NVIDIA DEEP LEARNING INSTITUTE

第93回お試しアカウント付き並列プログラミング講習会
「REEDBUSH スパコンを用いたGPUディープラーニング入門」

ハンズオン#2: マルチGPUによる高速化体験

山崎和博
NVIDIA, ディープラーニング ソリューションアーキテクト

AGENDA

ハンズオン#2のテーマ: 分散学習

タスク#1: マルチGPUでの学習

タスク#2: マルチノードでの学習

ハンズオン#2のゴール

複数のノードを利用できるようになる

- Reedbushの複数ノードを利用してジョブを流す方法を把握する
- マルチGPU/マルチノードでの学習方法を理解する

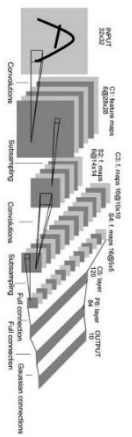
ハンズオン#2のテーマ: 分散学習

分散学習

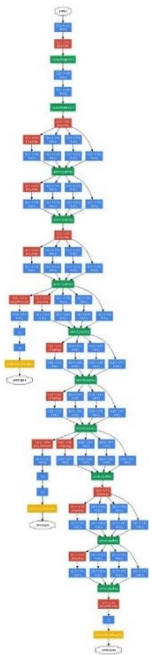
モデル/データの大規模化

モデルの複雑化

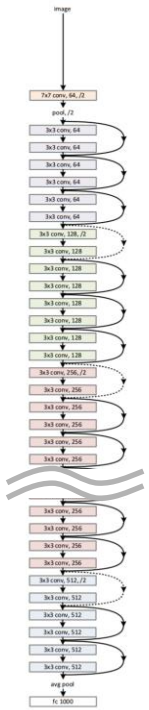
LeNet
5layers



GoogLeNet
24layers



ResNet
152layers



?

データの拡大

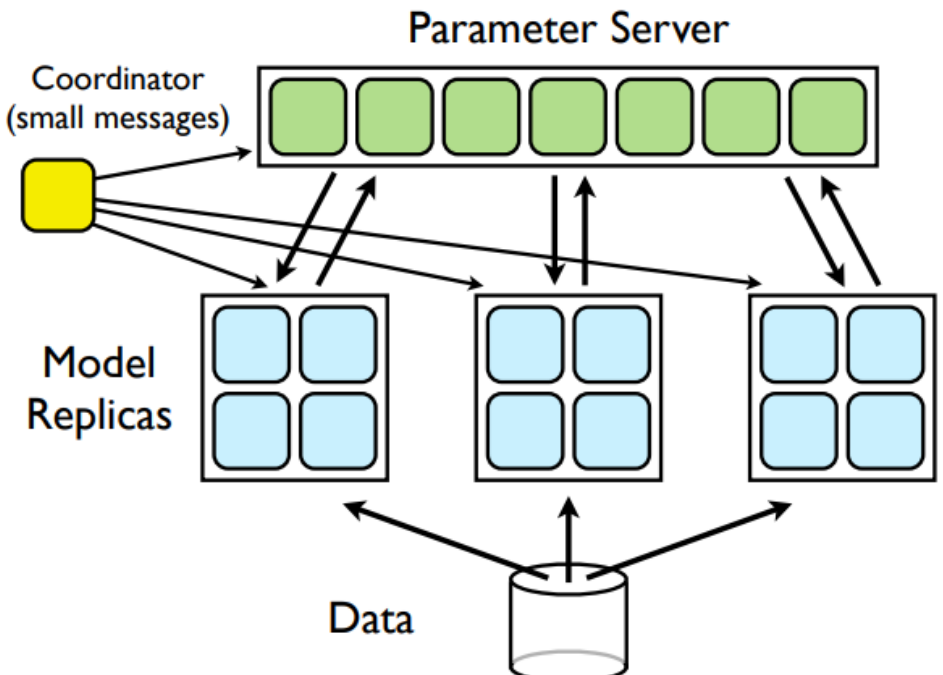
?

