1J-03

# スーパーコンピュータを用いた分散深層学習の性能評価

羽生 達郎 †
名古屋大学 情報学部コンピュータ科学科 †
大島 聡史 ‡
名古屋大学 情報基盤センター ‡
片桐 孝洋 §
名古屋大学 情報基盤センター §
永井 亨 ¶
名古屋大学 情報基盤センター ¶

## 1 はじめに

分散学習にはデータ並列とモデル並列の 2 通りがある．それぞれに異なる長所があり，長所は，データ並列は学習の高速化，モデル並列は大規模モデルの学習が挙げられる．しかし現状では，大規模な分散機械学習の適用が不十分である．加えて，データ並列に比べ，モデル並列を扱うライブラリは少ない．

例えば，Horovod[1] は，TensorFlow，Keras, PyToarch などに対応した分散ディープラーニングフレームワークであり，容易に利用できる分散学習環境を提供している。

一方，RaNNC (Rapid Neural Network Connector)[2] は，2021 年に提案された大規模なニューラルネットワークを学習させるために使用される自動並列化ミドルウェアである．近年のディープラーニング用のネットワークは多くの学習パラメタを持つことが多いため，GPU のメモリに収まりきらない．RaNNC はこのような巨大なネットワークをモデル並列で自動的に分割し，複数の GPU を使って計算する．

そこで本研究では，大規模 GPU 環境での分散機械学習フレームワークの性能評価を目標に，GPU スーパーコンピュータを利用して性能評価を行う．

Performance Evaluation of Distributed Deep Learning using Supercomputers
† Tatsuro Hanyu, Department of Computer science, School of Informatics, Nagoya University
‡ Satoshi Ohoshima, Information Technology Center, Nagoya University
§ Takahiro Katagiri, Information Technology Center, Nagoya University
¶ Toru Nagai, Information Technology Center, Nagoya University

## 2 RaNNC

ここでは，深層学習のフレームワークの１つである RaNNC について説明する．

Megatron-LM や Mesh-TensorFlow など，ユーザーが特定のネットワークのパーティショニングを実装する必要がある既存のフレームワークと比較して，RaNNC は記述を変更することなく，PyTorch のネットワークを自動的にパーティショニングすることができる．さらに RaNNC は，基本的にネットワークアーキテクチャに制限がない．既存のフレームワークでは，トランスベースのネットワークにしか適用できないという問題がある．

## 3 予備評価

### 3.1 実験環境

本予備評価では，名古屋大学情報基盤センターに設置されたスーパーコンピュータ「不老」Type II サブシステムを使用した．

ハードウェア構成，ソフトウェアのバージョンを表 1，表 2 に示す．

### 3.2 ベンチマークコード

本予備評価では，RaNNC のチュートリアル [2] を用いた．本チュートリアルは，データパラレルの学習コード例となっており，PyTorch の確率的勾配降下法による関数 optim.SGD を，学習率 lr=0.01 で呼びだすプログラムとなっている．概略は以下である．

```
batch_size = int(sys.argv[1])
```

表 1　「不老」Type II サブシステムのハードウェア構成

| | |
|---|---|
| CPU | Intel Xeon Gold 6230 × 2 |
| GPU | NVIDIA Tesla V100 × 4 |
| メモリ | メインメモリ 384GiB<br>デバイスメモリ 32GiB × 4 |
| ノード数 | 221 ノード |
| 総演算性能 | 7.489PFLOPS |

表 2　ソフトウェアのバージョン

| | |
|---|---|
| Python | Ver. 3.7.7 |
| RaNNC | Ver. 0.7.5 |
| PyTorch | Ver. 1.11.0 |
| CUDA | Ver. 11.3 |

```
hidden = int(sys.argv[2])
layers = int(sys.argv[3])
model = Net(hidden, layers)
if pyrannc.get_rank() == 0:
    print("#Parameters={}".format
    (sum(p.numel() for p
    in model.parameters())))
opt = optim.SGD(model.parameters(), lr=0.01)
model = pyrannc.RaNNCModule(model, opt)
x = torch.randn(batch_size, hidden,
    requires_grad=True).to
    (torch.device("cuda"))
out = model(x)
target = torch.randn_like(out)
out.backward(target)
opt.step()
```

なおここでは，batch_size=64，hidden=512，layers=10 を設定して実行した．

### 3.3　実行結果とまとめ

図 1〜図 2 に，実行結果を示す．

図 1 より，1 ノード 4GPU までは，3GPU 利用時に性能劣化が観測された．また，4GPU 利用時が最も高速であった．

図 2 より，1 ノード当たり 4GPU 利用する前提において，2 ノード〜4 ノードまで変化させて実行させた．ノード並列により高速化されることは確認できるものの，4 ノード 16GPU 実行時の実行時間 15.46 秒に対して，図 1 の 1 ノード 4GPU 実行時は 13.59 秒であり，1 ノード実行の方が高速である．この理由は，マルチノード実行による MPI の通信時間の増加が理由と考えられるが，詳細な解析が必要である．

当日は，さらに詳細な性能解析結果を示す予定である．

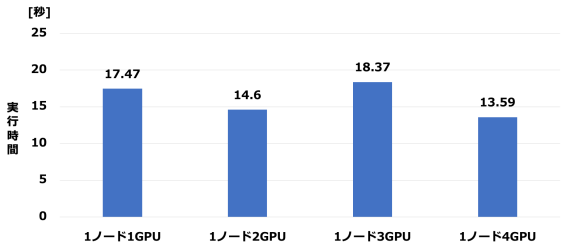


図 1　1 ノード，複数 GPU での実行

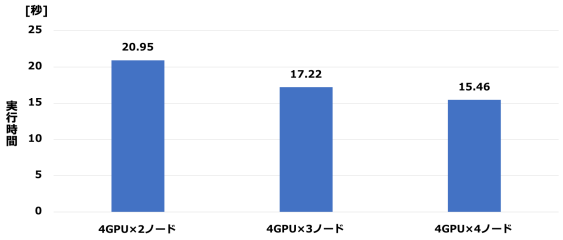


図 2　複数ノード，複数 GPU での実行

## 謝辞

本研究は JSPS 科研費 JP19H05662 の助成を受けたものです。また、研究内容についてコメントを頂いた星野哲也准教授に感謝の意を表します。

## 参考文献

[1] Horovod　https://github.com/horovod/horovod (最終閲覧日 2023/01/10)

[2] RaNNC (Rapid Neural Network Connector), https://github.com/nict-wisdom/rannc (最終閲覧日 2023/01/06)