB-LSC) - Inference

Run on Gradient

Molecular property prediction using GPS++ (OGB-LSC) - Training

Run on Gradient

Link prediction training for knowledge graphs using Distributed KGE (OGB-LSC)

Run on Gradient

graphcore.ai/ipu-jupyter-notebooks

LARGE LANGUAGE MODELS

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

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
stable_diffusion
text_embeddings_models
wav2vec2
README.md
audio_classification.ipynb
deberta-blog-notebook.ipynb
external_model.ipynb
flan_t5_inference.ipynb
image_classification.ipynb

Documentation • Share feedback

optimum-graphcore / notebooks /

Top

Introduction to Optimum Graphcore	Introduce Optimum-Graphcore with a BERT fine-tuning example.	Run on Gradient
Sentiment analysis with pipelines	Use the sentiment-analysis pipeline to quickly evaluate pre-trained models on the IPU.	Run on Gradient
Real Time Name Entity Recognition on the IPU	Use Gradio and pipelines to prototype a web application doing fast token classification.	Run on Gradient
Train an external model	Show how to train an external model that is not supported by Optimum or Transformers.	Run on Gradient
Train your language model	Show how to train a model for causal or masked language modelling from scratch.	Run on Gradient
How to fine-tune a model on text classification	Show how to preprocess the data and fine-tune a pretrained model on any GLUE task.	Run on Gradient
How to fine-tune a model on language modeling	Show how to preprocess the data and fine-tune a pretrained model on a causal or masked LM task.	Coming soon on Gradient
How to fine-tune a model on token classification	Show how to preprocess the data and fine-tune a pretrained model on a token classification task (NER, PoS).	Run on Gradient
How to fine-tune a model on question answering	Show how to preprocess the data and fine-tune a pretrained model on SQUAD.	Run on Gradient
How to fine-tune a model on multiple choice	Show how to preprocess the data and fine-tune a pretrained model on SWAG.	Run on Gradient
How to fine-tune a model on translation	Show how to preprocess the data and fine-tune a pretrained model on WMT.	Run on Gradient
How to fine-tune a model on summarization	Show how to preprocess the data and fine-tune a pretrained model on XSUM.	Run on Gradient
How to fine-tune a model on audio classification	Show how to preprocess the data and fine-tune a pretrained Speech model on Keyword Spotting	Coming soon on Gradient
How to fine-tune a model on image classfication	Show how to preprocess the data and fine-tune a pretrained model on image classification.	Run on Gradient
wav2vec 2.0 Fine-Tuning on IPU	How to fine-tune a pre-trained wav2vec 2.0 model with PyTorch on the Graphcore IPU-POD16 system.	Run on Gradient
wav2vec 2.0 Inference on IPU	How to run inference on the wav2vec 2.0 model with PyTorch on the Graphcore IPU-POD16 system.	Run on Gradient
Stable Diffusion Text-to-Image generation	Run a Stable Diffusion (Conditional UNet) pipeline on the text-to-image generation task.	Run on Gradient

ai_for_simulation
finance
gnn
multimodal
nlp
bert
bloom/popxl
dolly/popxl
gpt2/pytorch
gpt3_175B/popxl
gpt3_2.7B/popxl
gpt_j/popxl
t5/popxl
preview
probability
speech
tutorials
utils
vision
.git-blame-ignore-revs

README.md

GPT-3 training on IPUs using PopXL

Framework	Domain	Model	Datasets	Tasks	Training	Inference	Reference
PopXL	NLP	GPT-3	Wikipedia	Next sentence prediction, Question/Answering	✅ Min. 256 IPUs (POD256) required	❌	Language Models are Few-Shot Learners

This README describes how to run GPT-3 models for NLP pre-training on Graphcore IPUs using the PopXL library. A combination of phased execution, tensor model parallelism, data parallelism, and remote tensor sharding are utilised to train the models.

This application shows how to run larger models on IPU. The techniques to do this mean that performance is lower than for models that fit in IPU memory. Large model training or fine-tuning requires a big Pod installation. The minimum to run pre-training with this model is a Pod256. PopXL is an experimental framework and may be subject to change in future releases.