ImageNet
256x256, 1M imgs

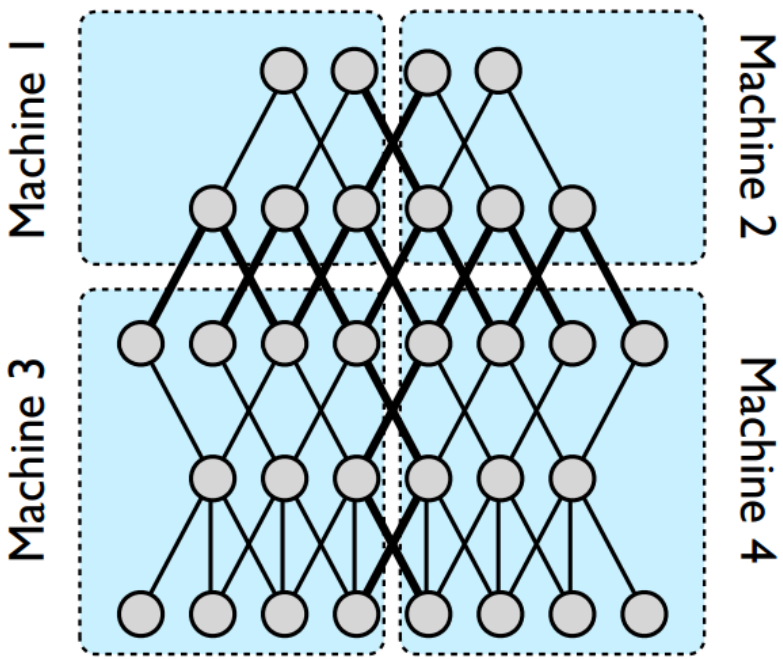
MNIST
28x28,60K imgs

分散学習

マルチGPU/マルチノードで分散して学習



データ並列（data parallelism）

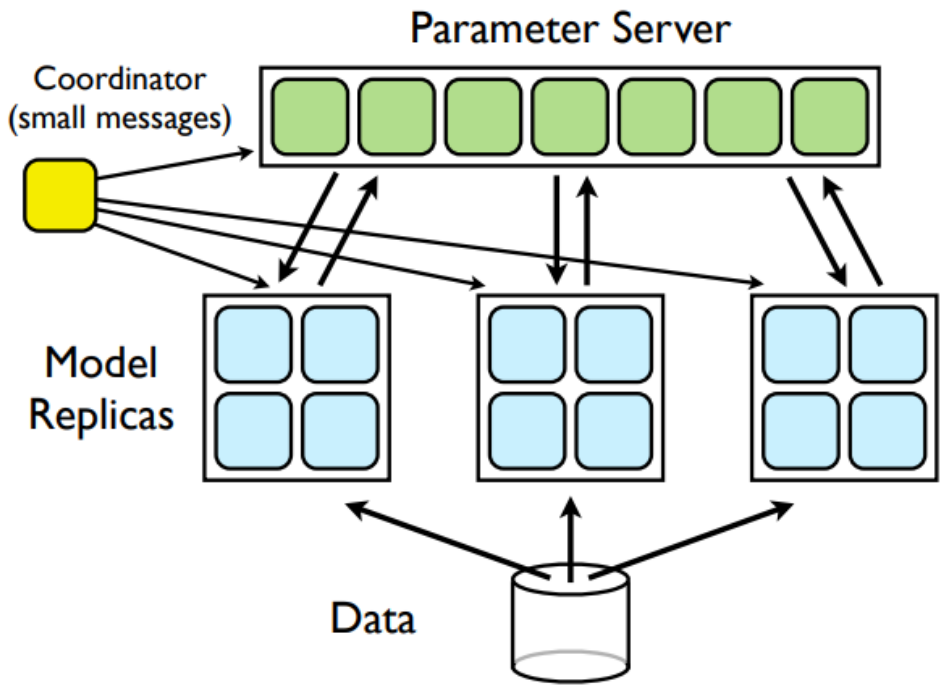


モデル並列（model parallelism）

Large Scale Distributed Deep Networks より

