

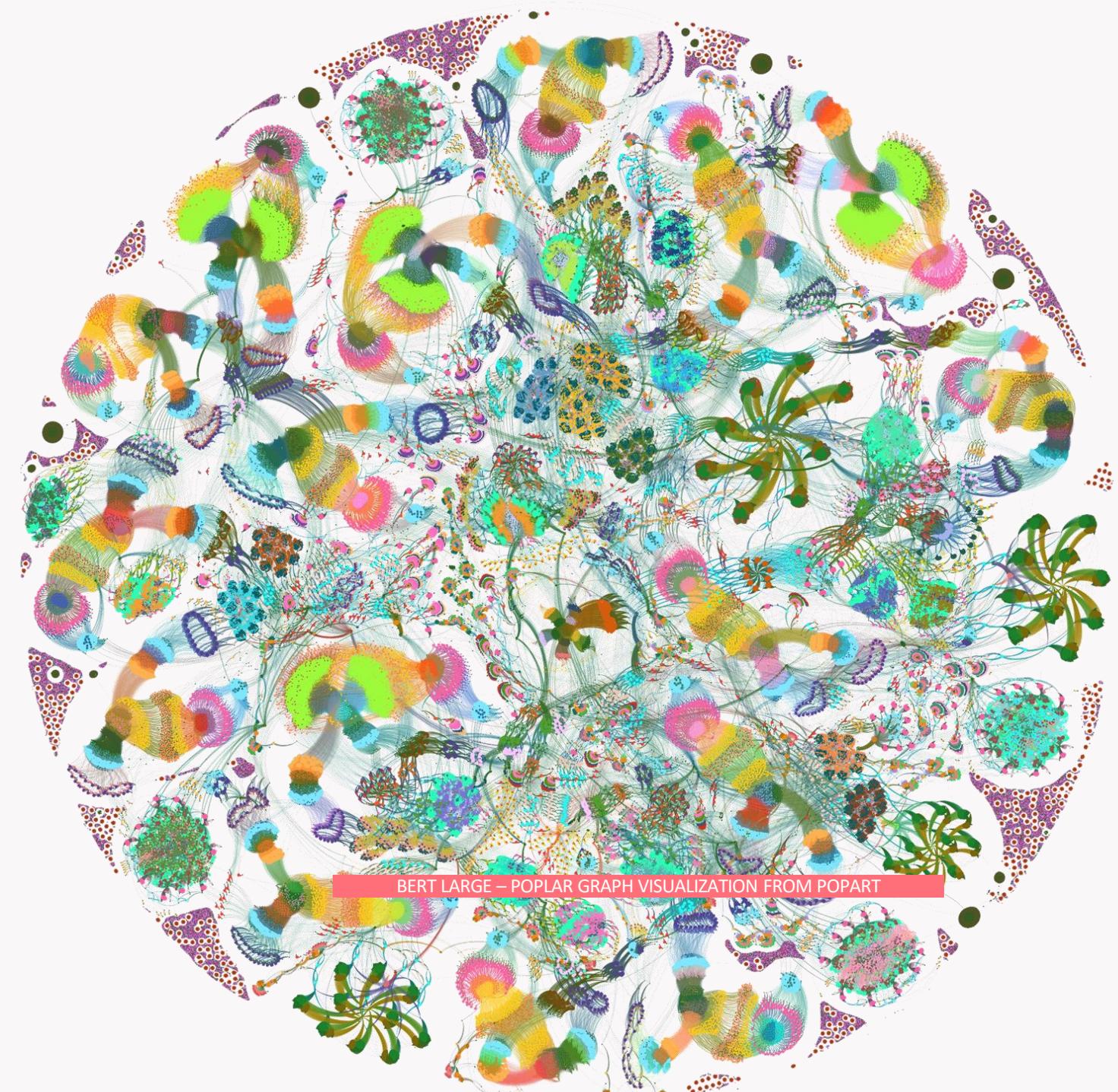
# IPU HANDS-ON SESSION

Minkyu Kim  
Field AI Engineer

# GRAPHCORE



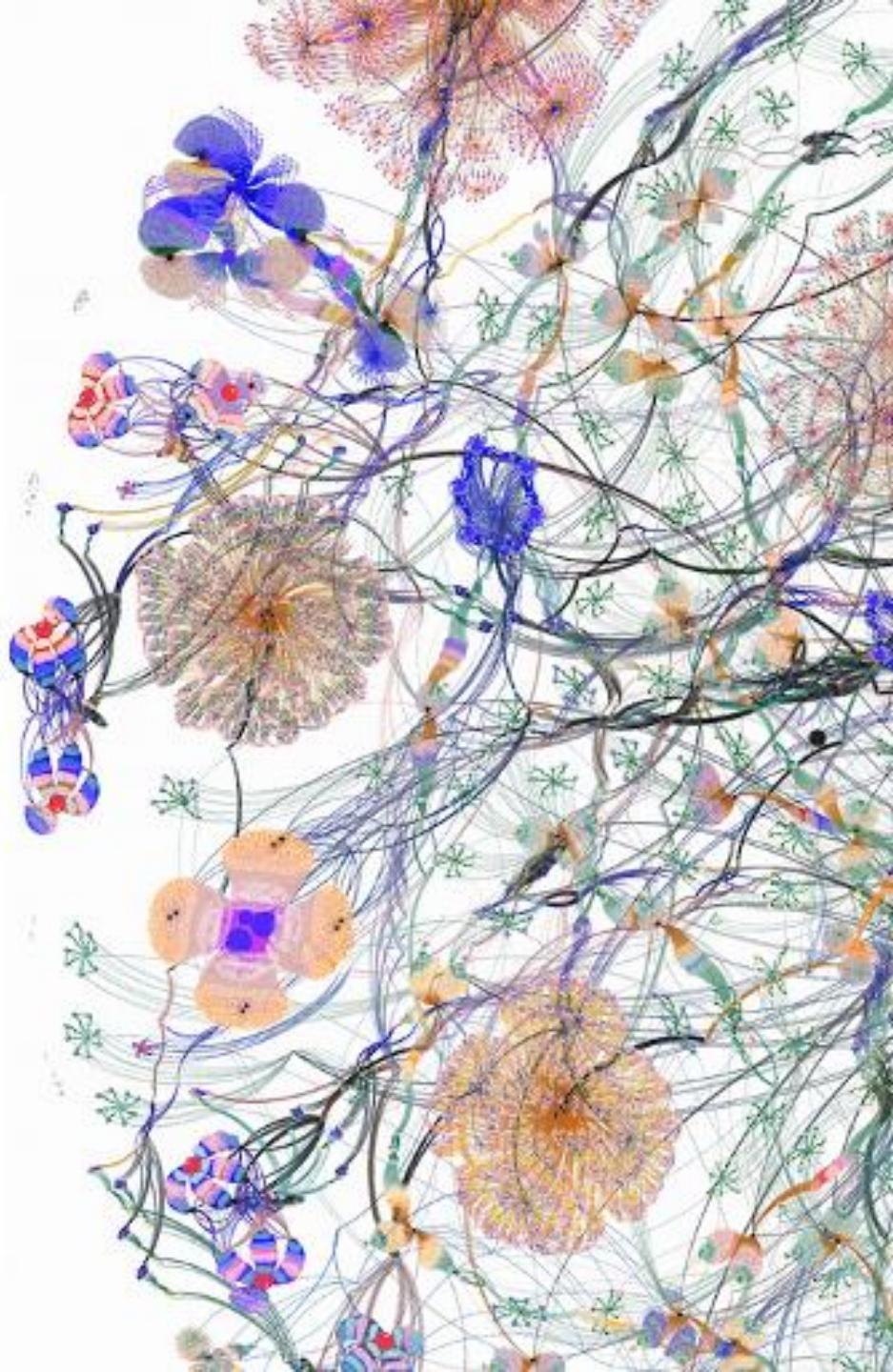
Reinforcement Learning Neural network visualization from **POPLAR™**



## Day02

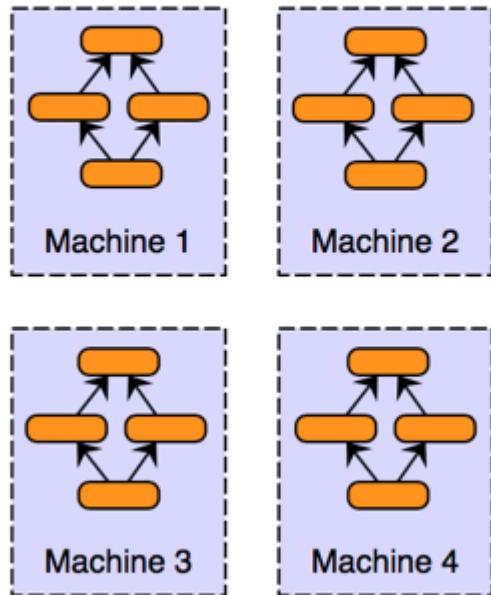
1. Multi IPU Programming
  - Data parallel
  - Model parallel
2. PopVision

# MULTI-IPU PROGRAMMING

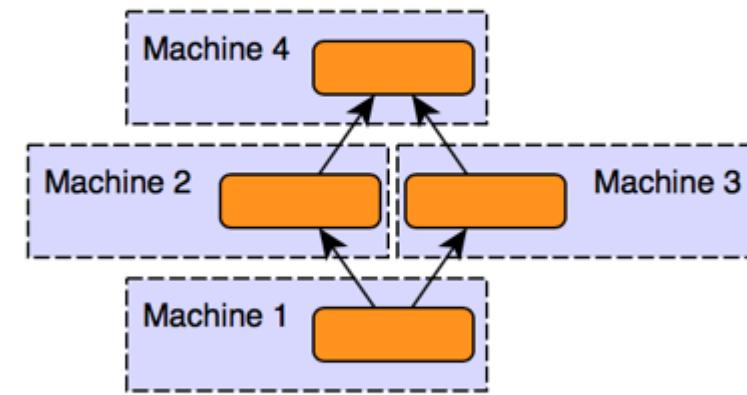


# DATA AND MODEL PARALLELISM

Data Parallelism



Model Parallelism



모델을 여러 칩에 복제하고 서로 다른 데이터를 input

하나의 모델을 여러 칩에 나누어 적재

# DATA PARALLELISM

## ■ 데이터 병렬처리

```
# Create a poptorch.Options instance to override default options
opts = poptorch.Options()

# Run a 100 iteration loop on the IPU, fetching a new batch each time
opts.deviceIterations(100)

# Duplicate the model over 4 replicas.
opts.replicationFactor(4)

training_data = poptorch.DataLoader(opts,
                                    dataset=ExampleDataset(shape=[3, 2],
                                                            length=100000),
                                    batch_size=model_batch_size,
                                    shuffle=True,
                                    drop_last=True)

model = ExampleModelWithLoss(data_shape=[3, 2], num_classes=2)
# Wrap the model in a PopTorch training wrapper
poptorch_model = poptorch.trainingModel(model, options=opts)
```

### replicationFactor(*replication\_factor*)

Number of times to replicate the model (default: 1).