Instructions summary

1. Install and enable the Poplar SDK (see Poplar SDK setup)
2. Install the system and Python requirements (see Environment setup)
3. Download the WIKI-103 dataset (See Dataset setup)

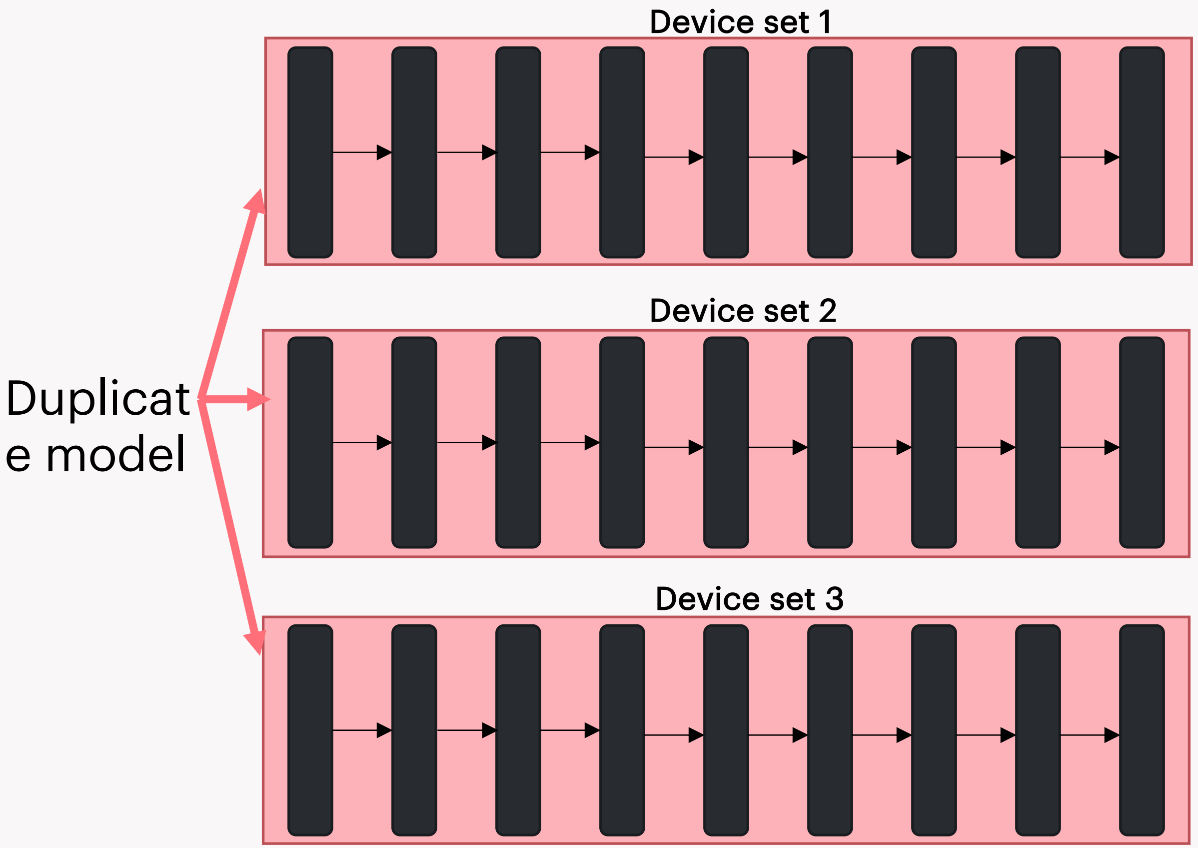
Poplar SDK setup

To check if your Poplar SDK has already been enabled, run:

