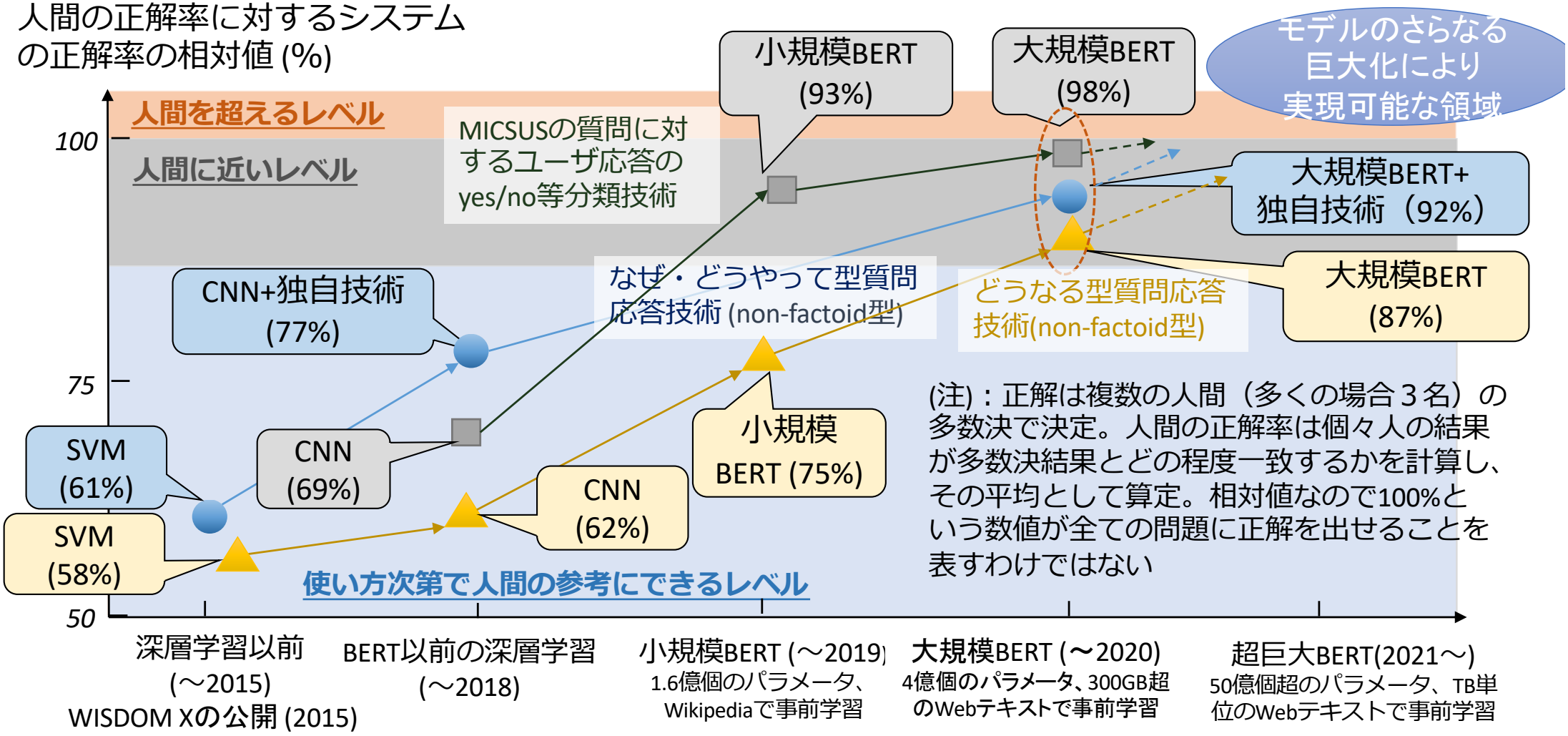
超大規模ニューラルネットワークのための自動並列化深層学習ミドルウェア RaNNC

国立研究開発法人 情報通信研究機構
ユニバーサルコミュニケーション研究所
データ駆動知能システム研究センター (DIRECT)

巨大言語モデル導入による精度推移

- 巨大言語モデル（BERT等）で質問応答等自然言語処理の精度を格段に向上



自動並列化深層学習ミドルウェアRaNNC

- ニューラルネットワークや学習コーパスの規模は急速に拡大し、新たなネットワークアーキテクチャも次々提案

- **大規模ニューラルネットワークは、学習パラメータがGPUのメモリに収まらないため、ネットワークを分割して並列に学習させる必要**
  - 多くの既存の大規模ネットワーク学習フレームワークは、各ネットワークごとに分割を開発者が実装、最適化する必要があり、利用のハードルが高い
  - 代表的な大規模ネットワーク学習フレームワークであるMegatron-LMやMesh-TensorFlowは、BERT等のTransformer系ネットワークにしか使えない

多様な大規模ニューラルネットワークを自動的に分割し、並列に学習できるミドルウェアが望ましい
→ RaNNC (Rapid Neural Net Connector) を開発

自動並列化深層学習ミドルウェアRaNNC

- **GitHubで公開中（MITライセンス）**

- IPDPS (35th IEEE International Parallel & Distributed Processing Symposium) で発表

Automatic Graph Partitioning for Very Large-scale Deep Learning, Masahiro Tanaka, Kenjiro Taura, Toshihiro Hanawa and Kentaro Torisawa, In *the Proceedings of 35th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2021)*, pp. 1004-1013, May, 2021.