分散学習

マルチGPU/マルチノードで分散して学習



データ並列（data parallelism）

- 複数GPUで同じネットワーク（パラメータ）を共有
- データを分割してそれぞれ勾配計算し、パラメータを更新

→ **学習時間の短縮が期待できる**

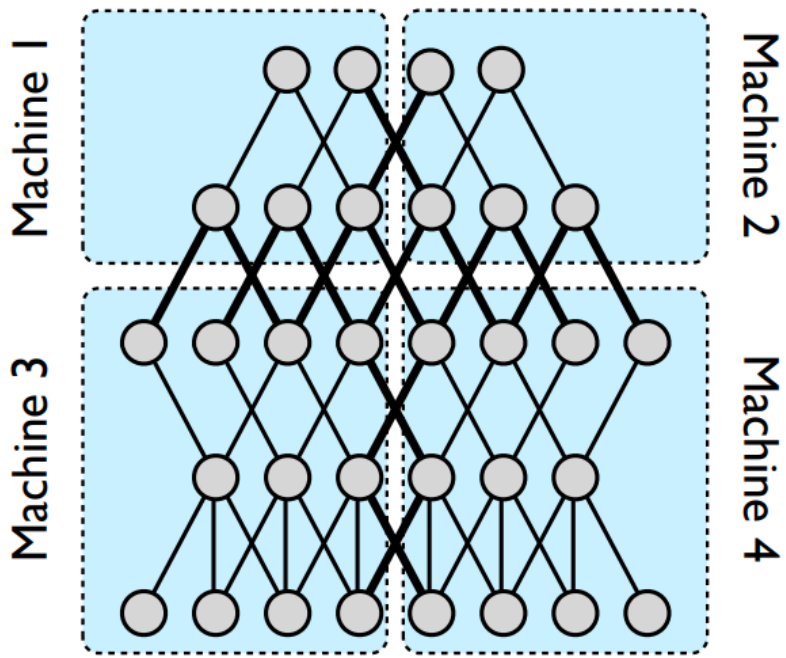
Large Scale Distributed Deep Networks より