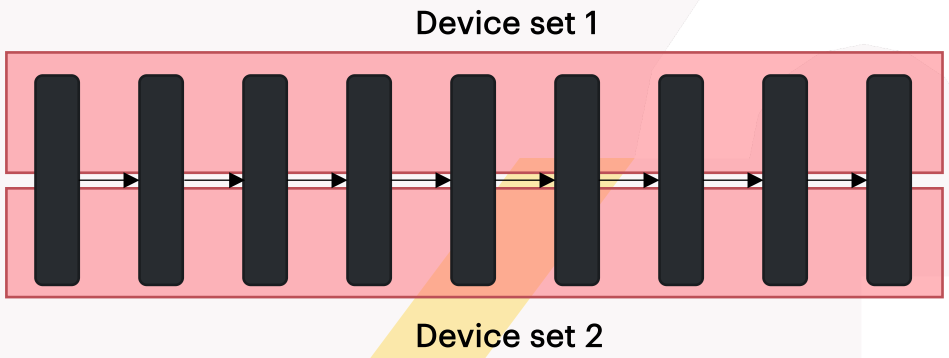
```
echo $POPLAR_SDK_ENABLED
```

Modes of Execution

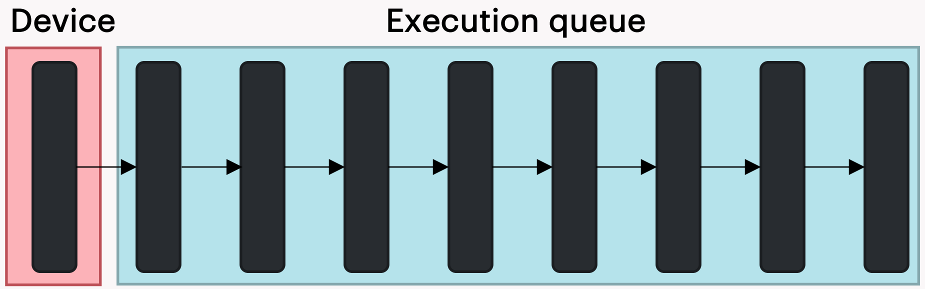
Data Parallelism (DP)



Tensor Parallelism (TP)



Phased Execution (PE)



Matmul TP

- Consider sharding a matmul in two ways:

$$X \to \{n, m\}$$
$$A \to \{m, k\}$$

$$f(X) = XA$$

$$XA = \begin{pmatrix} X_0 & X_1 \\ X_2 & X_3 \end{pmatrix} \begin{pmatrix} A_0 & A_1 \\ A_2 & A_3 \end{pmatrix}$$

$$XA = \begin{pmatrix} X_0A_0 + X_1A_2 & X_0A_1 + X_1A_3 \\ X_2A_0 + X_3A_2 & X_2A_1 + X_3A_3 \end{pmatrix}$$

$$A_A \to \{m, k_A\}$$
$$A_B \to \{m, k_B\}$$

$$\begin{pmatrix} X_0 & X_1 \\ X_2 & X_3 \end{pmatrix} \begin{pmatrix} A_0 & A_1 \\ A_2 & A_3 \end{pmatrix} = \left(\begin{pmatrix} X_0 & X_1 \\ X_2 & X_3 \end{pmatrix} \begin{pmatrix} A_0 \\ A_2 \end{pmatrix} \quad \begin{pmatrix} X_0 & X_1 \\ X_2 & X_3 \end{pmatrix} \begin{pmatrix} A_1 \\ A_3 \end{pmatrix} \right)$$

$$= \underbrace{(XA_A \quad XA_B)}_{\text{Concatenation}}$$

$$X_A \to \{n, m_A\}$$
$$X_B \to \{n, m_B\}$$
$$A_A \to \{m_A, k\}$$
$$A_B \to \{m_B, k\}$$

$$\begin{pmatrix} X_0 & X_1 \\ X_2 & X_3 \end{pmatrix} \begin{pmatrix} A_0 & A_1 \\ A_2 & A_3 \end{pmatrix} = \begin{pmatrix} X_0 \\ X_2 \end{pmatrix} (A_0 \quad A_1) + \begin{pmatrix} X_1 \\ X_3 \end{pmatrix} (A_2 \quad A_3)$$

$$= \underbrace{X_A A_A + X_B A_B}_{\text{Summation}}$$

Column-wise sharding:

$$(XA, XA) := \text{AllGather}(XA_A, XA_B)$$

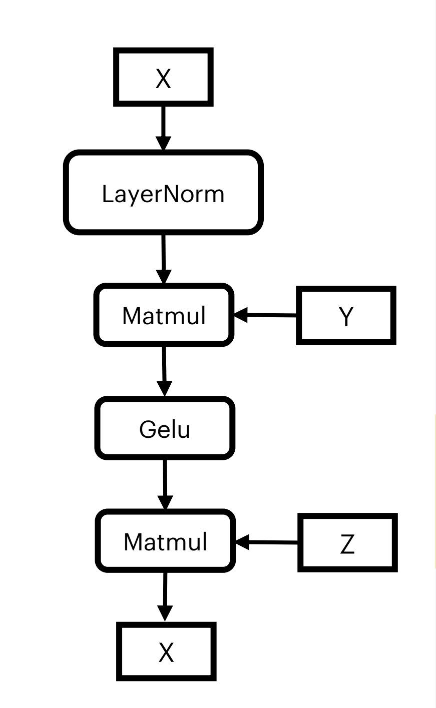
Row-wise sharding:

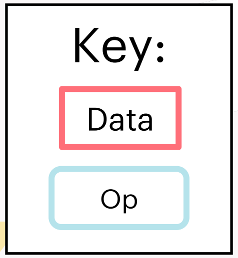
$$(XA, XA) := \text{AllReduce}(X_A A_A, X_B A_B)$$