PreprintはarXivにも掲載（https://arxiv.org/abs/2103.16063）

- GTC 2021 で発表

学習コード例

PyTorchが提供する演算モジュールでネットワークを定義

```
class BertSelfAttention(nn.Module):
...
        self.query = nn.Linear(...)
        self.key = nn.Linear(...)
        self.value = nn.Linear(...)
```

PyTorchによる
一般的な学習コード例

```
model = Net()                                  # ネットワークの定義
model.to(torch.device("cuda")) # ネットワークをGPUに移動
optimizer = optim.Adam(model.parameters(), lr=0.01)
                                               # Optimizerを定義
loss = model(input)                            # forwardを計算
loss.backward()                                # backwardを計算
optimizer.step()                               # パラメータ更新
```

学習コード例

- Megatron-LM[Shoeybi 2019]など既存の大規模ニューラルネットワーク用フレームワークを用いてモデルパラレルを実現するには、**複雑なネットワーク定義全体に渡って大幅な書き換えが必要**

ネットワークの定義は大幅に変更が必要

```
class ParallelSelfAttention(MegatronModule):
...
    self.query_key_value = mpu.ColumnParallelLinear(…)
    self.dense = mpu.RowParallelLinear(…)
```

- 分散計算用の特別な計算モジュールを使用
- どの箇所でどのような分散計算を使用するかは、**開発者が設計・最適化する必要**

```
model = Net()                          # ネットワークの定義
model.to(torch.device(“cuda”)) # ネットワークをGPUに移動
optimizer = optim.Adam(model.parameters(), lr=0.01)
                                       # Optimizerを定義
loss = model(input)                    # forwardを計算
loss.backward()                        # backwardを計算
optimizer.step()                       # パラメータ更新
```

学習コード例

- データパラレルによる学習は、PyTorchの標準機能でも容易

ネットワーク定義に変更なし

```
class BertSelfAttention(nn.Module):
...
        self.query = nn.Linear(...)
        self.key = nn.Linear(...)
        self.value = nn.Linear(...)
```

PyTorchによる
データパラレルの
学習コード例

```
model = Net()                                  # ネットワークの定義
model.to(torch.device("cuda")) # ネットワークをGPUに移動
optimizer = optim.Adam(model.parameters(), lr=0.01)
                                               # Optimizerを定義
# データパラレルのための指定
model = torch.nn.parallel.DistributedDataParallel(model)
loss = model(input)                            # forwardを計算
loss.backward()                                # backwardを計算
optimizer.step()                               # パラメータ更新
```

実行時はMPIを使用し、上記のプログラムのプロセスをGPU数だけ起動

学習コード例

- RaNNCは、ネットワーク定義の変更無しで、**自動で賢く並列学習**

ネットワーク定義に変更なし

```
class BertSelfAttention(nn.Module):
...
        self.query = nn.Linear(...)
        self.key = nn.Linear(...)
        self.value = nn.Linear(...)
```

RaNNCの使用

```
model = Net()                                  # ネットワークの定義
model.to(torch.device("cuda")) # ネットワークをGPUに移動
optimizer = optim.Adam(model.parameters(), lr=0.01)
                                               # Optimizerを定義
model = torch.nn.parallel.DistributedDataParallel(model)
        pyrannc.RaNNCModule(model, optimizer)
loss = model(input)                            # forwardを計算
loss.backward()                                # backwardを計算
optimizer.step()                               # パラメータ更新
```

GPUに配置するデータはRaNNCで管理

データパラレル同様に1行追加するだけ、GPUメモリ量などは自動認識

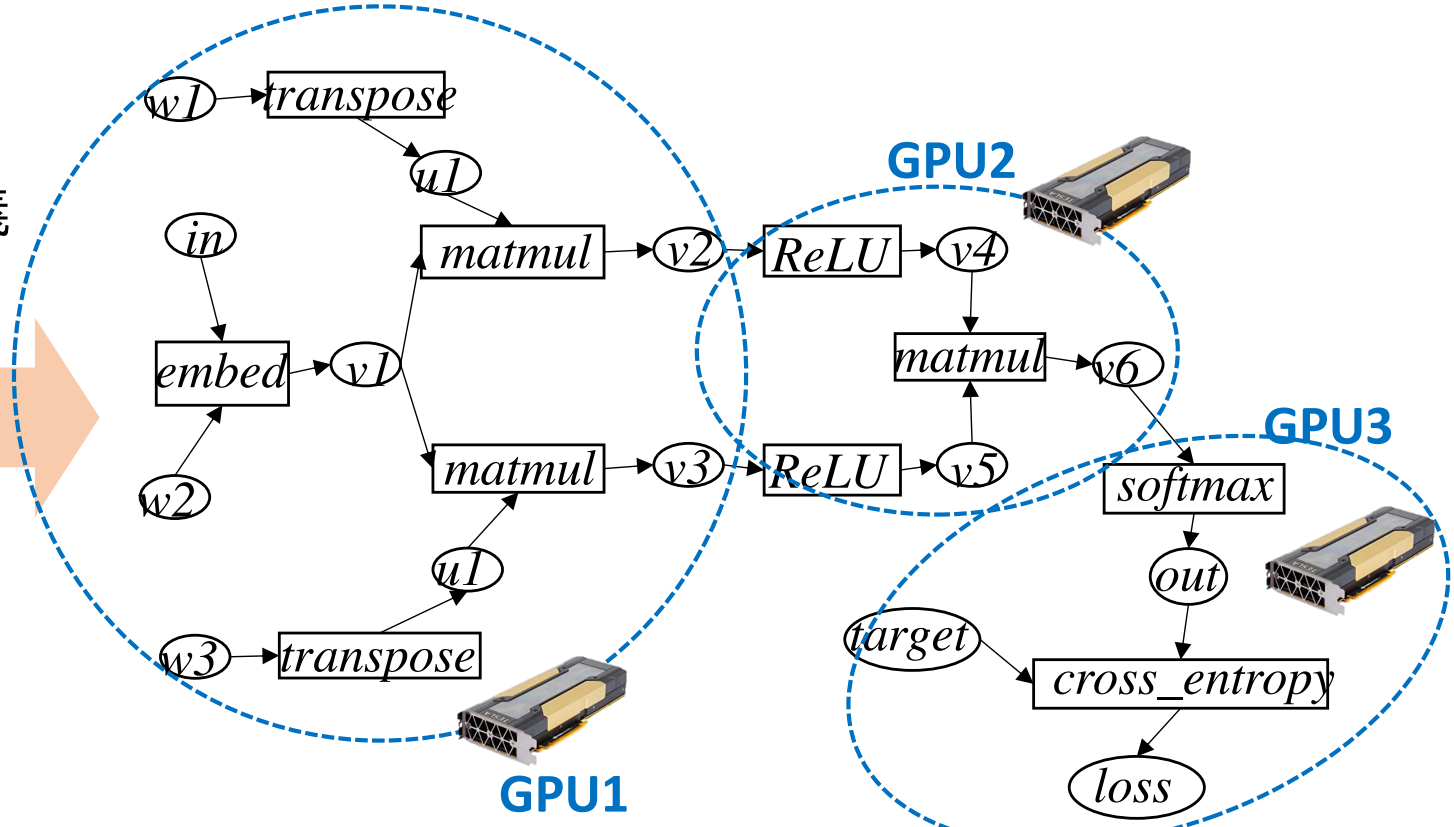
データパラレルと同様、MPIでプロセスをGPU数だけ起動

RaNNCの動作概要

- ニューラルネットワークの断片のプロファイルを取りつつ、**高速な並列学習が可能な分割を自動的に決定**
- 分割された断片をGPUに割り当て
- 分割された断片ごとに、データパラレルの並列度を自動最適化

