Vol.2021-ARC-244 No.16
Vol.2021-SLDM-194 No.16
Vol.2021-EMB-56 No.16
2021/3/25

TensorFlowアプリケーション用GPUサーバにおけるNVDIMMの利用可能性の検討

松下 哲也[1,a)]　三輪 忍[1]　八巻 隼人[1]　本多 弘樹[1]

概要：GPU サーバによる深層学習では GPU への訓練データ転送を効率よく行うことが学習時間を短縮する上で鍵となる．深層学習フレームワーク TensorFlow には，訓練データ転送と GPU による学習を並列に実行する入力パイプラインがあり，これを利用して高速な学習を行う．入力パイプラインが機能するケースでは，GPU を搭載するホストマシンの処理性能が多少低下したとしても学習時間に与える影響はほとんどない．一方，最近は NVDIMM のような次世代メモリが登場し始めているが，こうした次世代メモリは DRAM よりも大容量で待機電力が少ないもののアクセスレイテンシが大きいことから，その用途はまだ限られている．そこで本研究では，入力パイプラインが機能するための境界条件式を導出し，入力パイプラインが機能するために必要なメモリ性能を明らかにすることによって，TensorFlow アプリケーション用 GPU サーバにおける NVDIMM の利用可能性の検討を行う．評価の結果，2 つのネットワークモデルにおいて，メモリバンド幅 19GB/s，レイテンシ 240ns までのメモリ性能の低下を許容できることがわかった．

キーワード：NVDIMM，TensorFlow アプリケーション，入力パイプライン

1. はじめに

深層学習は計算量が多く，また，多くのメモリ量を必要とすることから，GPU を使用して計算が行われることが多い．特に近年は，TensorFlow を始めとする多くの深層学習フレームワークが GPU の利用をサポートしたことによって，GPU を搭載した高性能計算環境上で TensorFlow アプリケーションが実行されるケースが増加している．

GPU サーバ上で TensorFlow アプリケーションを実行した場合は，ホストマシンにおける訓練データに対する前処理，および，ホストマシンから GPU への訓練データの転送処理によって性能が悪化してしまうことが多い．この問題を緩和するため，TensorFlow には上記の処理と GPU による学習処理をパイプライン処理する仕組み (入力パイプライン) が備わっている．入力パイプラインが最も機能した場合には，ホストマシンによる訓練データの前処理時間と GPU へのデータ転送処理時間は GPU による学習時間によって完全に隠蔽され，アプリケーションの実行時間は GPU による学習時間によってほぼ支配される．

逆に言えば，TensorFlow を用いた深層学習では，入力パイプラインが機能する限りはホストマシンの処理性能が性能上のボトルネックとなることはない．ネットワークモデルや訓練データセットにもよるが，GPU による学習時間は訓練データの前処理時間やデータ転送時間よりも一般的に長い．そのため，TensorFlow アプリケーション用のホストマシンには低性能なハードウェアが利用できる可能性が高いと言える．

現在，多くのコンピュータはメインメモリに DRAM を使用しているが，最近 NVDIMM のような次世代メモリが登場し始めている．こうした次世代メモリは DRAM よりも大容量で待機電力が少ないというメリットはあるが，アクセスレイテンシが大きいというデメリットがあるため，その用途はまだ限られている．次世代メモリの用途を拡大するためには，まだ知られていない次世代メモリのユースケースを示すとともに，次世代メモリに対する要求性能を明らかにする必要がある．

そこで，本研究では，GPU サーバで TensorFlow アプリケーションを実行する状況が NVDIMM のユースケースとなり得るかを検討する．本論文では，TensorFlow の処理をモデル化して入力パイプラインが機能するための境界条件式を導出し，この境界条件式を用いて次世代メモリをメインメモリに用いた場合でも入力パイプラインが機能することを示す．また，入力パイプラインが機能するために必要なメモリ性能を明らかにすることによって，TensorFlow

[1] 電気通信大学
1-5-1, Chofugaoka, Chofu, Tokyo 182–8585, Japan
[a)] t.matsushita@hpc.is.uec.ac.jp

ⓒ 2021 Information Processing Society of Japan

Vol.2021-ARC-244 No.16
Vol.2021-SLDM-194 No.16
Vol.2021-EMB-56 No.16
2021/3/25

アプリケーション用 GPU サーバにおける利用を想定した次世代メモリの設計指針を得ることを目指す．

2. 背景

2.1 深層学習

深層学習は 4 層以上の多層ニューラルネットワークを用いて機械学習を行う技術である．現在，多くの分野で深層学習の有効性が確認されており，画像認識もそのうちの 1 つである．本実験では多層ニューラルネットワークの 1 つである CNN (CNN: Convolutional Neural Network) を使用するが，CNN は画像認識などに広く利用されているネットワークモデルである．その背景には，2012 年開催の一般物体認識のコンテスト ILSVRC で CNN が勝利したことなどがある [1].