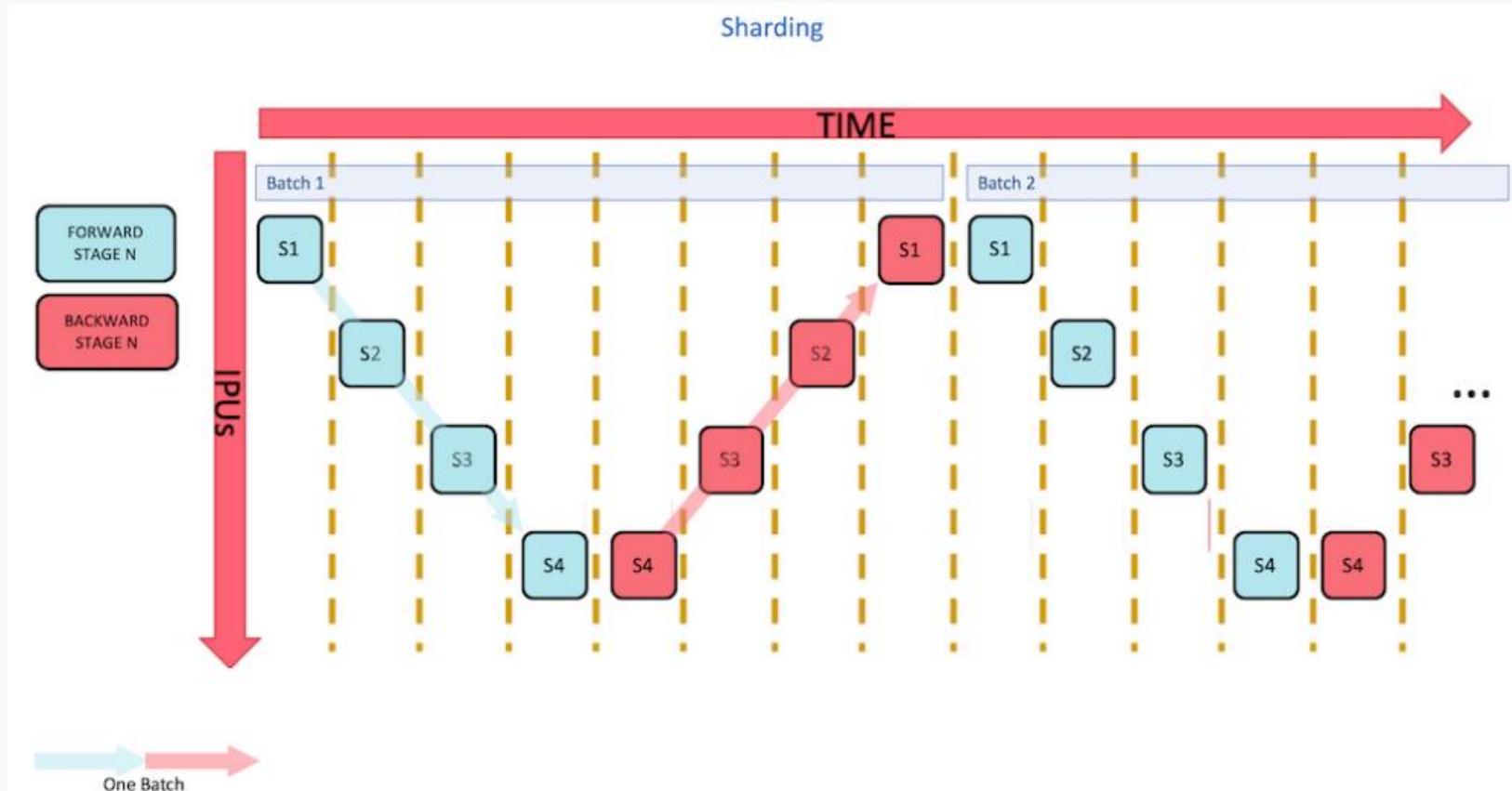
Replicating the model increases the data throughput of the model as PopTorch uses more IPUs. This leads to the number of IPUs used being scaled by *replication\_factor*, for example, if your model uses 1 IPU, a *replication\_factor* of 2 will use 2 IPUs; if your model uses 4 IPUs, a replication factor of 4 will use 16 IPUs in total.

Parameters: *replication\_factor* (*int*) – Number of replicas of the model to create.

Return type: *poptorch.Options*

# MODEL PARALLELISM

- Model Sharding



# MODEL PARALLELISM

## ■ Model Sharding

GPU pytorch 예시 코드

```
from torchvision.models.resnet import ResNet, Bottleneck

num_classes = 1000

class ModelParallelResNet50(ResNet):
    def __init__(self, *args, **kwargs):
        super(ModelParallelResNet50, self).__init__(
            Bottleneck, [3, 4, 6, 3], num_classes=num_classes, *args, **kwargs)

        self.seq1 = nn.Sequential(
            self.conv1,
            self.bn1,
            self.relu,
            self.maxpool,

            self.layer1,
            self.layer2
        ).to('cuda:0')

        self.seq2 = nn.Sequential(
            self.layer3,
            self.layer4,
            self.avgpool,
        ).to('cuda:1')

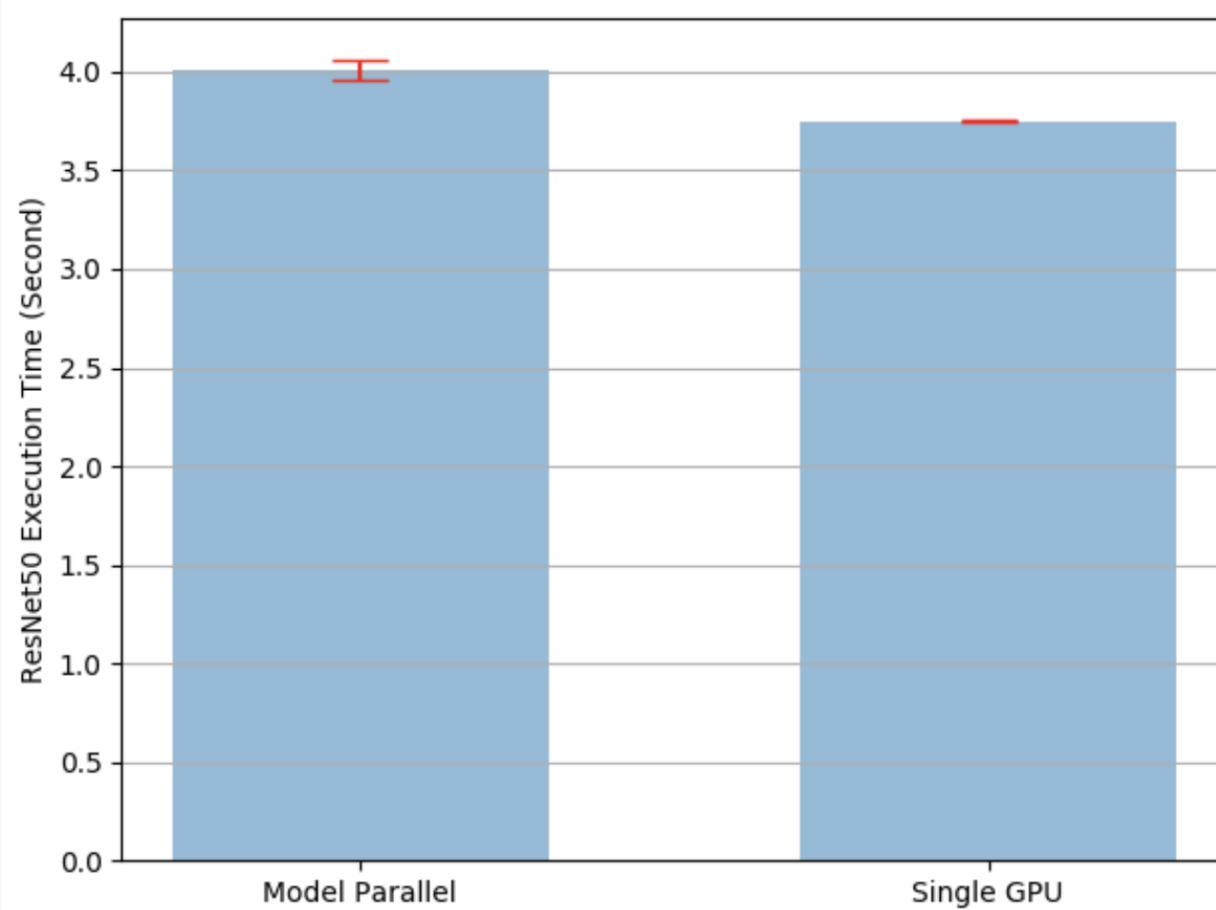
        self.fc.to('cuda:1')

    def forward(self, x):
        x = self.seq2(self.seq1(x).to('cuda:1'))
        return self.fc(x.view(x.size(0), -1))
```