PyTorchのネットワーク定義

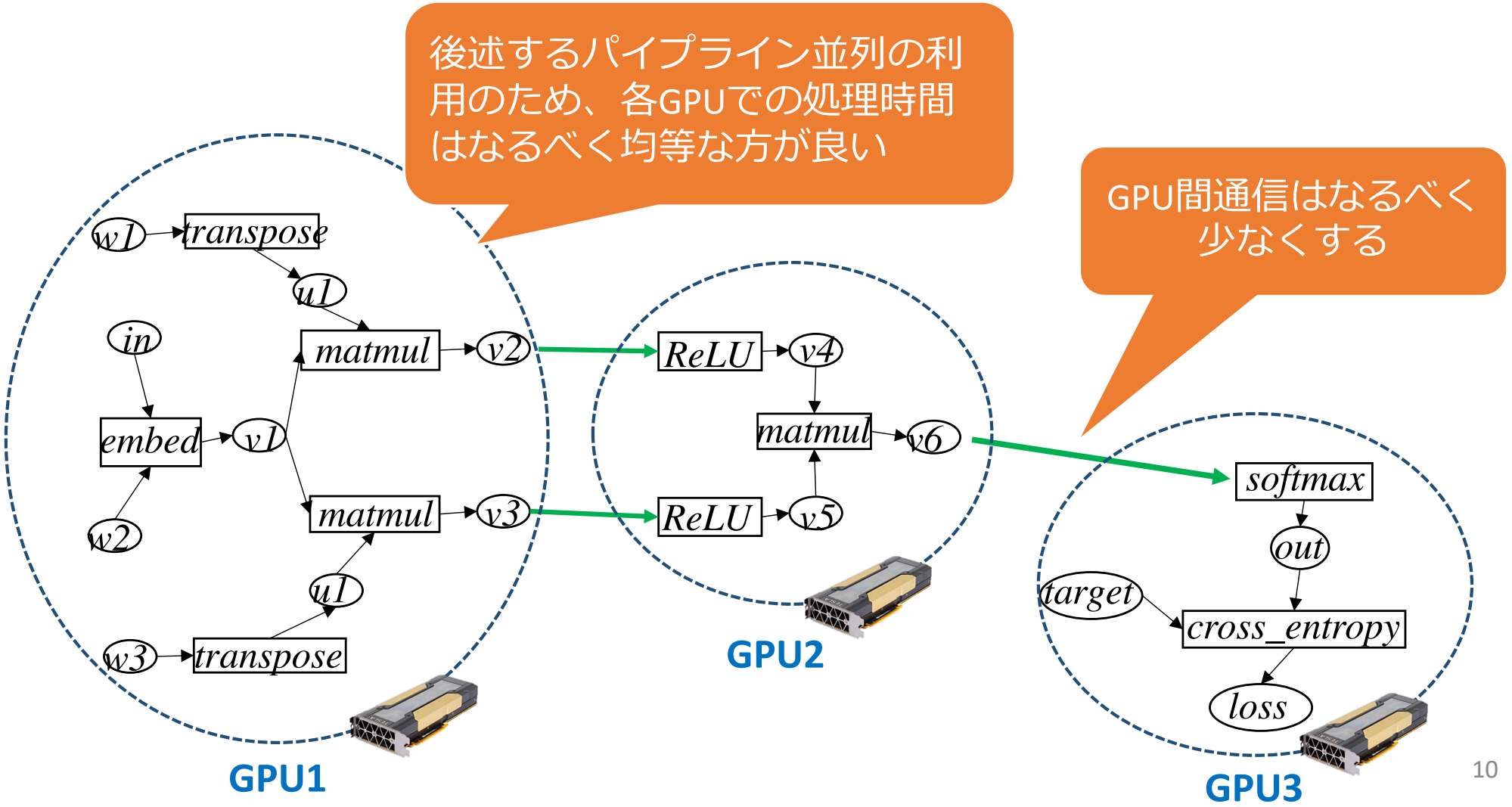
```
def BERTModel(…):
  def forward(…):
      v1 = self.embed(in)
      v2 = self.linear(v1)
…
```



