5J-01

畳み込みニューラルネットワークにおける分割モデルのGPUへの割り当て

Assignment of Decomposed Models to GPUs for Concvolutional Neural Network

綿貫 幸† 吉田 明正†
Yuki Watanuki Akimasa Yoshida

1 はじめに

深層学習は，画像認識をはじめとして広い分野に渡り活用されているが，精度向上のためには大規模なモデルや大量のデータによる学習が必要であり，学習の長時間化が課題となる．高い並列処理性能を持つGPUは深層学習の高速化に活用されており，マルチGPUを用いた並列処理により効率的な学習高速化を実現するアプローチとして，データ並列とモデル並列がある．データ並列は容易に実装することが可能であるが，学習パラメータを更新毎に複製するため，大規模なモデルでは通信時間が増大してしまう．これに対し，モデルをステージに分割して各GPUに割り当てるモデル並列は，ステージ間でのデータの受け渡し時にのみGPU間の通信が発生するため，通信時間を抑えることができる．本稿では，代表的な深層学習モデルであるCNNに対して，各GPUに複数ステージを割り当てたモデル並列を適用し，マルチGPU環境での高速化を図る．画像分類CNNのマルチGPU向け並列プログラムをCUDAとOpenMPを用いて実装し，NVIDIA Tesla K80搭載サーバ上で性能評価を行い，提案手法の有効性を確認した．

2 深層学習の並列処理

本章では，深層学習の並列処理手法であるモデル並列について述べる．

2.1 モデル並列

モデル並列とは，1つの学習モデルを複数のステージに分割し，それぞれをデバイスに割り当てる手法である．モデル並列の適用により，単一デバイスのメモリ制限を越える大規模なモデルの実装が可能となる．最も単純なモデル並列の実装例を図1に示す．図1では，モデルを0，1，2，3の4ステージに分割し，GPU0，1，2，3にそれぞれ割り当てている．図中のa，b，c，…はミニバッチを表し，0aはステージ0の処理をミニバッチaで実行していることを意味する．ステージ内でのミニバッチの処理が完了すると，出力されたデータを次のステージの入力として転送するため，GPU間の通信が発生する．この方法では，1つのミニバッチに対して各ステージを担当するGPUが順番に稼働するため，アイドル状態が多くなってしまう．

2.2 パイプライン型モデル並列

パイプライン型モデル並列は，パイプライン処理によって複数のミニバッチの学習を同時に進行させ，学習の高速化を達成する[1][2]．図2は，図1同様にモデルを4分割した上で，4つのミニバッチを同時に処理するパイプライン型モデル並列の実装例である．図1と比較して GPU の稼働率を大きく改善することができ，このときの速度向上率は，最も負荷の重いステージの処理時間に依存する．この実装では，ステージ毎に各ミニバッチで計算された勾配を足し合わせていき，最後尾のミニバッチの逆伝播完了時（図中の3d，2d，1d，0d）に勾配を平均化し，パラメータを更新する．そのため，バッチサイズを元の1/4に細分化して実装する．

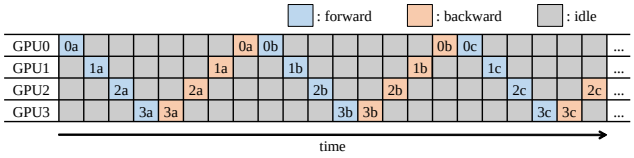

図1 単純なモデル並列.

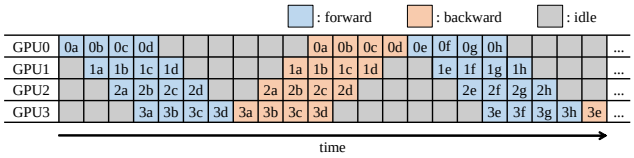

図2 4分割パイプライン型モデル並列.