ニューラルネットワークにおける処理に「学習」がある．「学習」とはある問題を解くためにニューラルネットワークの重みをチューニングするステップであり，多くの訓練データを用いる．近年，大規模なデータを容易に入手できるようになったことにともない，ニューラルネットワークは高いレベルの認識精度を実現するようになっている．特に画像認識では教師あり学習が利用されることが多く，教師あり学習ではネットワークの出力値と正しい値（教師信号）の誤差が小さくなるように重みの更新が行われる．学習の方法はいくつかあるが，現在，ミニバッチ学習が広く用いられている．ミニバッチ学習は訓練データをいくつかのグループに分け，グループごとに繰り返し学習を進める．それぞれのグループに含まれるデータの数をバッチサイズまたはミニバッチサイズという．

2.2 深層学習と GPU サーバ

深層学習では，ストレージやメモリから大量の訓練データを読み出し，ネットワークの出力と誤差信号を計算し，大量のパラメータに対して更新を行う，という一連の処理が繰り返される．そのため，深層学習を実行するハードウェアには高いメモリバンド幅と高い演算性能の両方が要求される．そのようなハードウェアとして TPU や FPGA などの専用ハードウェアを利用するケースもあるが，クラウド環境に設置された GPU サーバを利用するケースが一般には多い．

GPU は CPU よりもコア数がけた違いに多く，並列性を持った計算処理が得意である．ニューラルネットワークの学習アルゴリズムは，通常，分岐や複雑な制御が少ないため，GPU での実行に適している．また，ニューラルネットワークは同じ層の他のニューロンとは独立して処理できる多数の個別の「ニューロン」に分離することができるので，GPU による並列計算の恩恵を受けやすい．

高い性能を持つ GPU 用のコードを書くには通常はかなりの労力を要するが，深層学習用のソフトウェアライブラリが GPU ベンダーによって用意されており，エンドユーザはこのライブラリを利用することで GPU の高い演算性能を簡単に引き出すことができる．また，現在は，後述する TensorFlow を始めとする多くの深層学習フレームワークが GPU による実行をサポートしており，エンドユーザは上記のフレームワークを利用して記述した自身のプログラムを改変することなく GPU を学習に利用できる．

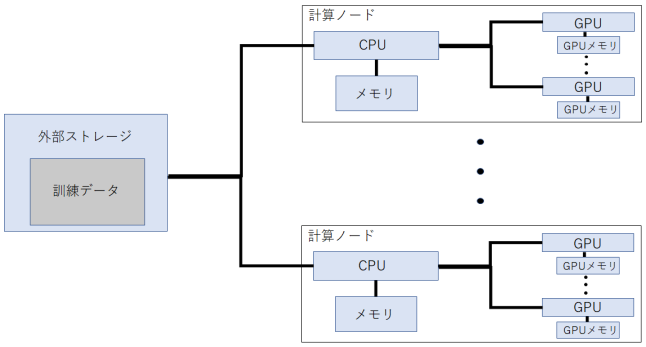

図 1: GPU サーバ環境

GPU サーバの一般的な構成を図 1 に示す．GPU サーバでは，複数台の GPU を搭載した計算ノード同士がネットワークを介して接続されており，さらに各計算ノードが外部ストレージとネットワークを介して接続される．エンドユーザは自身のタスクの規模に応じて計算に使用する GPU を 1 台以上選択し，深層学習アプリケーションを実行する．

GPU サーバ上で深層学習アプリケーションが実行されると，以下の処理が行われる．

まず最初に，外部ストレージに格納されたパラメータが GPU へ転送される．後述する訓練データの転送とは異なり，パラメータの転送は学習の開始時と終了時のみ発生し，学習中は基本的には発生しない．これは，パラメータは学習中も GPU メモリ内に保持され続けており，更新処理は GPU メモリ内のパラメータに対してのみ行われるためである．