ネットワークの断片のGPUへの割り当て

RaNNCの動作概要

- ニューラルネットワークの各断片は、GPUメモリに収まる必要がある
- その上で、速度を最適化するように分割



類似フレームワークとの比較

- 以下に挙げる類似フレームワークと処理速度を比較

分割されたネットワークで同期されたパラメータが使われるかどうか（Noの場合、学習精度低下の可能性）

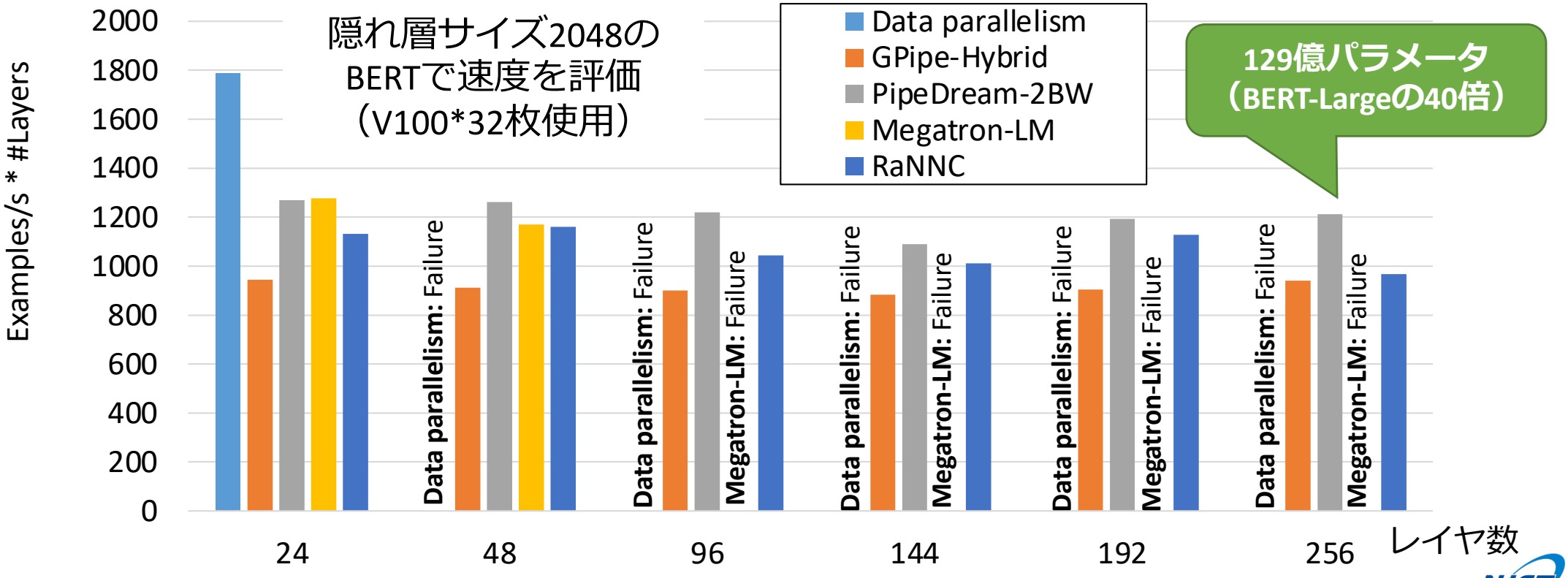
	自動分割	各種のネットワークに適用可能	Parameter staleness-free
Megatron-LM [Shoeybi 2019]	**No**	**No** (BERTなどTransformer系のみ)	**Yes**
GPipe[Huang 2018]*	**No**	**Yes**	**Yes**
PipeDream-2BW [Narayanan 2020]	**一部のみ自動化** (ネットワーク定義の書き換えが必要)	**Yes**	**No**
RaNNC	**Yes**	**Yes**	**Yes**

*GPipeは、オリジナルの実装がTensorFlowを使用し、またデータパラレルとの併用が不可能なため、以降に示す比較実験ではPyTorchを用いた以下の2つの実装を使用

- GPipe-Hybrid: データパラレルとの併用可能、複数ノード利用可能、実装上の問題で実質的にBERTにのみ適用可能（PipeDream-2BWの著者らが開発）
- GPipe-Model: データパラレルとの併用不可、複数ノード利用不可、各種ネットワークに適用可能（[Kim 2020]で開発）

大規模BERTモデルの学習速度

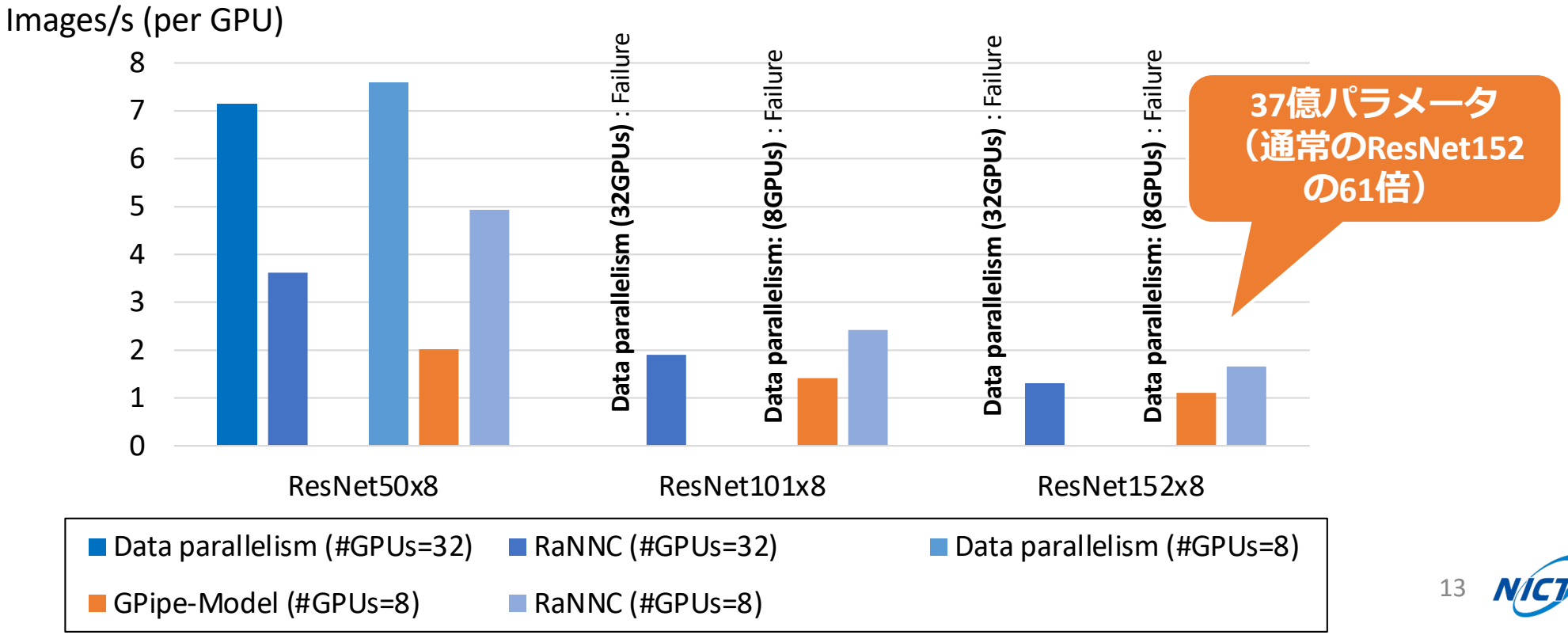
- Megatron-LMとの比較で、**同じ枚数のGPUでRaNNCは5倍大きなネットワークを計算可能***（比較可能なネットワークではほぼ同等の速度）
  - 分散処理のためのメモリ使用効率が良いため
- PipeDream-2BWは幾分RaNNCより速いが、parameter stalenessによる精度低下の恐れ（ネットワーク定義書き換えも必要）