3 マルチGPU環境でのCNNの並列処理

本稿では，画像分類CNNに対し，マルチGPU環境におけるパイプライン型モデル並列の適用による高速化手法を提案する．本稿で扱うCNNのプログラムはC++により実装されており[3]，ネットワークは7つの畳み込み層とバッチ正規化層，3つのプーリング層，4つのドロップアウト層を持つ．

3.1 CNNのモデル分割とGPUへの割り当て

2章で説明したモデル並列の従来法では，各GPUに1つのステージを割り当てている．本稿では，モデルをさらに細分化し，各GPUに2つのステージを割り当てることで，より高速なパイプライン型モデル並列の実現を図る（図3）．パイプライン処理を適用した実装は図4のようになる．このとき理論上では，各ステージの負荷が完全に均等であると仮定した場合，図2の実装と比較して処理時間を $((22/2)/14) \times 100 \approx 78.6\%$ に短縮することができる．

3.2 OpenMP/CUDAによる並列処理の実装

本稿で扱うCNNの各層の処理はCUDAカーネルにより実装され，順伝播及び逆伝播の処理はGPU上で実行される．また，行列演算にはNVIDIAが提供するcuBLASライブラリ関数を使用している．マルチGPU上での並列処理の実装には，OpenMPを使用する（図5）．`num_threads()`を伴うparallel指示節により4スレッドの並列領域を作成し，スレッド番号とGPUのIDを対応

†明治大学大学院先端数理科学研究科ネットワークデザイン専攻
Department of Network Design, Graduate School of Advanced Mathematical Sciences, Meiji University

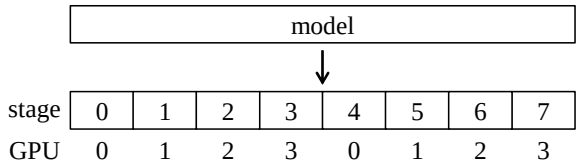

図 3　提案手法によるモデルの分割と割り当て.

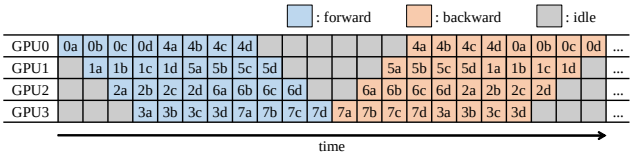

図 4　8 分割パイプライン型モデル並列（提案）.

させることで並列処理を行い，`#pragma omp barrier` でステージが進む毎に同期を行っている．図 5 の 16 行以降の逆伝播を行う関数 `backward()` では，第 3 引数で最後尾バッチを判定するフラグを渡し，最後尾バッチで学習パラメータの勾配の平均化を行っている．

4　NVIDIA Tesla K80 搭載サーバ上での性能評価

本章では，性能評価について述べる．

4.1　性能評価環境

性能評価に用いるマルチ GPU サーバの構成を，表 1 に示す．本性能評価では，CIFAR-10 データセット [4] の訓練用画像 5 万枚を用いた 10 クラス画像分類 CNN プログラムに提案手法を適用し，学習時間を測定する．

表 1　性能評価に用いるマルチ GPU サーバの構成.

マシン	NVIDIA Tesla K80 搭載サーバ
プロセッサ	Intel Xeon E5-2680 v3 (2.5GHz, 12Core×2)
メモリ	64GB
GPU	NVIDIA Tesla K80×2 (GK210×4)
OS	CentOS 6.9
CUDA Toolkit	9.1
g++	5.3.1

4.2　マルチ GPU 上でのモデル並列 CNN の性能評価

図 6 は，バッチサイズを 400 として学習した CNN の訓練データの正解率を示す．ここで，1 分割 1GPU はモデル並列を適用しない実装，4 分割 4GPU は図 2 の実装，8 分割 4GPU は提案手法である図 4 の実装である．図 6 の結果から，ともに 17 エポック終了時点で正解率 80%を越えており，提案手法の適用後も同等の分類精度を維持していることが確認できた．

また，表 2 より，提案手法でのエポック平均学習時間は 2736.8[s] となり，1 分割 1GPU 実行比で 2.076 倍の速度向上が得られた．4 分割 4GPU 実行と比較すると，実行時間を約 89%に短縮することができた．以上の結果から，マルチ GPU 上で各 GPU に複数のステージを割り当てるパイプライン型モデル並列について，有効性が確認された．