모델을 GPU0와 GPU1에  
나누어 적재

# MODEL PARALLELISM

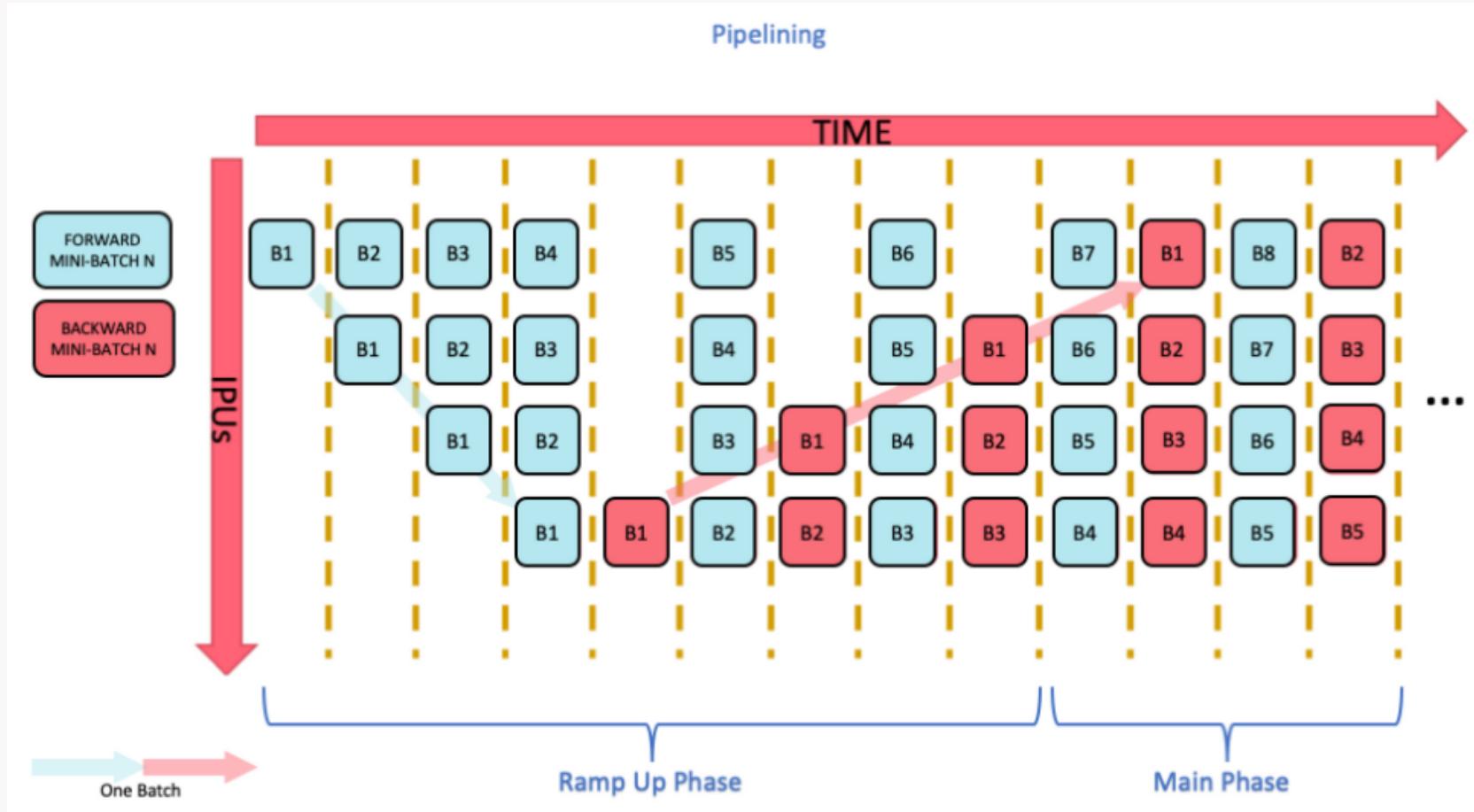
- Model Sharding



=> Single GPU 보다 느림

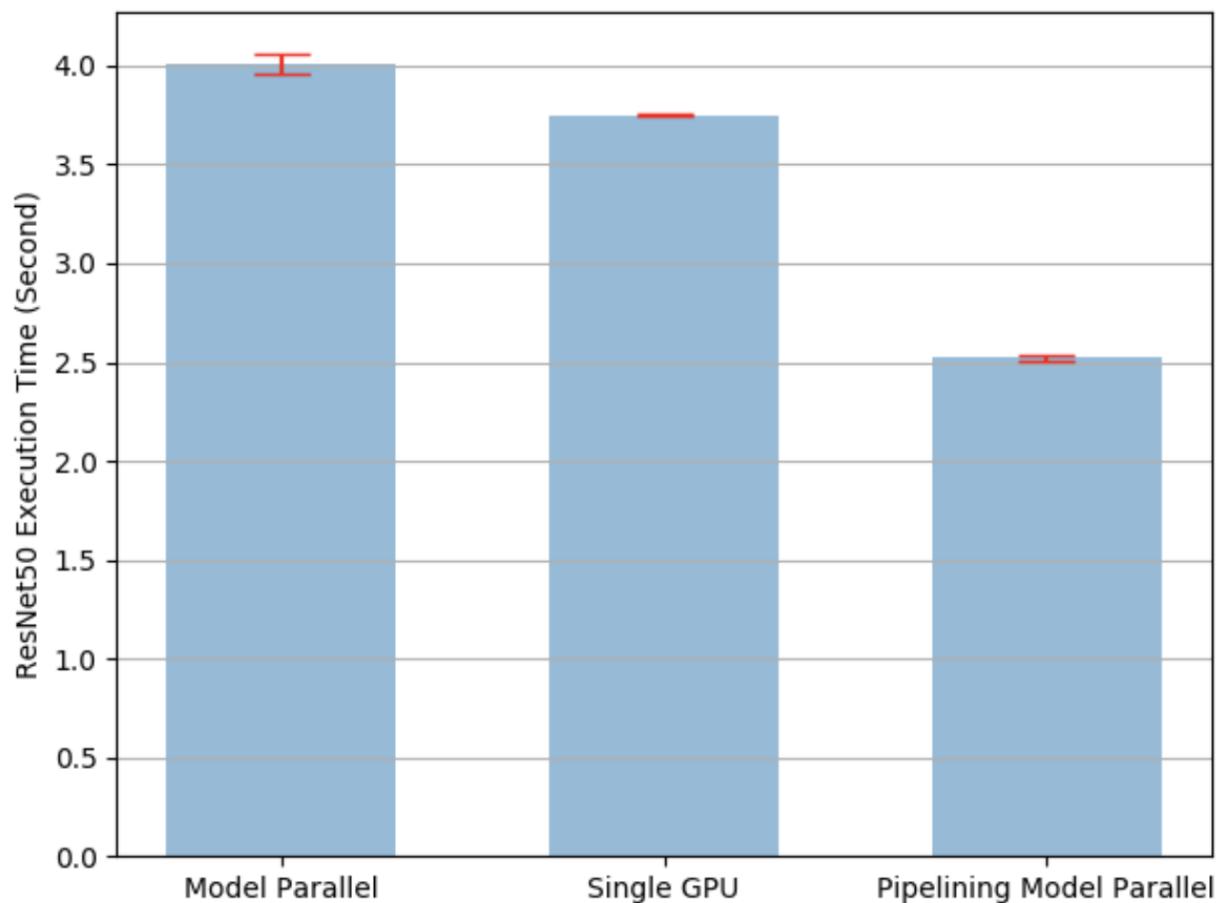
# MODEL PARALLELISM

- Model Pipelining



# MODEL PARALLELISM

- Model Pipelining



=> Pipelining 속도 향상

# MODEL PARALLELISM

## ■ Model Pipelining

GPU pytorch 예시 코드

```
class PipelineParallelResNet50(ModelParallelResNet50):
    def __init__(self, split_size=20, *args, **kwargs):
        super(PipelineParallelResNet50, self).__init__(*args, **kwargs)
        self.split_size = split_size

    def forward(self, x):
        splits = iter(x.split(self.split_size, dim=0))
        s_next = next(splits)
        s_prev = self.seq1(s_next).to('cuda:1')
        ret = []

        for s_next in splits:
            # A. s_prev runs on cuda:1
            s_prev = self.seq2(s_prev)
            ret.append(self.fc(s_prev.view(s_prev.size(0), -1)))

            # B. s_next runs on cuda:0, which can run concurrently with A
            s_prev = self.seq1(s_next).to('cuda:1')

            s_prev = self.seq2(s_prev)
            ret.append(self.fc(s_prev.view(s_prev.size(0), -1)))

        return torch.cat(ret)
```

iterator와 for문을 사용하여  
pipelining 직접 구현

# MODEL PARALLELISM

## ■ IPU의 Model Pipelining

IPU의 Poplar SDK에서는 API를 통해 split만 나누어주면 pipelining 자동 실행

### 1) 모델 Layer를 직접 작성하는 경우

```
class Network(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = torch.nn.Linear(5, 10)
        self.layer2 = torch.nn.Linear(10, 5)
        self.layer3 = torch.nn.Linear(5, 5)
        self.layer4 = torch.nn.Linear(5, 5)

        self.act = torch.nn.ReLU()
        self.softmax = torch.nn.Softmax(dim=1)
```

```
def forward(self, x):

    # Explicit Layers on a certain IPU
    poptorch.Block.useAutoId()
    with poptorch.Block(ipu_id=0):
        x = self.act(self.layer1(x))

    with poptorch.Block(ipu_id=1):
        x = self.act(self.layer2(x))

    with poptorch.Block(ipu_id=2):
        x = self.act(self.layer3(x))
        x = self.act(self.layer4(x))

    with poptorch.Block(ipu_id=3):
        x = self.softmax(x)
    return x
```

Layer1 → IPU0

Layer2 → IPU1

Layer3, 4 → IPU2

Softmax → IPU3

# MODEL PARALLELISM

## ■ IPU의 Model Pipelining

IPU의 Poplar SDK에서는 API를 통해 split만 나누어주면 pipelining 자동 실행

2) 모델을 import하여 사용하는 경우

```
class WrappedModel(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.wrapped = transformers.BertForQuestionAnswering.from_pretrained(
            pretrained_weights)

# All Layers before "model.bert.encoder.Layer[0]" will be on IPU 0 and all Layers from
# "model.bert.encoder.Layer[0]" onwards (inclusive) will be on IPU 1.
model.bert.encoder.layer[0] = poptorch.BeginBlock(model.bert.encoder.layer[0],
                                                ipu_id=1)

# Now all Layers before Layer are on IPU 1 and this Layer onward is on IPU 2
model.bert.encoder.layer[2] = poptorch.BeginBlock(model.bert.encoder.layer[2],
                                                ipu_id=2)

# Finally all Layers from this Layer till the end of the network are on IPU 3.
model.bert.encoder.layer[4] = poptorch.BeginBlock(model.bert.encoder.layer[4],
                                                ipu_id=3)
```