*Megatron-LMのパイプライン並列は使用せず（実験時には実装されていなかったため）

大規模ResNetモデルの学習速度

- Megatron-LM, Mesh-TensorFlow[Shazeer 2018]などはBERT含むTransformer系ネットワークにしか適用できないが、**RaNNCは基本的にネットワークアーキテクチャに関して制限無し**
  → ResNet等、画像系ネットワークにも適用可能
- データパラレルでは学習不可能な規模のネットワークにおいて、GPipeより顕著に高速に動作



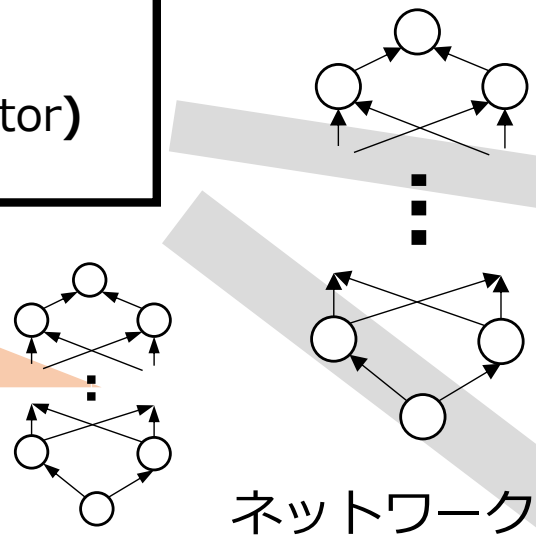
RaNNCの構成

処理時間やメモリのプロファイルに基づき、**ネットワーク分割を決定**

バックエンドにPyTorchを使用

分散深層学習フレームワーク
RaNNC
(Rapid Neural Network Connector)