分散学習

マルチGPU/マルチノードで分散して学習

- ネットワーク（パラメータ）を複数GPUで分散して保持
- 同じデータに対し、各GPUが協調してパラメータを更新

→ **1GPUに乗り切らない大規模なネットワークを処理できる可能性**

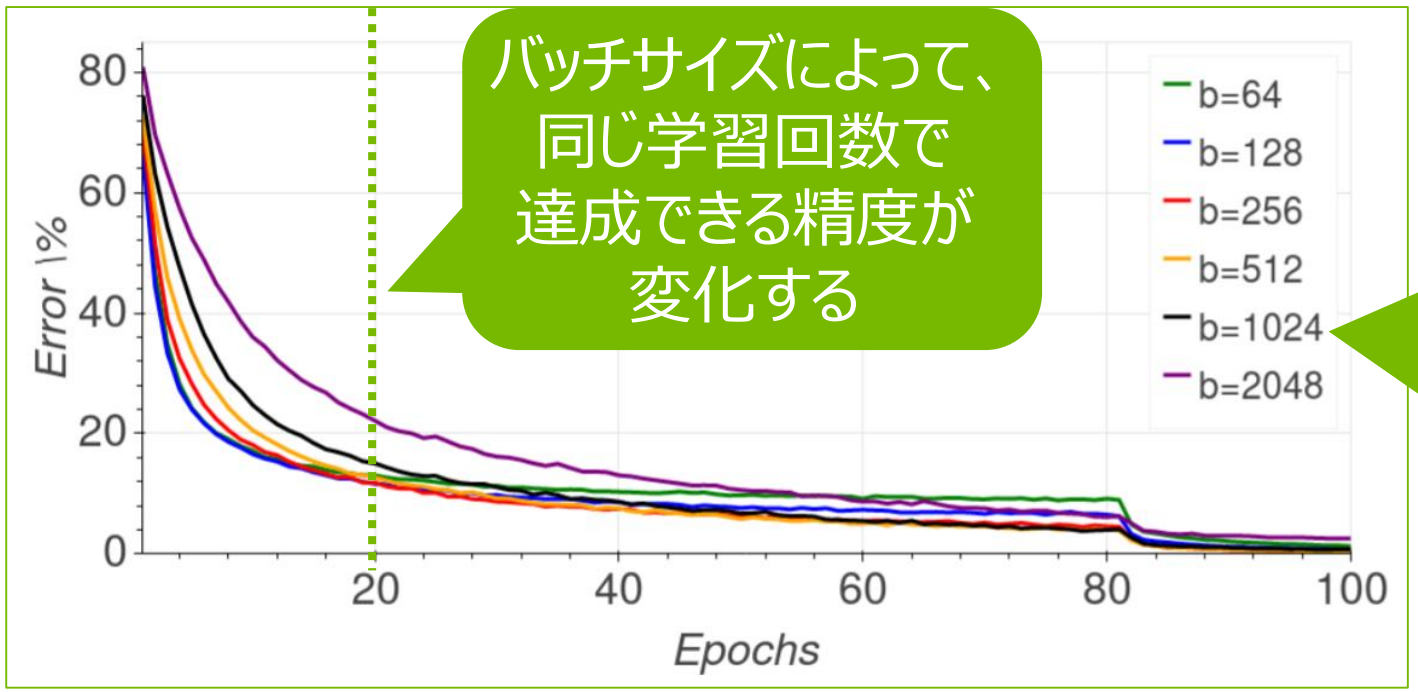


モデル並列（model parallelism）

Large Scale Distributed Deep Networks より

分散学習における注意点

バッチサイズと精度の関係



バッチサイズによって、
同じ学習回数で
達成できる精度が
変化する

分散学習すると、
トータルのバッチサイズ
は大きくなりがち
→ 問題になる
可能性あり

https://arxiv.org/abs/1705.08741 より

分散学習における注意点

バッチサイズと精度の関係

学習率やバッチサイズを調整することで、精度低下を回避する研究が多数

- Train longer, generalize better: closing the generalization gap in large batch training of neural networks
- Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour
- LARGE BATCH TRAINING OF CONVOLUTIONAL NETWORKS
- Don't Decay the Learning Rate, Increase the Batch Size

……が、今日は細かいところに踏み込みません

https://arxiv.org/abs/1705.08741 より

タスク#1: マルチGPUでの学習

マルチGPUで学習ジョブを流す

簡単な例で、ジョブの流し方を把握する

以下の作業を完了させる。

1. run_multi_gpu_training.shの実行環境指定を完成させる
2. train_cifar_multi_gpu.pyに「PUT YOUR CODE」という箇所があるので、書き換えてプログラムを完成させる
3. スクリプトの修正完了したら実行
    1. qsub -j oe run_multi_gpu_training.sh
4. 結果を確認

CHAINERのAPI

学習の実行

- ネットワークの更新ロジックを指定
  - chainer.training.updaters.MultiprocessParallelUpdater(iterators, optimizer, devices)
  - 基本はシングルGPUのクラスと同じ使い方
    - GPUへのモデル転送は明示的に実施する必要なし
  - 学習データのイテレータをGPU数と同じ数準備する
    - デバイス名をキーとするdictを作成
    - メインのGPUのみデバイス名を「main」にする必要あり

CHAINERのAPI

データの分割

- chainer.datasets.split_dataset_n_random(dataset, n)
  - 学習データのイテレータ作成時に使用
  - データをランダムにGPU数分割