기본으로 전체 → IPU0

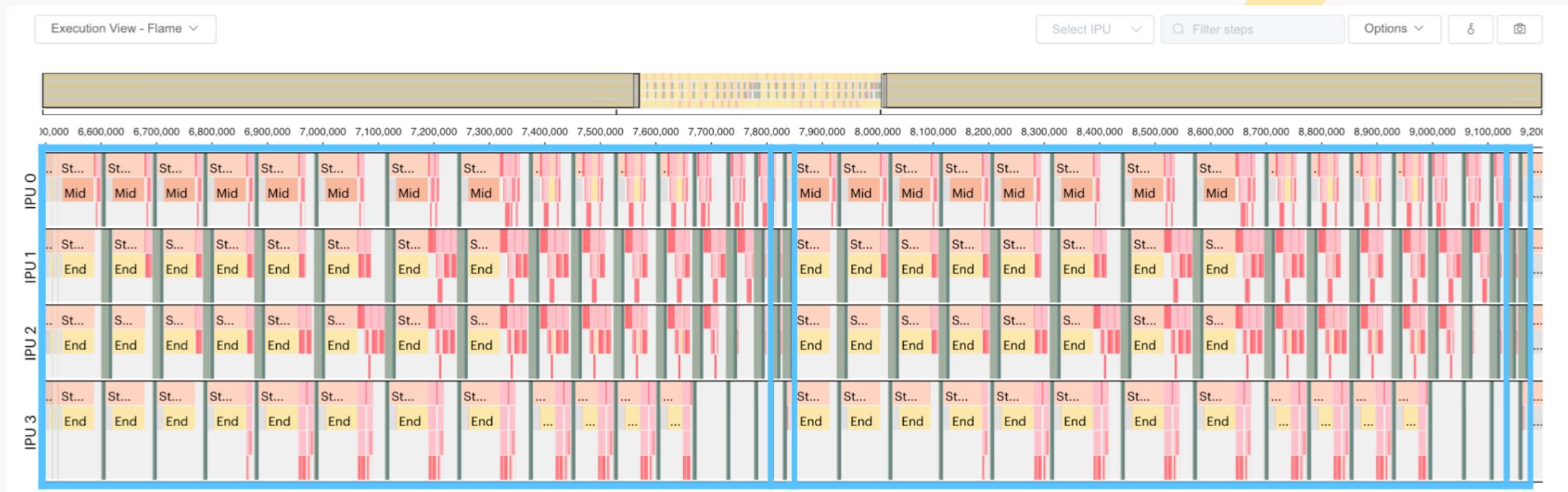
Layer0 부터 → IPU1

Layer2 부터 → IPU2

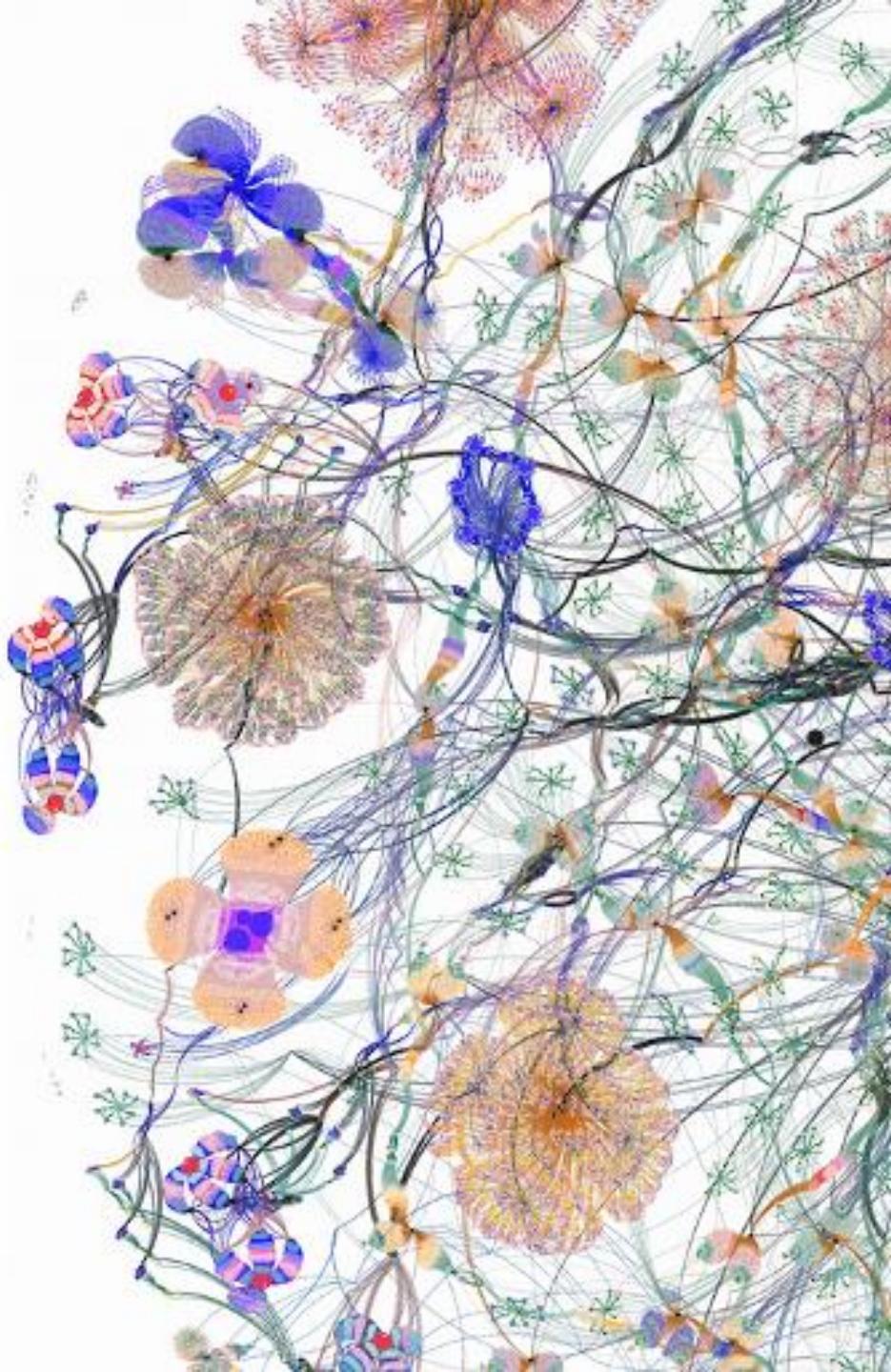
Layer4 부터 → IPU3

# MODEL PARALLELISM

## Pipelining Popvision 시작화



# HANDS-ON TUTORIAL



# GRAPHCORE COMMAND LINE TOOLS

- gc-monitor : 현재 파티션 내 ipu 하드웨어 정보 및 ipu 프로세스 확인

gc-monitor								
Partition: 'pt-minkyuk-16-ipus' has 16 reconfigurable IPUs								
IPU-M	Serial	ICU FW	Type	Server version	ID	PCIe ID	Routing	GWSW
10.1.5.21	0046.0002.8204521	2.4.4	M2000	1.9.0	0	3	DNC	2.4.2
10.1.5.21	0046.0002.8204521	2.4.4	M2000	1.9.0	1	2	DNC	2.4.2
10.1.5.21	0046.0001.8204521	2.4.4	M2000	1.9.0	2	1	DNC	2.4.2
10.1.5.21	0046.0001.8204521	2.4.4	M2000	1.9.0	3	0	DNC	2.4.2
10.1.5.22	0012.0002.8204521	2.4.4	M2000	1.9.0	4	3	DNC	2.4.2
10.1.5.22	0012.0002.8204521	2.4.4	M2000	1.9.0	5	2	DNC	2.4.2
10.1.5.22	0012.0001.8204521	2.4.4	M2000	1.9.0	6	1	DNC	2.4.2
10.1.5.22	0012.0001.8204521	2.4.4	M2000	1.9.0	7	0	DNC	2.4.2
10.1.5.23	0040.0002.8204521	2.4.4	M2000	1.9.0	8	3	DNC	2.4.2
10.1.5.23	0040.0002.8204521	2.4.4	M2000	1.9.0	9	2	DNC	2.4.2
10.1.5.23	0040.0001.8204521	2.4.4	M2000	1.9.0	10	1	DNC	2.4.2
10.1.5.23	0040.0001.8204521	2.4.4	M2000	1.9.0	11	0	DNC	2.4.2
10.1.5.24	0050.0002.8204521	2.4.4	M2000	1.9.0	12	3	DNC	2.4.2
10.1.5.24	0050.0002.8204521	2.4.4	M2000	1.9.0	13	2	DNC	2.4.2
10.1.5.24	0050.0001.8204521	2.4.4	M2000	1.9.0	14	1	DNC	2.4.2
10.1.5.24	0050.0001.8204521	2.4.4	M2000	1.9.0	15	0	DNC	2.4.2
No attached processes								

- vipu-admin list partitions : IPU-POD 내 파티션 정보 확인

# SETUP

## 1. Poplar SDK 활성화

- source SDK\_PATH/poplar\*/enable.sh
- source SDK\_PATH/popart\*/enable.sh

## 2. 가상환경 생성 및 활성화

- virtualenv -p python3 venv\_tutorial
- source venv\_tutorial/bin/activate

## 3. 패키지 설치

- (pytorch)pip install SDK\_PATH/poptorch\*.whl
- (tensorflow)pip install SDK\_PATH/tensorflow-2\*amd\*.whl (tf1/tf2, amd/intel 지정)

Package	Version
dataclasses	0.8
importlib-resources	5.4.0
pip	21.3.1
pkg_resources	0.0.0
poptorch	2.4.0+40669
setuptools	59.6.0
torch	1.10.0+cpu
tqdm	4.64.0
typing_extensions	4.1.1
wheel	0.37.1
zipp	3.6.0

poptorch 설치 시 호환되는 torch 버전 자동으로 함께 설치

```
(venv_tutorial) minkyuk@pod642:~/projects/NHN/hands_on_tutorials/day01$ ls ~/sdks/2.4.0/tensorflow-
tensorflow-1.15.5+gc2.4.0+139618+803bd5d97f3+amd_znver1-cp36-cp36m-linux_x86_64.whl      tensorflow-2.4.4+gc2.4.0+139613+8debb698097+amd_znver1-cp36-cp36m-linux_x86_64.whl
tensorflow-1.15.5+gc2.4.0+139620+803bd5d97f3+intel_skylake512-cp36-cp36m-linux_x86_64.whl  tensorflow-2.4.4+gc2.4.0+139619+8debb698097+intel_skylake512-cp36-cp36m-linux_x86_64.whl
```

# TUTORIAL

## Popvision Graph Analyzer



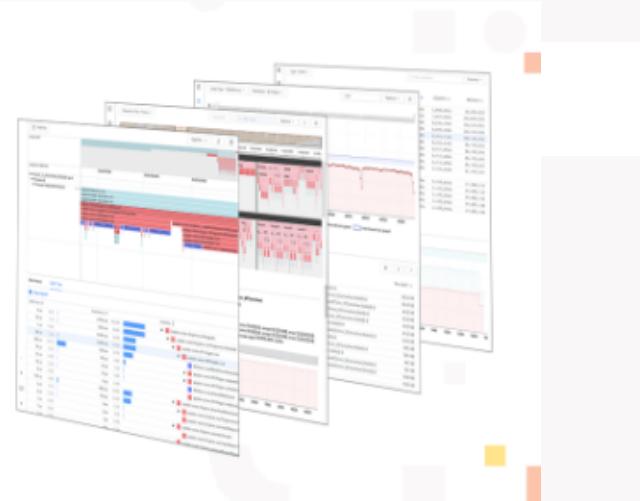
Home About ▾ Products ▾ Industries ▾ **Developer** ▾ Blog Careers ▾ Get Started →

Developer Tools Academic Documentation Performance Results Resources

### POPVISION TOOLS

Graphcore's PopVision™ family of analysis tools helps developers gain a deep understanding of IPU performance and utilisation. Visualise your code's inner workings with a user-friendly, graphical interface to optimise your machine learning models.