Feed-Forward Layer: No Parallelism

Shapes

Data/ Op output	Standard
X	[s, h]
Y	[h, 4h]
X := X @ Y	[s, 4h]
Z	[4h, h]
X := X @ Z	[s, h]

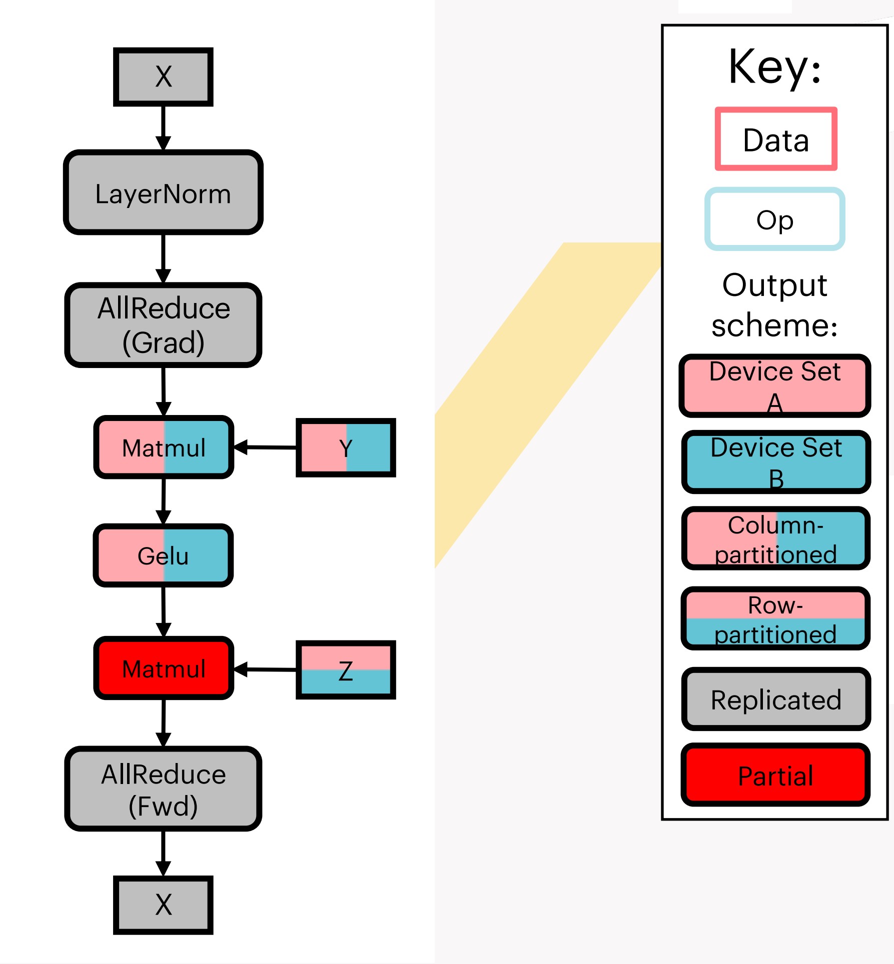




Feed-Forward Layer: 1D Tensor Parallelism

Shapes

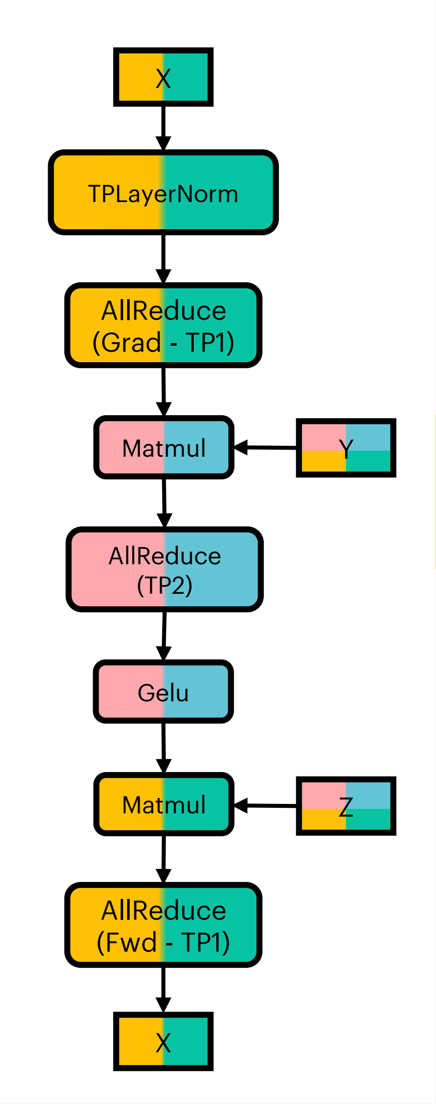
Data/ Op output	Standard	1DTP
X	[s, h]	[s, h]
Y	[h, 4h]	[h, 4h/tp1]
X := X @ Y	[s, 4h]	[s, 4h/tp1]
Z	[4h, h]	[4h/tp1, h]
X := X @ Z	[s, h]	[s, h]