5　おわりに

本稿では，マルチ GPU 上での CNN の学習高速化のために，モデルの分割数を増やし，各 GPU に複数ステージを割り当てるパイプライン型モデル並列を提案した．

```
01: #pragma omp parallel private(p, np, pp) num_threads(4)
02: {
03:     p  = omp_get_thread_num(); // スレッド番号をpに取得
04:     cudaSetDevice(p);     // p番目のGPUにセット
05:     if (p<3) np = p + 1; // next device id
06:     if (p>0) pp = p - 1; // prev device id
        ...
07:     #pragma omp barrier  // forward
08:     if      (p==0) { forward0(model0, batch3);
09:                      cudaSetDevice(np); cudaMemcpy(...); } // GPU0
10:     else if (p==1) { forward1(model1, batch2);
11:                      cudaSetDevice(np); cudaMemcpy(...); } // GPU1
12:     else if (p==2) { forward2(model2, batch1);
13:                      cudaSetDevice(np); cudaMemcpy(...); } // GPU2
14:     else if (p==3) { forward3(model3, batch0);
15:                      cudaSetDevice(np); cudaMemcpy(...); } // GPU3
        ...
16:     #pragma omp barrier // backward
17:     if      (p==0) { backward4(model4, batch0, false);
18:                      cudaSetDevice(pp); cudaMemcpy(...); } // GPU0
19:     else if (p==1) { backward5(model5, batch1, false);
20:                      cudaSetDevice(pp); cudaMemcpy(...); } // GPU1
21:     else if (p==2) { backward6(model6, batch2, false);
22:                      cudaSetDevice(pp); cudaMemcpy(...); } // GPU2
23:     else if (p==3) { backward7(model7, batch3, true);
24:                      cudaSetDevice(pp); cudaMemcpy(...);   // GPU3
25:                      cudaSetDevice(p);  optimizer7.update(); }
        ...
26: }
```

図 5　マルチ GPU 並列プログラム（8 分割 4GPU）.

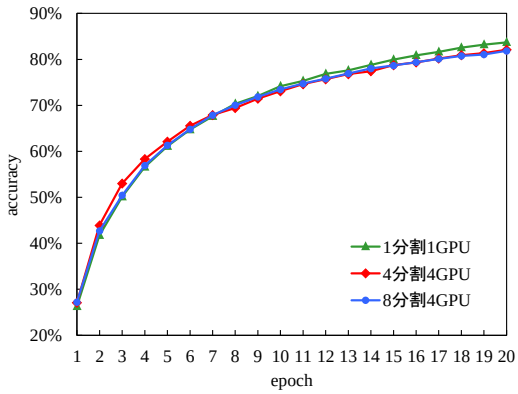

図 6　訓練データの正解率.

表 2　学習時間（エポック平均）.

	1 分割 1GPU	4 分割 4GPU	8 分割 4GPU
実行時間 [s]	5682.7	3075.1	2736.8
速度向上比 [倍]	1.000	1.848	2.076

マルチ GPU 向けの並列処理プログラムは，CUDA と OpenMP により実装した．マルチ GPU サーバ上での性能評価の結果，10 クラス画像分類 CNN の学習において，各 GPU に 1 ステージを割り当てる方法と比較して提案手法は約 89%の実行時間短縮を達成し，有効性が確認された．

参考文献

[1] Huang, Yanping, et al. GPipe: Efficient training of giant neural networks using pipeline parallelism, Proceedings of the 33rd International Conference on Neural Information Processing Systems, pp.103-112, 2019.

[2] Narayanan, Deepak, et al. PipeDream: generalized pipeline parallelism for DNN training, Proceedings of the 27th ACM Symposium on Operating Systems Principles, pp.1-15, 2019.

[3] 藤田毅. C++で学ぶディープラーニング, マイナビ出版, 2017.

[4] The CIFAR-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html.