Download Now →



<https://www.graphcore.ai/developer/popvision-tools>

# TUTORIAL

## Popvision Graph Analyzer

```
POPLAR_ENGINE_OPTIONS='{"autoReport.all":"true", "autoReport.directory": "./DIR_NAME", "debug.allowOutOfMemory":"true"}'
```

예시

```
(venv_tutorial) minkyuk@pod642:~/projects/NHN/hands_on_tutorials/day02$ POPLAR_ENGINE_OPTIONS  
='{"autoReport.all":"true", "autoReport.directory": "./reports/resnet50_model_data_parallel",  
"debug.allowOutOfMemory":"true"}' python multi_ipu.py
```



```
(venv_tutorial) minkyuk@pod642:~/projects/NHN/hands_on_tutorials/day02$ tree reports/  
reports/  
└── resnet50  
    └── debug.cbor  
        └── training  
            └── archive.a  
            └── debug.cbor -> ../../debug.cbor  
            └── profile.pop  
            └── profile.pop_cache  
└── resnet50_model_data_parallel  
    └── debug.cbor  
        └── training  
            └── archive.a  
            └── debug.cbor -> ../../debug.cbor  
            └── profile.pop  
└── resnet50_model_parallel  
    └── debug.cbor  
        └── training  
            └── archive.a  
            └── debug.cbor -> ../../debug.cbor  
            └── profile.pop  
            └── profile.pop_cache
```

# TUTORIAL

- base.py

## Summary

Program Information	
<b>Target</b>	
Type	IPU
Architecture	Mk2
Timestamp	2022-05-30 19:25:29
Tiles per IPU	1472
Total Tiles	1472
Total IPUs	1
Memory per Tile	624 kB
Memory per IPU	897 MB
Total Memory	897 MB
Compute Set Instrumentation	Tile
External Exchange Instrumentation	Tile
<b>Graph</b>	
Number of compute sets	1,771
Number of edges	11,534,194
Number of variables	7,105,125
Number of vertices	3,518,939
<b>Poplar</b>	
Version	2.4.0 (10a96ee536)

# TUTORIAL

## ■ base.py

### Insights

Your application 'training' is within memory capacity

**801 MB/897 MB**  
89.28% of capacity

**Insights**

**Tiles with the highest memory usage**  
These are the top 5 tiles with the highest memory consumption

Max Memory	Memory Required	IPU
Tile 552	605 kB	IPU O
Tile 522	605 kB	IPU O
Tile 480	605 kB	IPU O
Tile 482	604 kB	IPU O

**Tiles Histogram**  
This is the distribution of memory on the tiles

**Reducing memory usage**

You can reduce the memory requirements of your application in the following ways. More information on these can be found in the [Graphcore Memory and Performance Optimisation Guide](#)

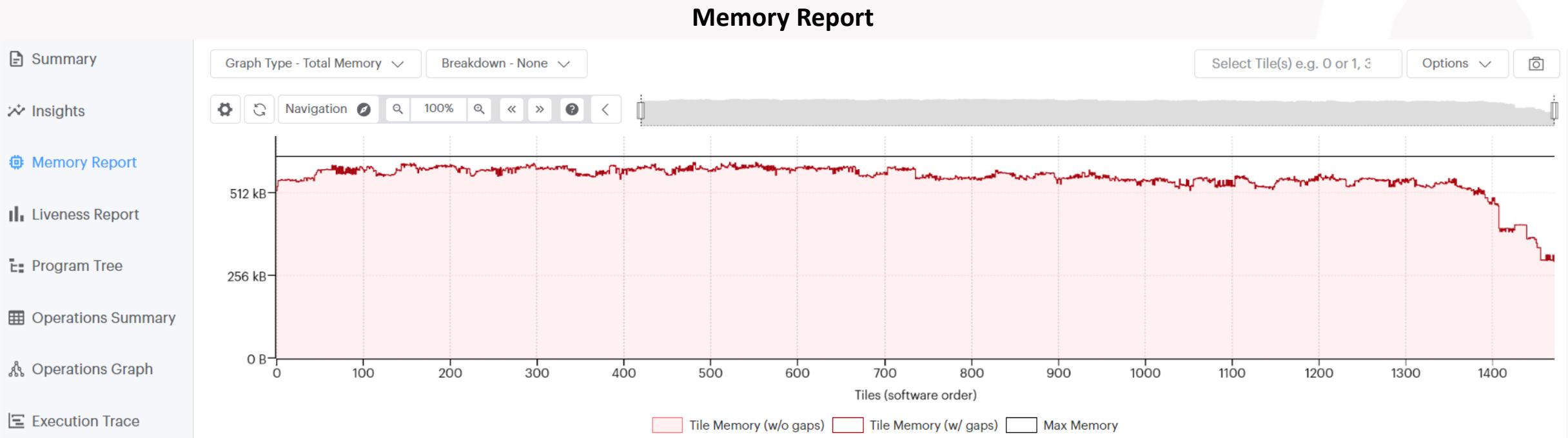
**These solutions may affect the performance, throughput, convergence and/or training characteristics of your model**

[Reducing the batch size](#)   [Using FP16](#)   [Using recomputation](#)   [Using recomputation checkpoints](#)   [Setting the available memory proportion](#)

[Offloading variables](#)   [Reusing identical parts of your graph](#)   [Disable profiling on the IPU](#)

# TUTORIAL

- base.py



# TUTORIAL

## ■ base.py

### Liveness Report

Summary

Insights

Memory Report

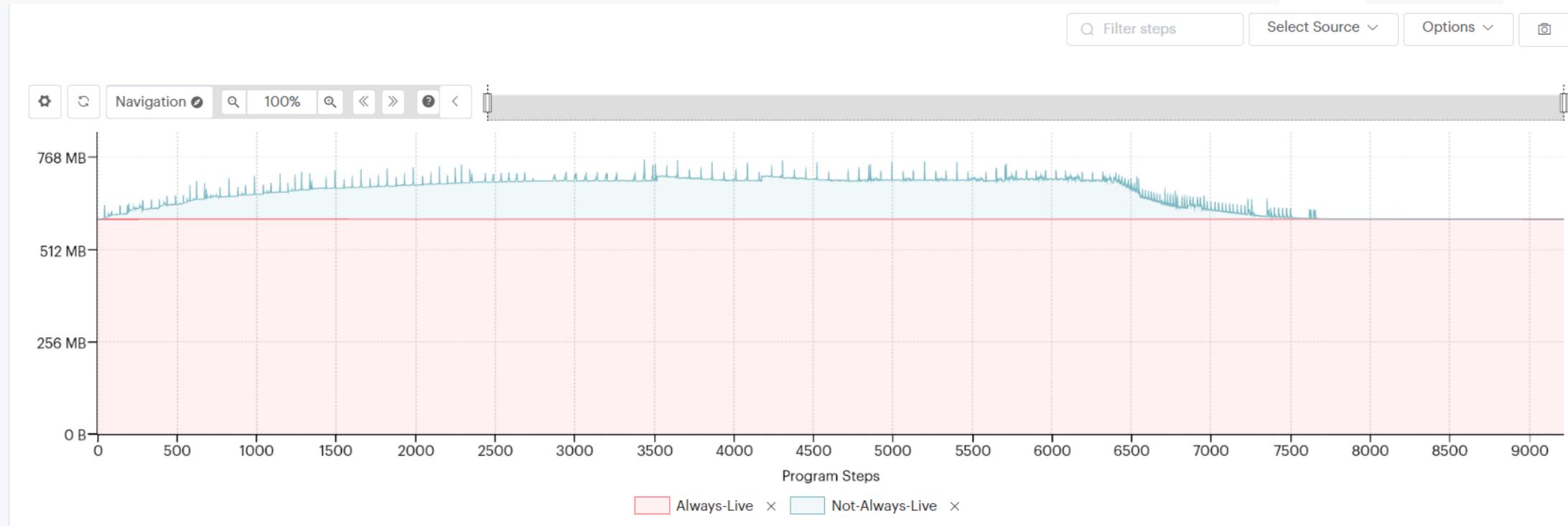
Liveness Report

Program Tree

Operations Summary

Operations Graph

Execution Trace

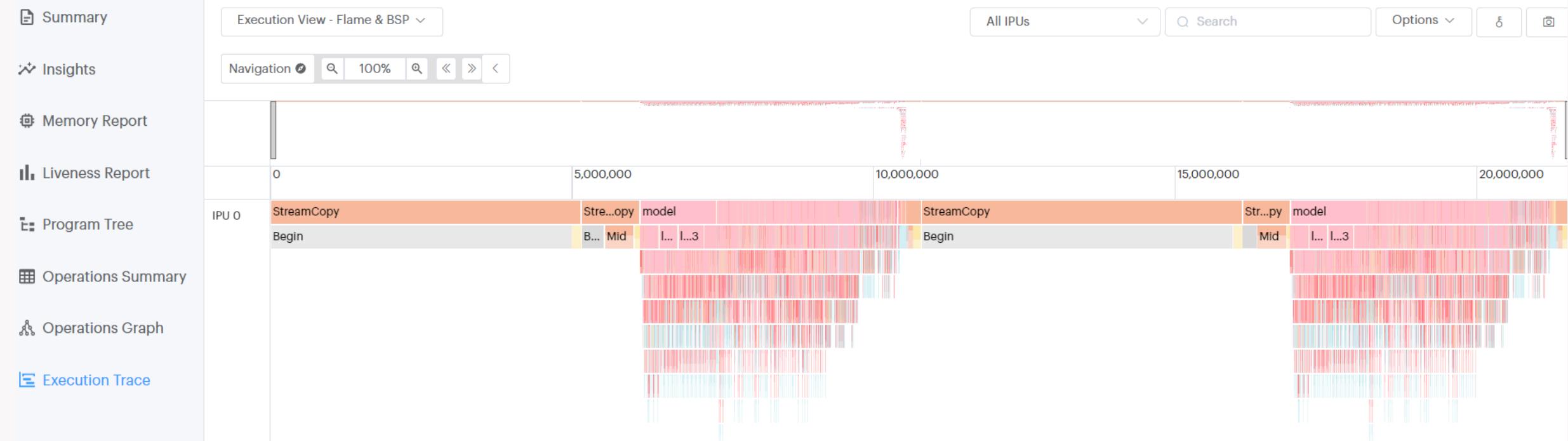


- Always-Live : 그래프 연산의 lifetime 전체에 걸쳐 메모리를 점유하는 변수. Ex) code, constants
- Not-Always-Live : 특정 프로그램 step에서만 사용되는 변수.