マルチGPUでの学習実行結果

```
Start a training script using single GPU.
# Minibatch-size: 32
# epoch: 4

Using CIFAR10 dataset: /lustre/gt00/share/lecture/20180122_dl_intro/dataset//cifar10_256px_5p.pickle.gz
epoch  main/loss   validation/main/loss  main/accuracy  validation/ma
1      4.03905     2.57575               0.113924       0.104687
(中略)
4      2.13622     2.10527               0.210337       0.219531                  195.589
Throughput: 51.12377404072687 [images/sec.] (10000 / 195.6037125121802)
Start a training script using multiple GPUs.
# GPUs: 2
# Minibatch-size: 32
# epoch: 4

Using CIFAR10 dataset: /lustre/gt00/share/lecture/20180122_dl_intro/dataset//cifar10_256px_5p.pickle.gz
{'main': 0, 'gpu1': 1}
/lustre/app/acc/cuda9/chainermn/1.0.0/lib/python3.6/site-
packages/chainer/training/updaters/multiprocess_parallel_updater.py:141: UserWarning: optimizer.lr is changed to 0.025 by
MultiprocessParallelUpdater for new batch size.
  format(optimizer.lr))
epoch  main/loss   validation/main/loss  main/accuracy  validation/ma
1      3.03975     2.40538               0.114844       0.10625
(中略)
4      2.8941      2.50813               0.126603       0.0980469                 114.111
Throughput: 87.6300597715253 [images/sec.] (10000 / 114.11609242390841)
```

シングルGPUだと
51.1image/sec.

同条件で2GPU使うと
87.6image/sec.と高速に

タスク#2: マルチノードでの学習

マルチノードで学習ジョブを流す

簡単な例で、ジョブの流し方を把握する

以下の作業を完了させる。

1. run_multi_node_training.shの実行環境指定を完成させる
2. train_cifar_multi_node.pyに「PUT YOUR CODE」という箇所があるので、書き換えてプログラムを完成させる
3. スクリプトの修正完了したら実行
    1. qsub -j oe run_multi_node_training.sh
4. 結果を確認

ジョブスクリプト

マルチノード利用の条件指定

```
#!/bin/bash
#PBS -q (対象ジョブキュー名)
#PBS -l (利用ノード数など)
#PBS -W (利用グループ名)
#PBS -l (実行時間制限)
#PBS -N (ジョブ名)

...
```

利用ノードなどのパラメータは、以下をコロン「:」区切りで指定

- `select` ノード数
- `mpiprocs` mpiプロセス数/ノード
- `ompthreads` スレッド数/プロセス

ノードごとに2GPUで2ノードなので……

CHAINERMNのAPI

環境情報の取得

- `communicator.rank`
  - ワーカーID(プロセスID)を取得
- `communicator.intra_rank`
  - ノード内のプロセスIDを取得
- ノード内2プロセスで、2ノード使用の場合の例
  - ノード1: rank=0&intra_rank=0, rank=1&intra_rank=1
  - ノード2: rank=2&intra_rank=0, rank=3&intra_rank=1

CHAINERMNのAPI

分散学習用のAPI

- chainermn.scatter_dataset(dataset, comm, shuffle)
  - データセットを分割して、各プロセスに配布
- chainermn.create_multi_node_optimizer(actual_optimizer, communicator)
  - マルチノード環境で動作するよう、optimizerをラップ
- chainermn.create_multi_node_evaluator(actual_evaluator, communicator)
  - マルチノード環境で動作するよう、evaluatorをラップ

マルチノードでの学習実行結果

```
[1516085554.200252] [a087:13926:0]         sys.c:744  MXM  WARN  Conflicting CPU frequencies detected,
using: 2600.14
(中略)
Start a training script using multiple nodes.
==========================================
Num process (COMM_WORLD): 4
Using GPUs
Using hierarchical communicator
Num Minibatch-size: 32
Num epoch: 4
==========================================
Using CIFAR10 dataset:
/lustre/gt00/share/lecture/20180122_dl_intro/dataset//cifar10_256px_5p.pickle.gz
epoch  main/loss   validation/main/loss  main/accuracy  validation/main/accuracy  elapsed_time
1      2.61112     36.5229               0.110937       0.140356
(中略)
4      2.73942     2.32497               0.129688       0.103987                  58.078
Throughput: 172.12098418175134 [images/sec.] (10000 / 58.09866848913953)
```

2ノードx2GPUでは
172.1image/sec.