ニューラルネットワークを分析しやすい形式に変換



ネットワークの断片

アダプタ

PyTorch

GPUサーバ

PyTorch

NVLink or InfiniBand

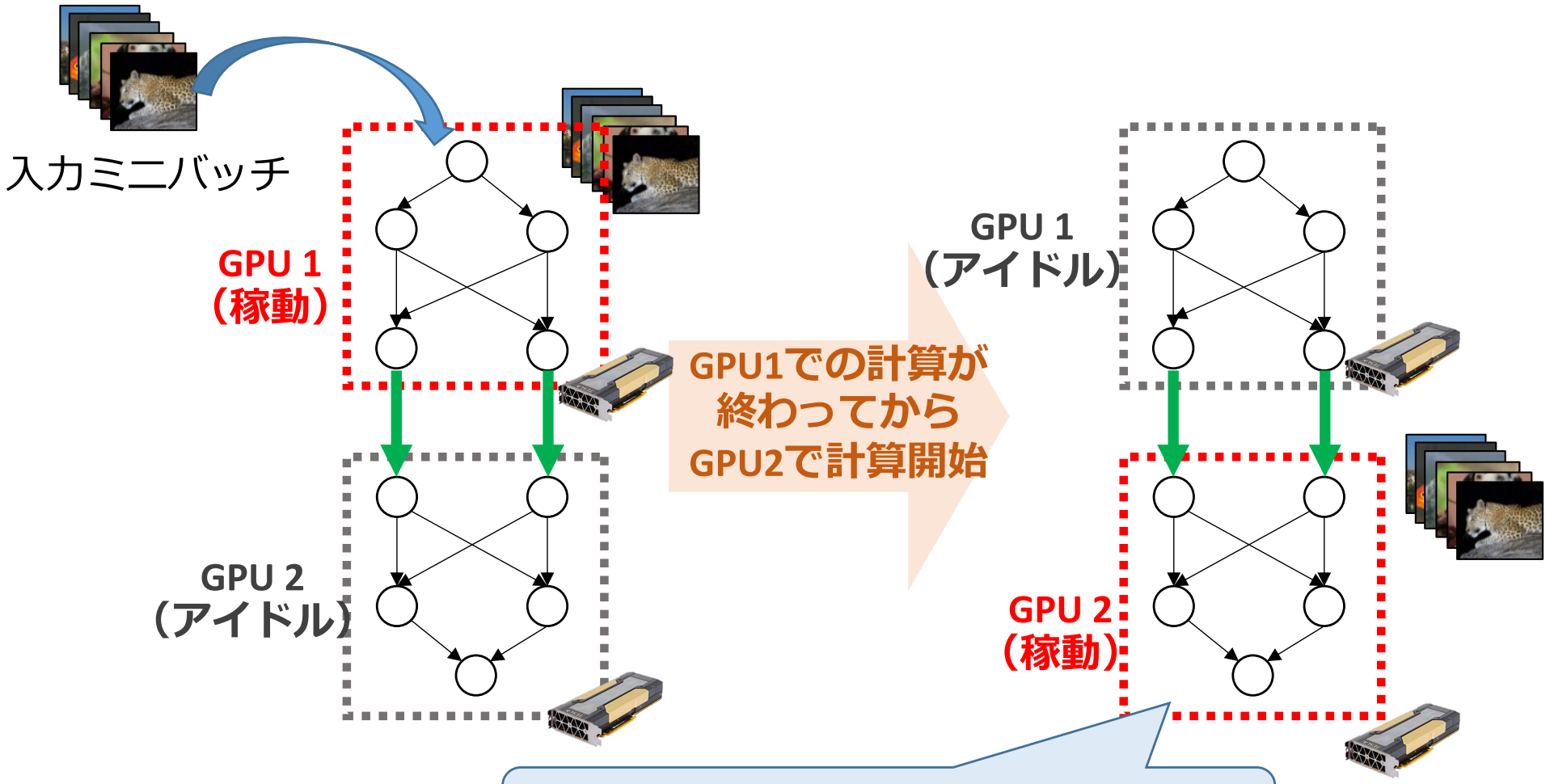
PyTorch

GPUサーバ

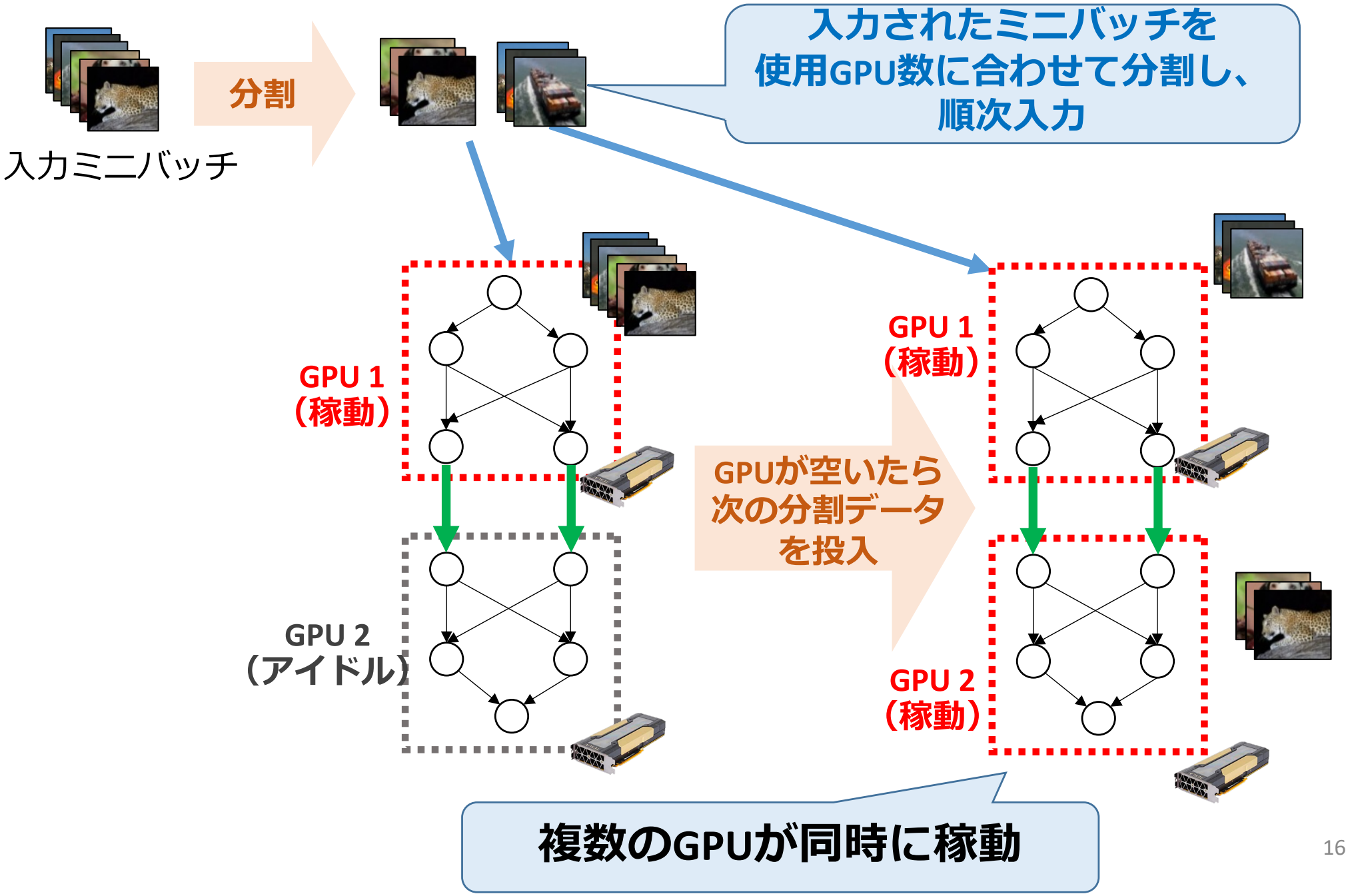
複数のGPU/サーバで並列計算

ネットワーク分割の課題

- ネットワークの断片に依存関係があると、順番に計算することになり、**同時に一つのGPUしか稼動しない**
  → GPUの利用率が下がる



パイプライン並列

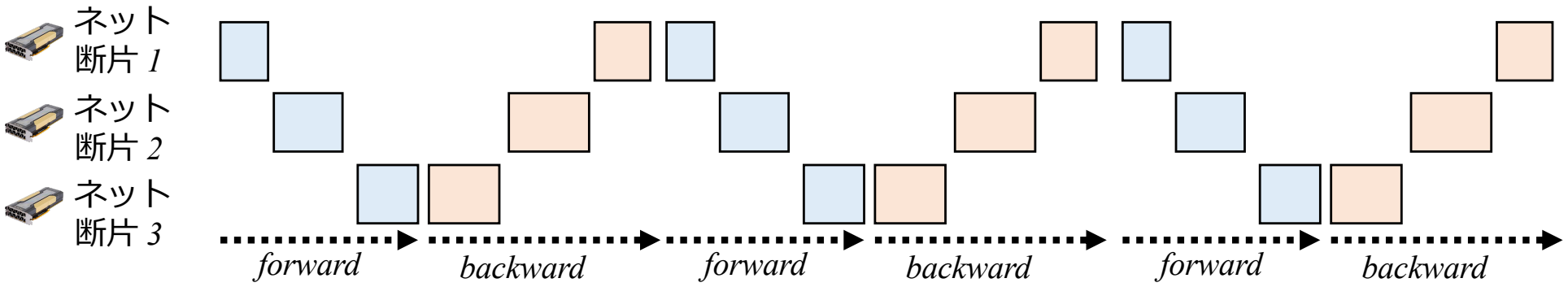

入力ミニバッチ
分割
入力されたミニバッチを
使用GPU数に合わせて分割し、
順次入力
GPU 1
（稼動）
GPU 2
（アイドル）
GPUが空いたら
次の分割データ
を投入
GPU 1
（稼動）
GPU 2
（稼動）
複数のGPUが同時に稼動