# TUTORIAL

- base.py

## Execution Trace



# TUTORIAL

## ■ Model Parallel

```
56 |     opts = poptorch.Options()
57 |
58 |     model = torchvision.models.resnet50()
59 |
60 |     model.layer4 = poptorch.BeginBlock(model.layer4, ipu_id=1)
61+
```

Summary

Insights

Memory Report

Liveness Report

Program Tree

Operations Summary

Operations Graph

Execution Trace

### Program Information

#### Target

Type	IPU
Architecture	Mk2
Timestamp	2022-05-30 19:18:02

#### Tiles per IPU

Total Tiles	2944
Total IPUs	2

Memory per Tile	624 kB
Memory per IPU	897 MB
Total Memory	1,794 MB

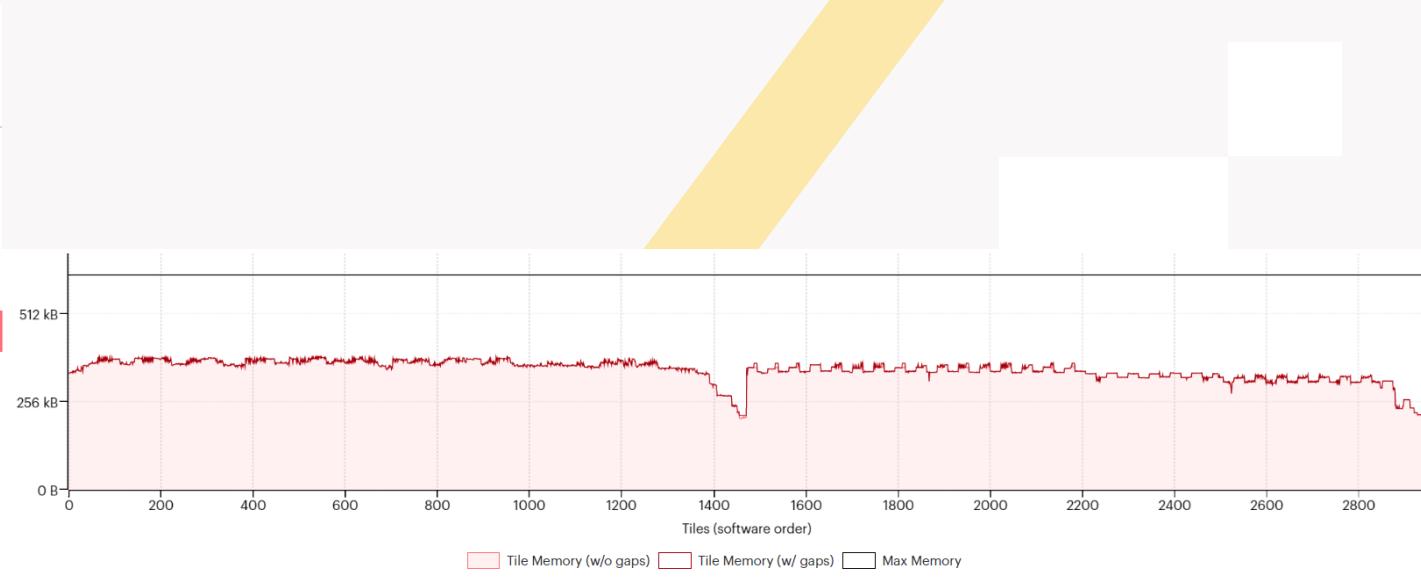
Compute Set Instrumentation	Tile
External Exchange Instrumentation	Tile

#### Graph

Number of compute sets	1,994
Number of edges	12,866,597
Number of variables	7,857,014
Number of vertices	4,163,455