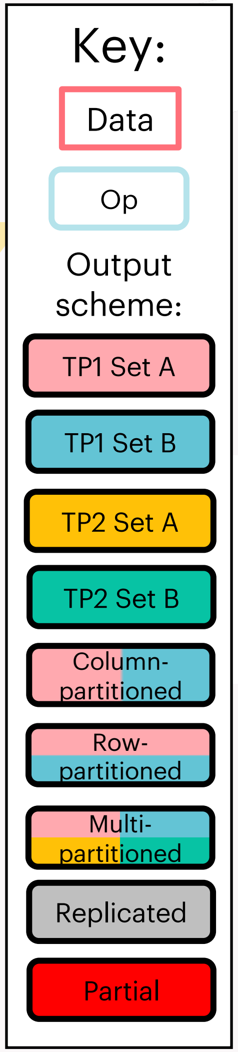


Feed-Forward Layer: 2D Tensor Parallelism

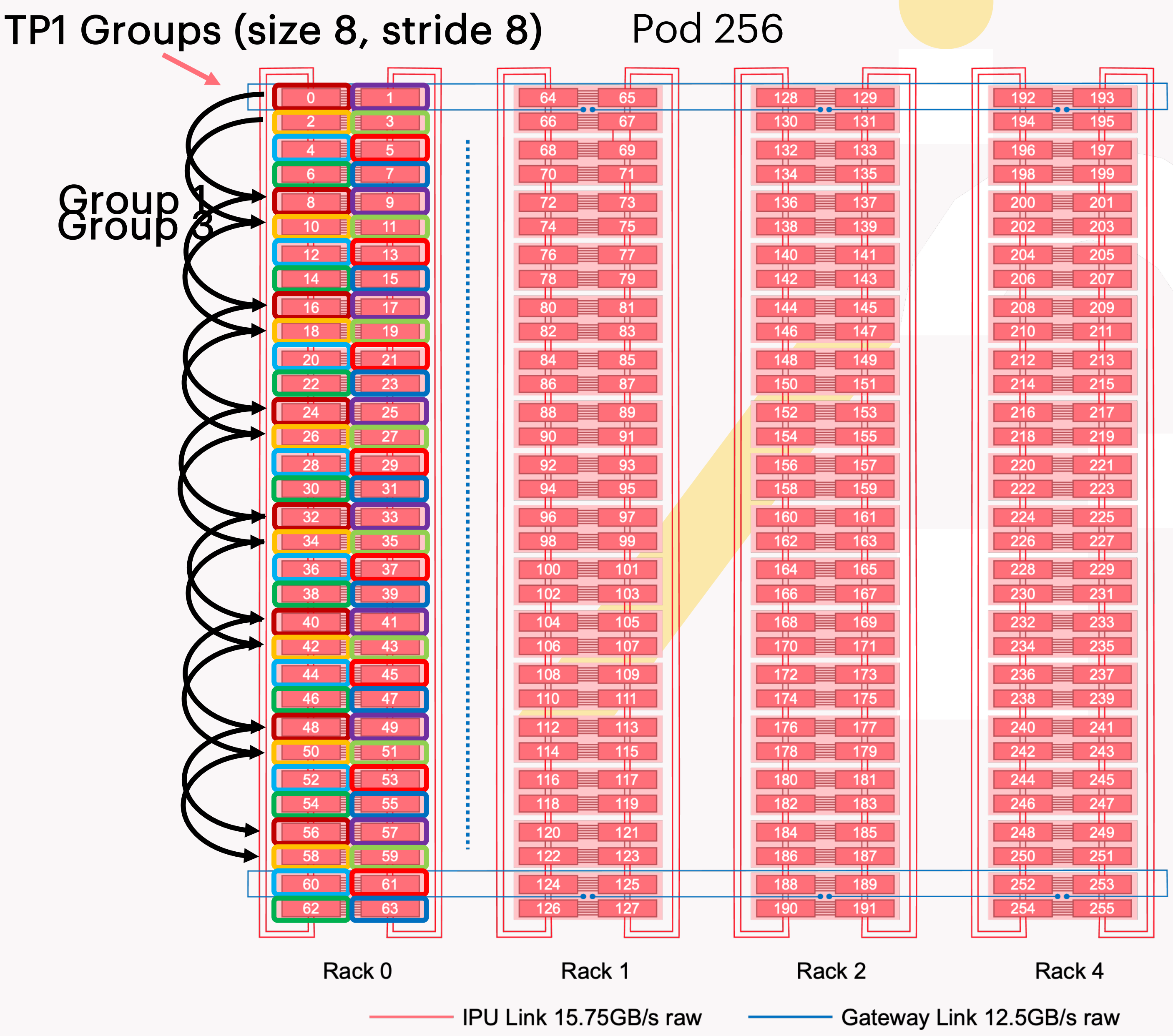
Shapes

Data/ Op output	Standard	1DTP	2DTP
X	[s, h]	[s, h]	[s, h/tp2]
Y	[h, 4h]	[h, 4h/tp1]	[h/tp2, 4h/tp1]
X := X @ Y	[s, 4h]	[s, 4h/tp1]	[s, 4h/tp1]
Z	[4h, h]	[4h/tp1, h]	[4h/tp1, h/tp2]
X := X @ Z	[s, h]	[s, h]	[s, h/tp2]





Model-Device Mapping



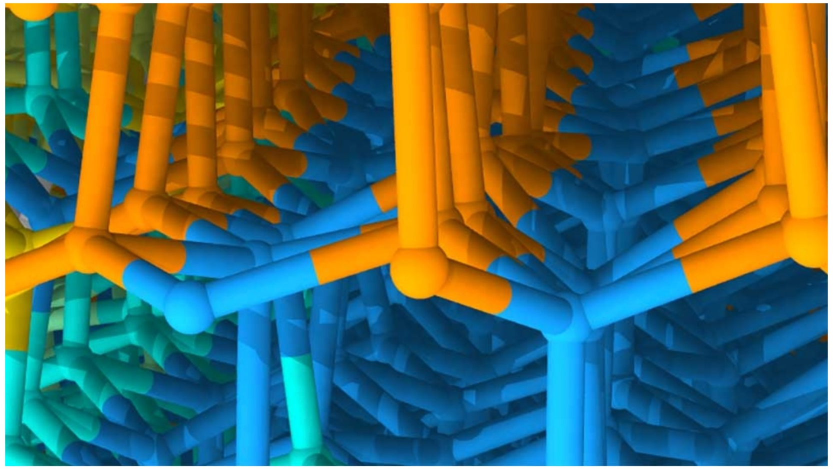
APPLY AND JOIN TODAY

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.

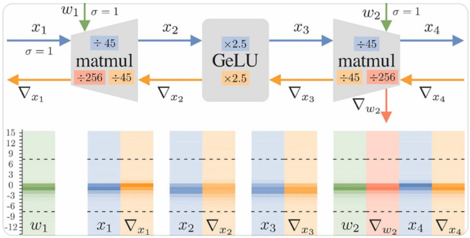
- You can find our blog post here: https://www.graphcore.ai/posts/simple-fp16-and-fp8-trainin
- Paperspace notebook (try it yourself!): https://ipu.dev/qXfm2a
- Arxiv paper: https://arxiv.org/abs/2303.11257

(& 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

Tuesday, 25 July

1:30 PM	→ 1:45 PM	Introduction	15m
1:45 PM	→ 2:15 PM	Graphcore BowPod64 Hardware	30m
2:15 PM	→ 3:00 PM	Software Stack: TensorFlow, PyTorch, and Poplar	45m
3:00 PM	→ 3:15 PM	Break	15m
3:15 PM	→ 3:45 PM	Porting applications with Poplar	30m
3:45 PM	→ 4:30 PM	How to use Bow Pod64@ ALCF	45m

Wednesday, 26 July

1:30 PM	→ 2:15 PM	Deep Dive on Graph neural networks and Large Language Models	45m
2:15 PM	→ 2:45 PM	Profiling with PopVision	30m
2:45 PM	→ 3:00 PM	Break	15m
3:00 PM	→ 3:45 PM	Hands-on session	45m
3:45 PM	→ 4:15 PM	Best Practices, Q&A	30m

THANK YOU!

Q&A