(OPTION) 追加で試す

早く終わった方のために

- プロセス間通信のアルゴリズムを変更するとどうなるか比較してみる
  - デフォルトはhierarchicalだが、ほかにも存在
  - 詳細は http://chainermn.readthedocs.io/en/latest/reference/index.html#chainermn.create_communicator を参照
- Chainer単体で2GPUを使った結果と、ChainerMNで1ノード2GPUを使った結果とを比較してみる
  - 違いがあるのかどうか検証

(FYI) 性能が思うように出ないときに

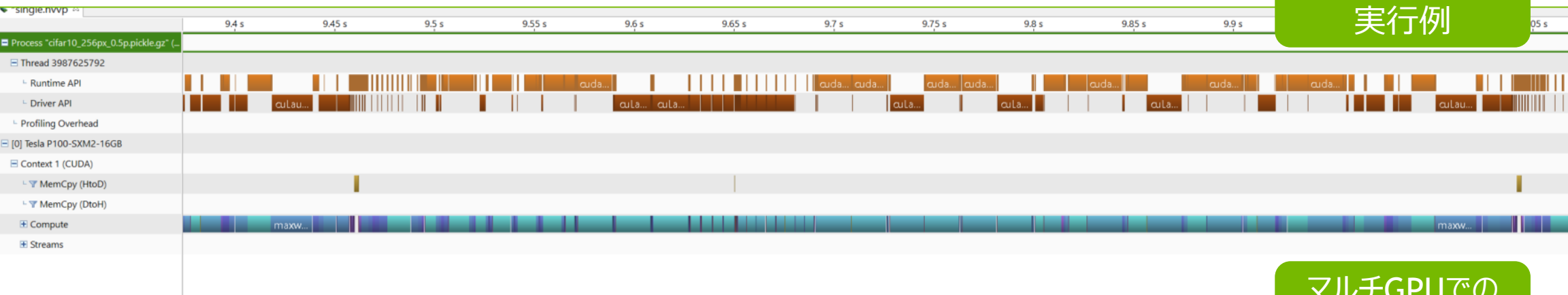
プロファイラによる解析

- 十分にGPUを使い切れているかどうかを確認する際、プロファイラが有効なことがある
  - コマンドはnvprof
  - 結果の可視化にはNVIDIA Visual Profilerを使う
- 実行するには対象コマンドcmdに対し、`nvprof -o ${resultfile} cmd`とすればOK
  - オプション`--profile-child-processes`を追加すると子プロセスも追跡できる
  - 子プロセスを追うときは、出力ファイル名に%pを含める（ファイル名にプロセスIDを含めるため）

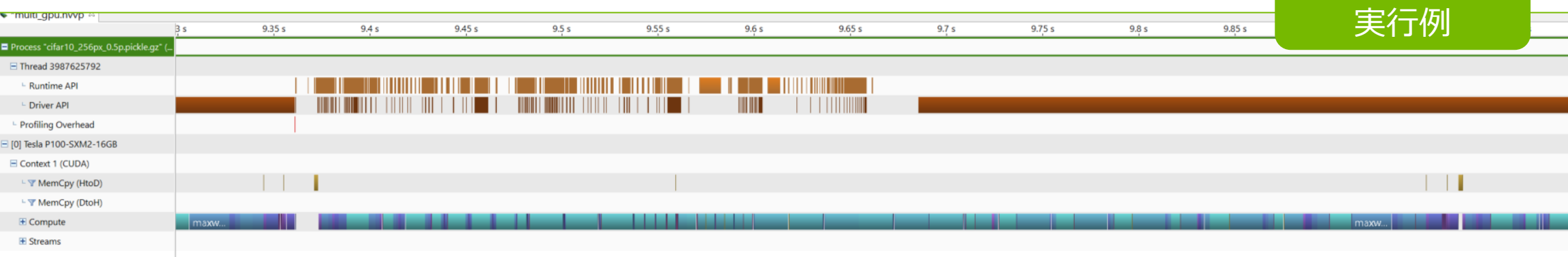
(FYI) 性能が思うように出ないときに

プロファイラによる解析

シングルGPUでの実行例



マルチGPUでの実行例



NVIDIA | DEEP LEARNING INSTITUTE

www.nvidia.com/dli