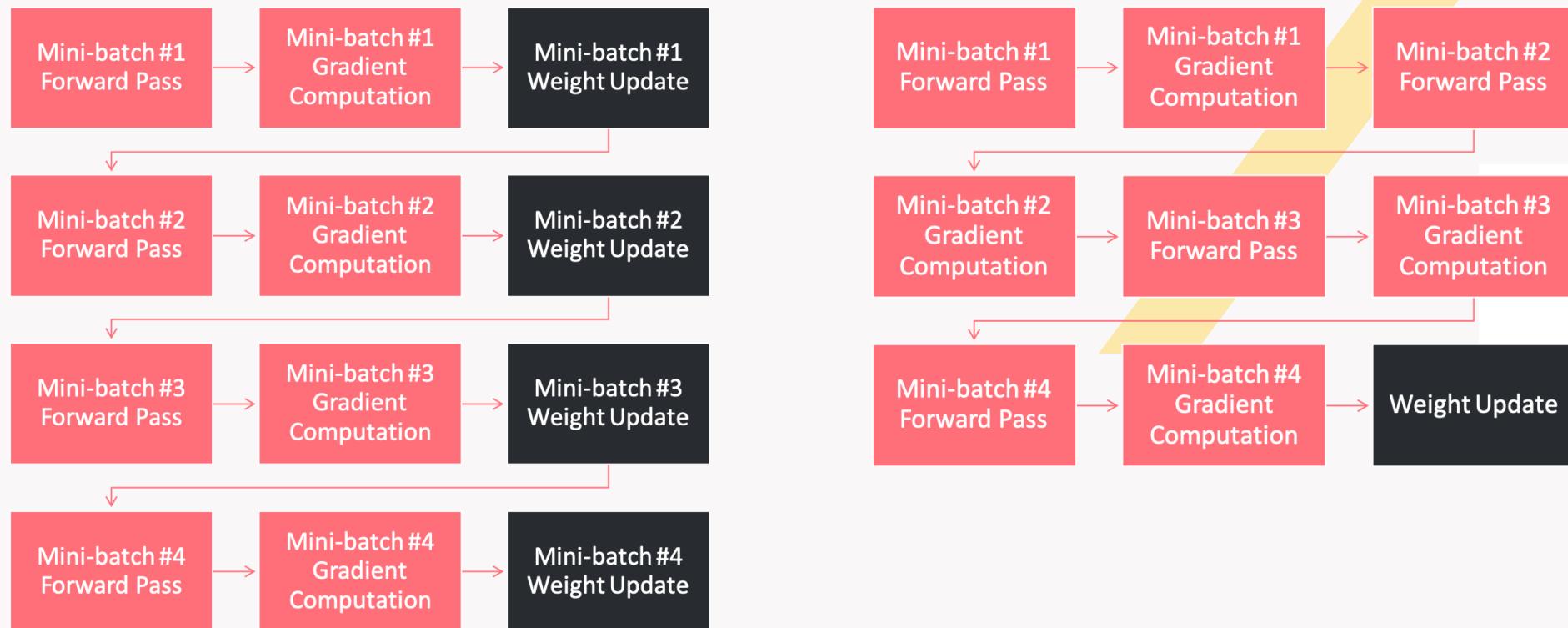
#### Poplar

Version	2.4.0 (10a96ee536)
---------	--------------------



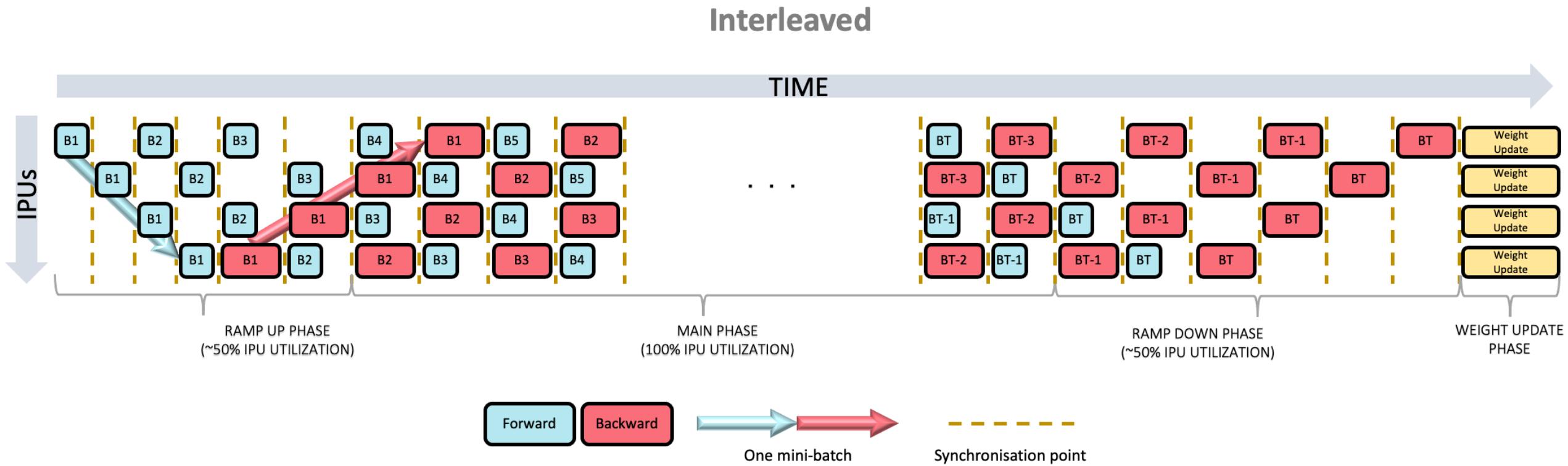
# GRADIENT ACCUMULATION

- 한 번의 weight update에 더 많은 샘플 사용
- 메모리 사용은 유지하면서 배치사이즈 늘리는 효과



# GRADIENT ACCUMULATION

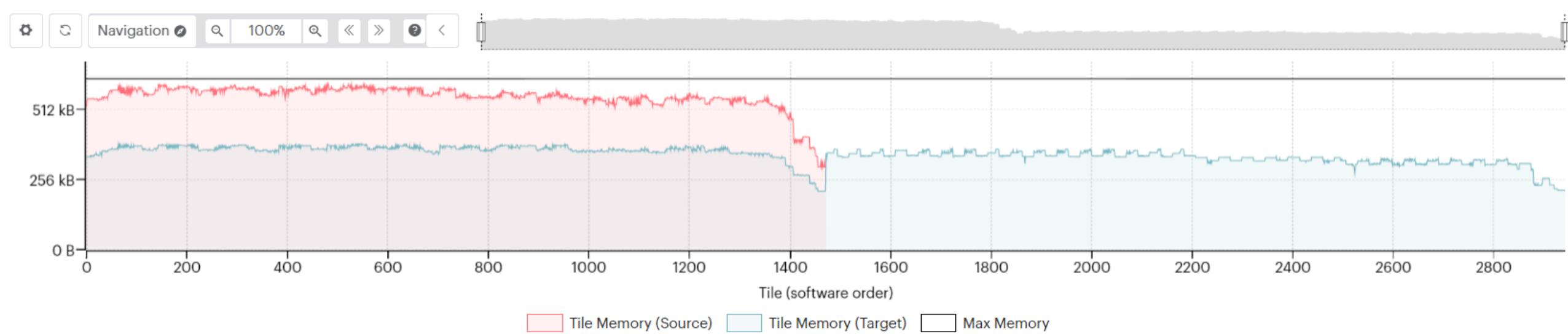
- Pipelining 시 main execution 의 비율 확대 → 속도 향상



# TUTORIAL

- Model Parallel

File > compare two reports



# TUTORIAL

## ■ Model Parallel + Data Parallel

```
opts = poptorch.Options()
      56  opts = poptorch.Options()
      57+ opts.Training.gradientAccumulation(3)
      58+ opts.replicationFactor(2)

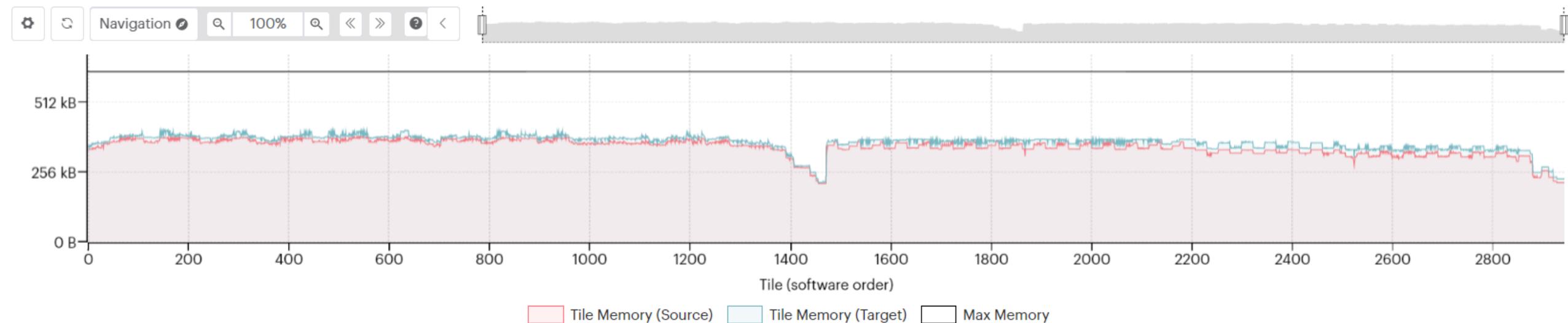
model = torchvision.models.resnet50()
      59
      60  model = torchvision.models.resnet50()
      61+ model.layer4 = poptorch.BeginBlock(model.layer4, ipu_id=1)
```

Target	
Type	IPU
Architecture	Mk2
Timestamp	2022-05-30 20:11:37
Tiles per IPU	1472
IPUs per Replica	2
Replicas	2
Total Tiles	5888
Total IPUs	4
Memory per Tile	624 kB
Memory per IPU	897 MB
Memory per Replica	1,794 MB
Total Memory	3,588 MB
Compute Set Instrumentation	Tile
External Exchange Instrumentation	Tile
Graph	
Number of compute sets	2,216
Number of edges	13,208,391
Number of variables	8,832,650
Number of vertices	4,335,377
Poplar	
Version	2.4.0 (10a96ee536)