パイプライン並列

学習のサイクルにおけるGPU利用のイメージ

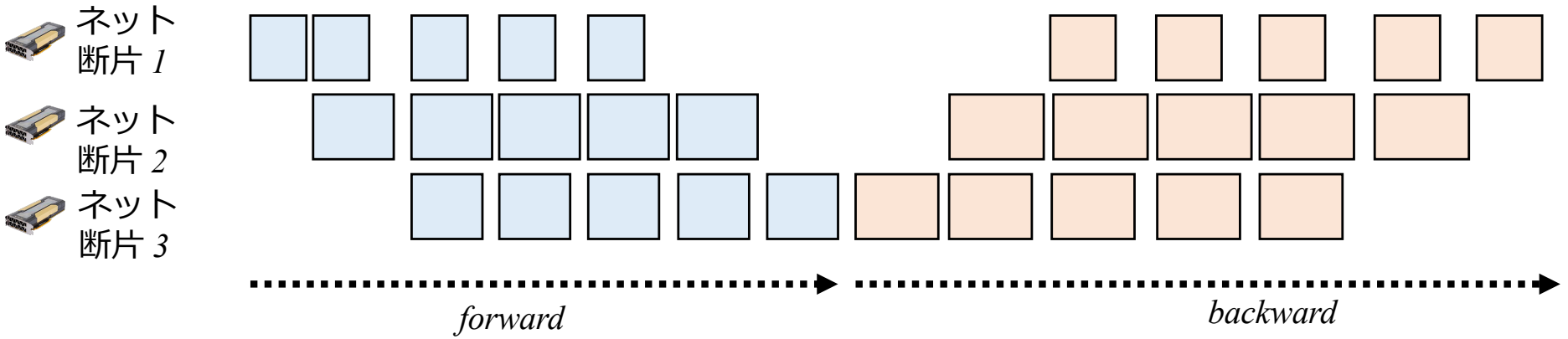
パイプライン並列なし

GPU利用率が低い



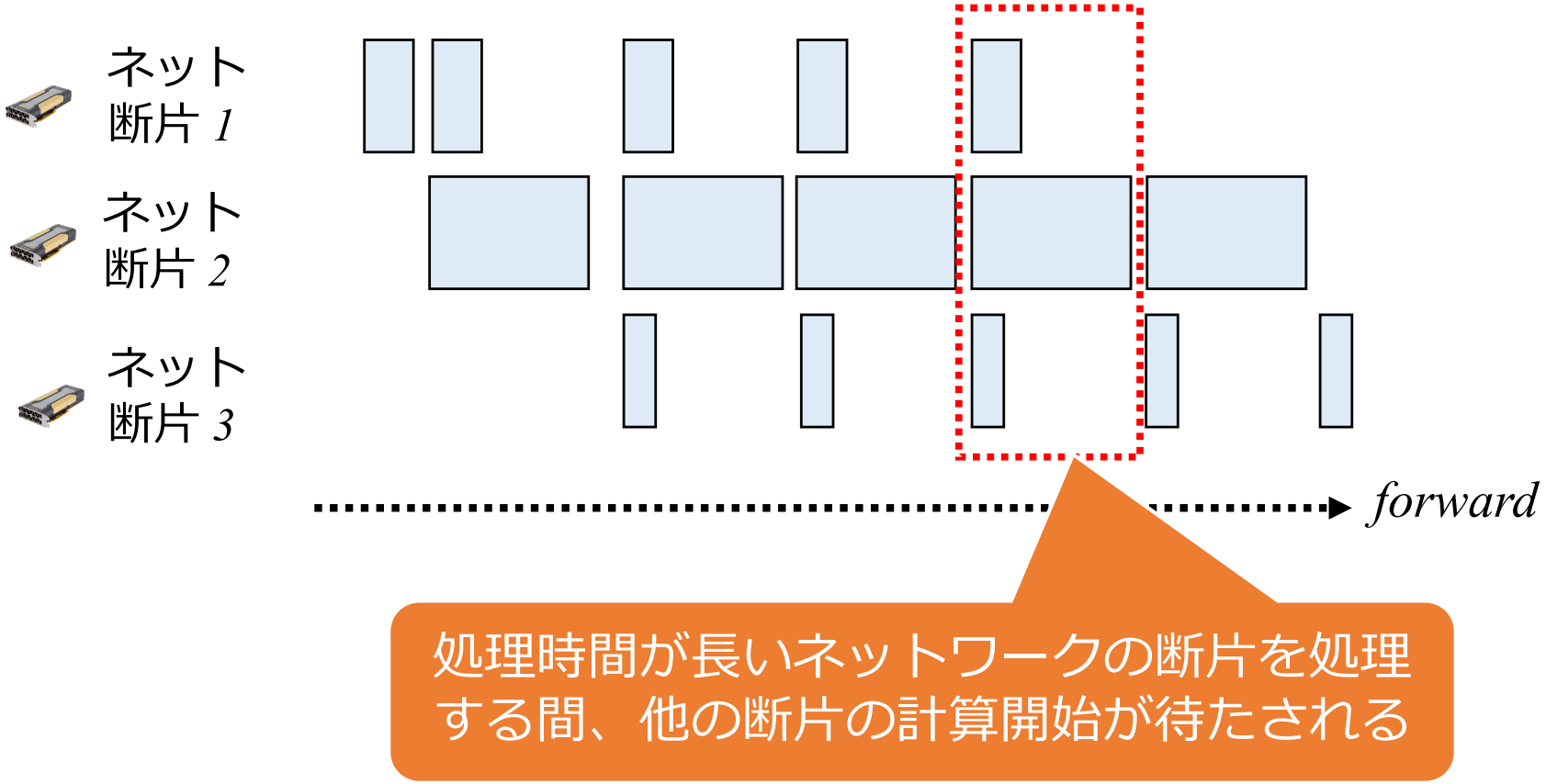
パイプライン並列あり

GPU利用率が改善



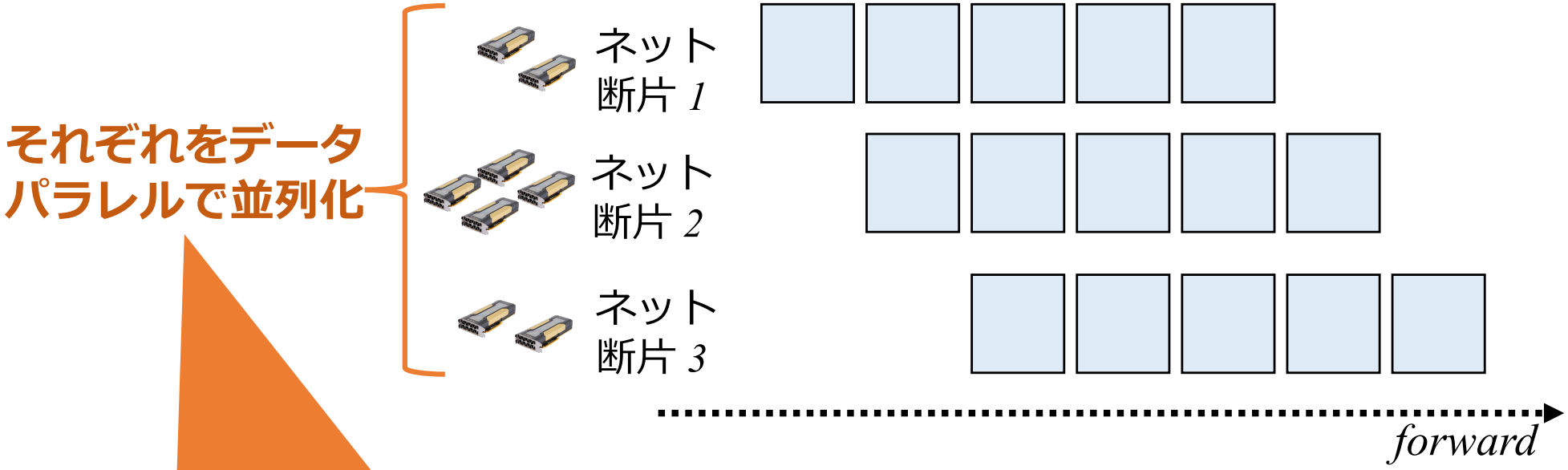
パイプライン並列

- もっとも計算時間が長いニューラルネットワークの断片が、全体のボトルネックになる
  → **各断片の処理時間が均等なとき、全体として最適な学習速度**



パイプライン並列

- 分割箇所と共に、ネットワークの各断片でのデータパラレルの並列度を調整
- 実際にネットワークの断片を計算して、処理時間と必要メモリをプロファイリング



ネットワーク断片ごとに、異なるデータパラレルの並列度を設定可能

探索空間の削減

- ネットワーク分割とデータパラレル並列度設定の探索空間は膨大
  - 単純な探索アルゴリズムでは、探索空間が大きすぎて、大きなネットワークを扱えない
  - BERT（隠れ層1024, 系列長512, GPU32枚）の場合、24時間の探索で、最大72レイヤまでしか実行可能なネットワーク分割とデータパラレル並列度を発見できず

- 以下の工夫を導入
  - 別のGPUに配置されると通信オーバヘッドが特に大きい演算を結合して、ネットワークの粗粒度な断片を作る
  - 粗粒度な断片の組み合わせと、データパラレル並列度を、動的計画法で効率的に探索

→ **256レイヤ以上のBERTでも探索可能に**（評価実験での上限、それ以上でもおそらく動作）

RaNNCの適用

様々な特徴のネットワークについて、既存のネットワーク定義をそのまま利用可能

動作を確認したネットワークの例

ネットワーク	大規模化
	言語処理
BERT	**約1000億パラメータ**まで動作確認
T5	約30億パラメータまで動作確認
	画像処理
ResNet	約37億パラメータまで動作確認
3D Unet	正常にネットワークを分割することを確認（バッチサイズ1でもGPUメモリに収まらないpatchサイズ(*160, 288, 288*)に設定して動作）

試していないが、さらなる大規模化も可能と思われる

パラメータが多いモデルだけでなく、入力データ・中間データが大きいネットワークにも有効

データ駆動知能システム研究センター（DIRECT）では、50億パラメータの大規模BERTを350GBの日本語テキストで学習

今後の課題

1. Tensor partitioningの導入

- Megatron-LM, Mesh-TensorFlowなどの既存フレームワークは、テンソルを特定の次元で分割し、並列計算（Transformer系モデルで有効）