# TUTORIAL

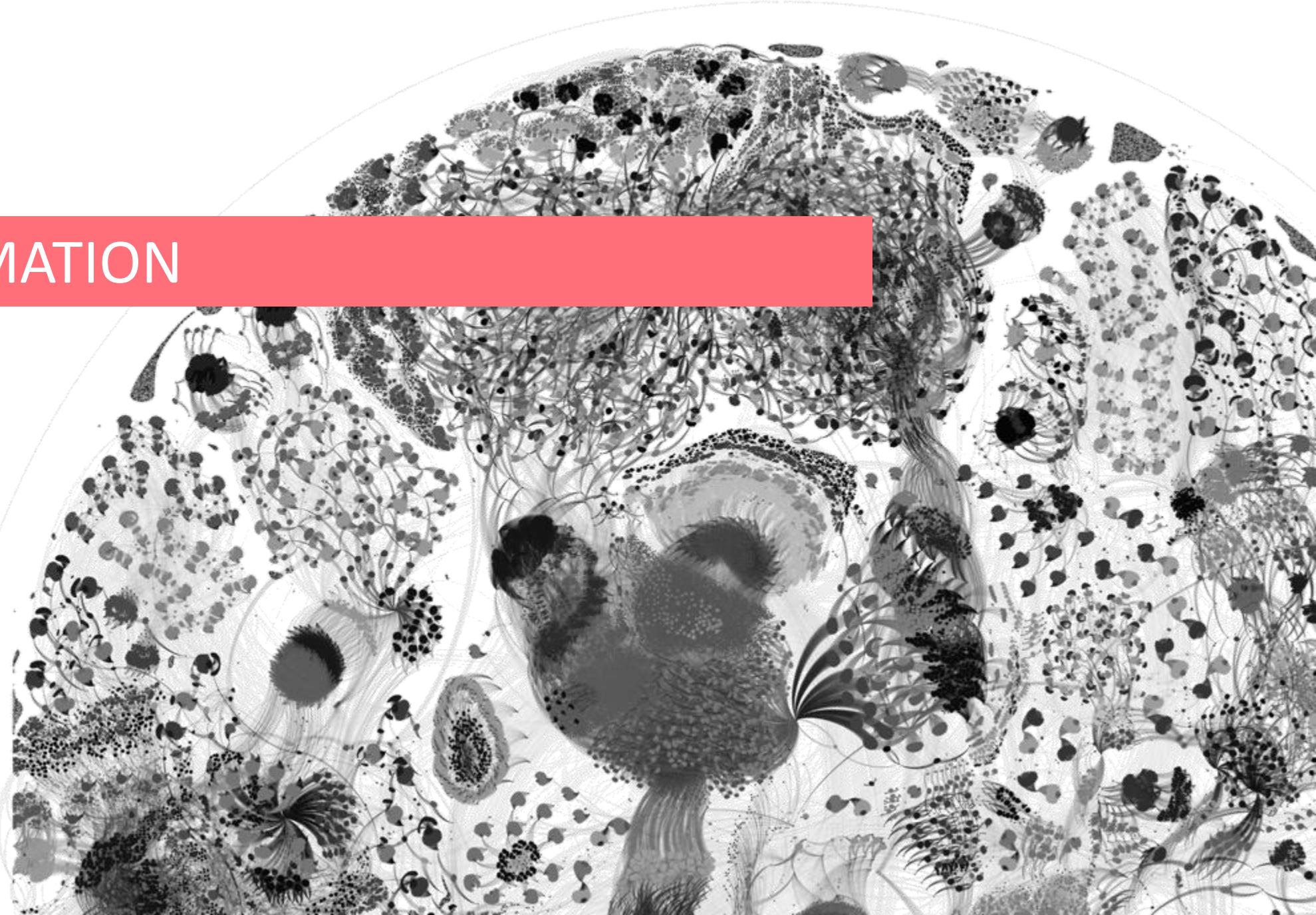


- Model Parallel + Data Parallel



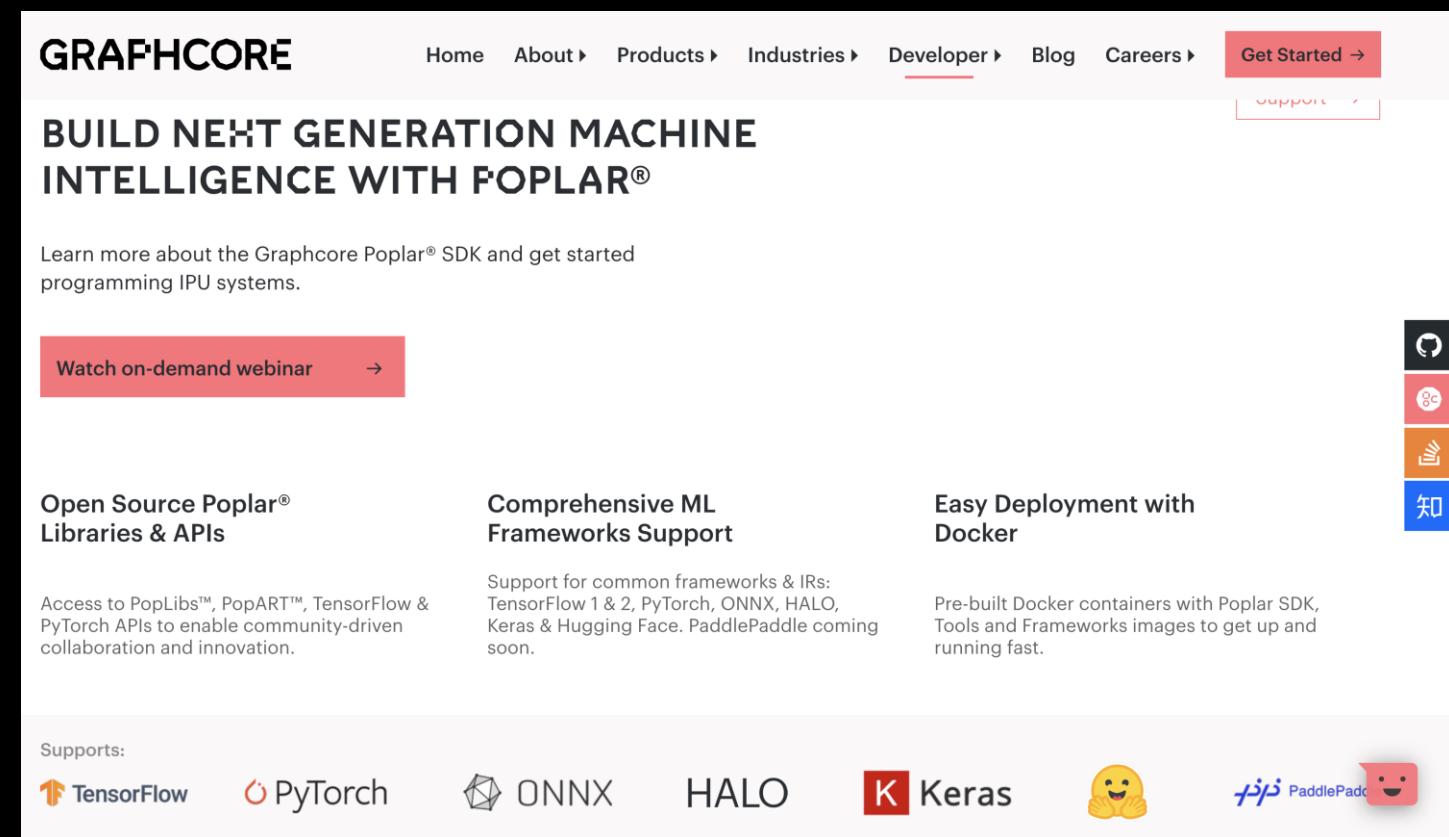
Graph	Model Parallel	Model + Data Parallel
Number of compute sets	1,994	2,216 (222)
Number of edges	12,866,597	13,208,391 (341,794)
Number of variables	7,857,014	8,832,650 (975,636)
Number of vertices	4,163,455	4,335,377 (171,922)

MORE INFORMATION



# DEVELOPER PORTAL

<https://graphcore.ai/developer>



The screenshot shows the Graphcore Developer Portal homepage. At the top, there's a navigation bar with links: Home, About ▾, Products ▾, Industries ▾, **Developer ▾**, Blog, Careers ▾, and Get Started ▾. Below the navigation is a large header with the text "GRAPHCORE" and "BUILD NEXT GENERATION MACHINE INTELLIGENCE WITH POPLAR®". A sub-header below it says "Learn more about the Graphcore Poplar® SDK and get started programming IPU systems." There's a red button labeled "Watch on-demand webinar →". To the right of the main content area, there are three columns: "Open Source Poplar® Libraries & APIs", "Comprehensive ML Frameworks Support", and "Easy Deployment with Docker". Each column has a brief description and some icons. At the bottom, there's a "Supports:" section with logos for TensorFlow, PyTorch, ONNX, HALO, Keras, and PaddlePaddle, along with their respective smiley face icons.

**GRAPHCORE**

Home About ▾ Products ▾ Industries ▾ **Developer ▾** Blog Careers ▾ Get Started ▾

BUILD NEXT GENERATION MACHINE INTELLIGENCE WITH POPLAR®

Learn more about the Graphcore Poplar® SDK and get started programming IPU systems.

Watch on-demand webinar →

Open Source Poplar® Libraries & APIs

Access to PopLibs™, PopART™, TensorFlow & PyTorch APIs to enable community-driven collaboration and innovation.

Comprehensive ML Frameworks Support

Support for common frameworks & IRs: TensorFlow 1 & 2, PyTorch, ONNX, HALO, Keras & Hugging Face. PaddlePaddle coming soon.

Easy Deployment with Docker

Pre-built Docker containers with Poplar SDK, Tools and Frameworks images to get up and running fast.

Supports:

TensorFlow PyTorch ONNX HALO Keras PaddlePaddle

# GRAPHCORE DOCUMENTS

<https://docs.graphcore.ai>



The screenshot shows a dark-themed documentation site for Graphcore. At the top, the word "GRAPHCORE" is written in large, bold, white capital letters. Below it, a dark sidebar contains the text "Graphcore Documents" and "Version: latest". A "Search docs" input field is also present. The main content area lists several categories: "Software Documents", "Hardware Documents", "Examples and Tutorials", "Document Updates", and "Graphcore End User License Agreement". At the bottom of the page, there are links for "Read the Docs" and a dropdown menu showing "v: latest". To the right of the screenshot, a vertical list of document types is displayed.

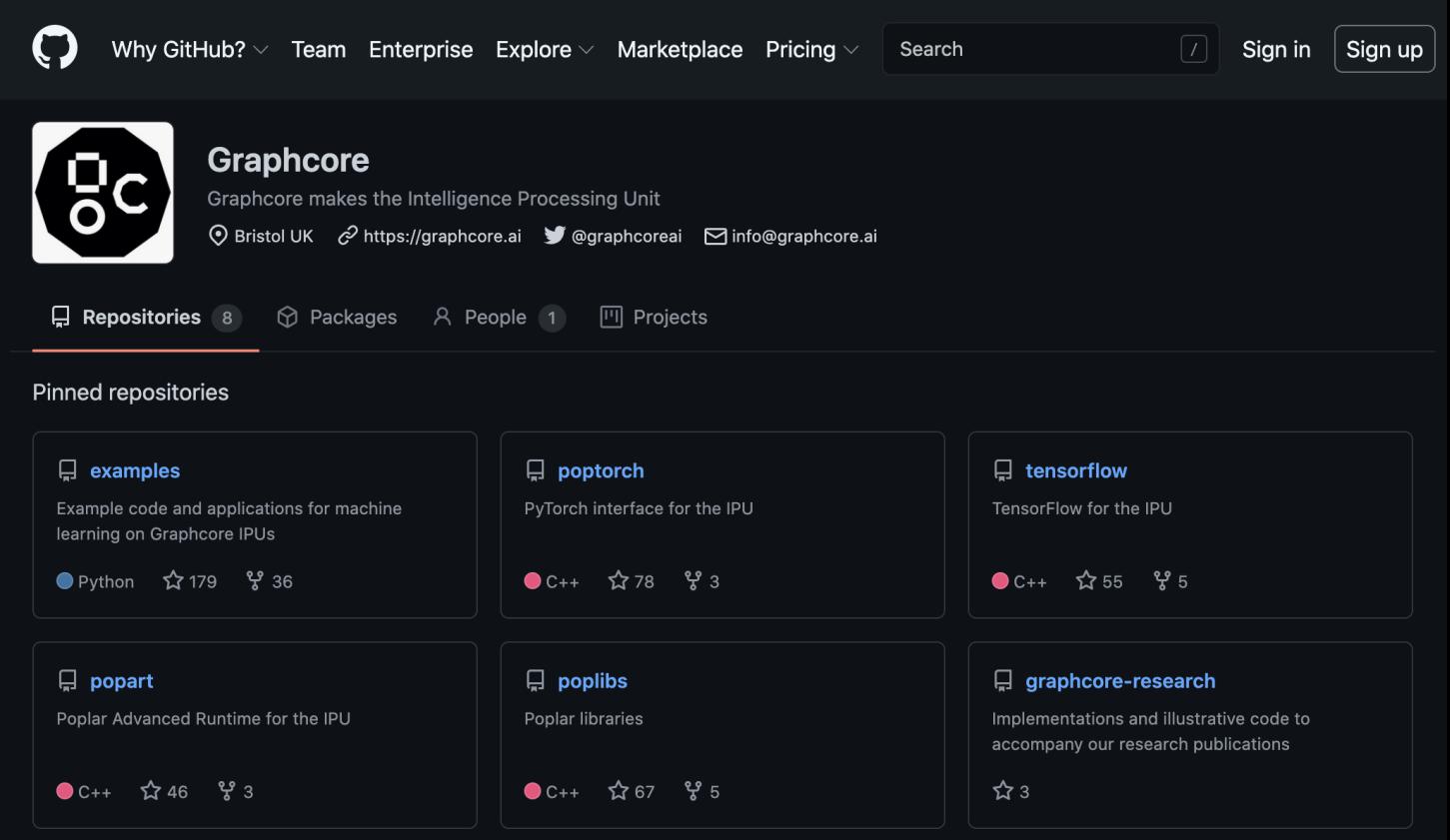
## GRAPHCORE DOCUMENTS

- Software Documents
  - Getting Started
  - TensorFlow
  - PyTorch
  - PopART
  - Poplar Graph Programming Framework
  - Technical Notes
  - Profiling and Debugging
  - Open Source Software
  - License Agreements
- Hardware Documents
  - Getting Started
  - IPU-POD Systems
  - Supporting Tools
- Examples and Tutorials
  - Simple examples
  - Application examples
  - Tutorials
- Document Updates
  - April 2021
  - March 2021
  - November 2020
- Graphcore End User License Agreement



# GRAPHCORE GITHUB

<https://github.com/graphcore>



The screenshot shows the GitHub profile page for the organization "Graphcore". The header includes the GitHub logo, navigation links for "Why GitHub?", "Team", "Enterprise", "Explore", "Marketplace", "Pricing", a search bar, and "Sign in/Sign up" buttons. The main content features the organization's logo (a black octagon with white letters "GC"), its name "Graphcore", a tagline "Graphcore makes the Intelligence Processing Unit", and location "Bristol UK". It also lists links to their website and social media. Below this, there are tabs for "Repositories" (selected), "Packages", "People", and "Projects". A section titled "Pinned repositories" displays six repositories: "examples" (Python, 179 stars, 36 forks), "poptorch" (C++, 78 stars, 3 forks), "tensorflow" (C++, 55 stars, 5 forks), "popart" (C++, 46 stars, 3 forks), "poplibs" (C++, 67 stars, 5 forks), and "graphcore-research" (3 stars). Each repository card includes a brief description and a star icon.



# THANK YOU

Minkyu Kim

minkyuk@graphcore.ai