- 演算を単位としてニューラルネットワークを複数の断片に分割する方式と、ハイブリッド化の予定

2. GPU稼働率の向上

- パイプライン並列では、forward/backwardそれぞれの開始時・終了時にアイドル時間が発生
- 計算と通信の精密なスケジューリングにより、アイドル時間の減少を目指す

3. より多様なネットワークへ適用

- ALBERT, Electra, StructBERT等、 BERTの派生ネットワークに適用の予定
- これらのネットワークも大規模化により性能向上することを確認していく

References

[Oh 2019] J-H. Oh, et al. "Open Domain Why-Question Answering with Adversarial Learning to Encode Answer Texts", In *the 57th Annual Meeting of the Association for Computational Linguistics (ACL 2019)*, pp.4227-4237, 2019

[Shoeybi 2019] M. Shoeybi et al., "Megatron-LM: Training multi-billion param- eter language models using model parallelism," arXiv preprint, arXiv:1909.08053, 2019.

[Shazeer 2018] N. Shazeer et al., "Mesh-TensorFlow: deep learning for supercomput- ers," in *Advances in Neural Information Processing Systems 31 (NIPS 2018)*, 2018, pp. 10 435–10 444.

[Huang 2018] Y.Huang et al.,"GPipe: Efficient training of giant neural networks using pipeline parallelism," arXiv preprint, arXiv:1811.06965, 2018.

[Kim 2020] C. Kim et al., "torchgpipe: On-the-fly pipeline parallelism for training giant models," arXiv preprint, arXiv:2004.09910, 2020.

[Narayanan 2020] D. Narayanan et al., "Memory-Efficient Pipeline-Parallel DNN Training," arXiv preprint, arXiv:2006.09503, 2020.