GPU へのパラメータの転送が完了すると，以下の 4 つの処理が行われる．

CPU へのデータ転送　外部ストレージに格納された訓練データの中から，CPU が学習に使用するミニバッチを選択して読み出す．CPU によって読み出されたミニバッチは，計算ノード内のメインメモリに格納される．

CPU による前処理　読み出したミニバッチに対してデータ形式の変換などの前処理を CPU が行い，結果をメインメモリに格納する．

GPU へのデータ転送　CPU が処理したミニバッチを GPU へ転送する．

GPU による学習　GPU メモリに格納されたミニバッチとパラメータを使用してミニバッチ学習を行う．

ⓒ 2021 Information Processing Society of Japan

Vol.2021-ARC-244 No.16
Vol.2021-SLDM-194 No.16
Vol.2021-EMB-56 No.16
2021/3/25

以降は上述した 4 つの処理を繰り返すことで学習が進み，指定された繰り返し回数に達するなどの終了条件に達すると学習は終了する．

2.3 TensorFlow

TensorFlow は現在最も広く利用されている深層学習フレームワークであり，GPU による実行をサポートしている．TensorFlow では深層学習アルゴリズムを高性能演算機能を提供するライブラリの呼び出しとして記述することで，エンドユーザは学習したいネットワークモデルを簡単に構築することができる．

また，TensorFlow は性能分析のための環境も充実している．Tensor Board は TensorFlow の可視化ツールキットであり，これを利用することで損失や精度の可視化，TensorFlow プログラムのプロファイリングなどを行うことができる．プロファイリングでは CPU 上での処理から GPU 上での処理まで確認が可能であり，処理ごとの所要時間などのパフォーマンス情報を収集し，可視化することも可能である．

2.4 入力パイプライン

前々節で述べたように，GPU サーバ上で深層学習アプリケーションを実行すると，(1)CPU へのデータ転送，(2)CPU による前処理，(3)GPU へのデータ転送，(4)GPU による学習の 4 つの処理が繰り返される．ここで (1)∼(3) は基本的にはホストマシン上（CPU，メインメモリ，ストレージ）で行われる処理であり，(4) の GPU 上で行われる処理とは異なるため，(4) の処理と (1)∼(3) の処理はパイプライン化して並列に実行できる．

GPU サーバ上で入力パイプラインを使用せずにミニバッチ学習を行った場合の各ハードウェアの使用状況を図 2 に示す．まず最初のミニバッチ 1 に対する (1)∼(3) の処理が行われ，その後，ミニバッチ 1 に対する (4) の処理が GPU で行われる．そして，(4) の処理が GPU で完了すると，次のミニバッチ 2 に対する (1)∼(3) の処理が開始される．入力パイプラインを使用しない場合は，(1)∼(3) の処理をホストマシンが行っている間は GPU がアイドル状態となり，逆に (4) の処理を GPU が行っている間はホストマシンがアイドル状態となる．

これに対し，入力パイプラインを使用してミニバッチ学習を行った場合の各ハードウェアの使用状況を図 3 に示す．図 2 とは異なり，GPU 上で行われるミニバッチ 1,2,··· に対する (4) の処理と並行して，ホストマシン側で次のミニバッチ 2,3,··· に対する (1)∼(3) の処理が行われる．したがって，(4) の処理時間が (1)∼(3) の処理時間の合計よりも長い場合は，(1)∼(3) の処理時間が (4) の処理時間によってすべて隠蔽されることになり，アプリケーション全体の実行時間に影響しなくなる．

図 2: TensorFlow 入力パイプラインを用いない場合

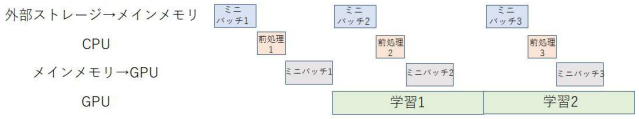

図 3: TensorFlow 入力パイプラインを用いた場合

一般には (4) の処理時間の方が (1)∼(3) の処理時間の合計よりも長いため，入力パイプラインを使用することで TensorFlow アプリケーションの実行時間を短縮できることが知られている [2]．このように，入力パイプラインは TensorFlow プログラミングにおける重要な性能チューニング技術の 1 つである．

3. TensorFlow 入力パイプラインのモデリング

前述のように，GPU サーバ上で TesnorFlow アプリケーションを実行して入力パイプラインが機能した（(1)∼(3) の処理時間が (4) の処理時間によって完全に隠蔽された）場合，TensorFlow アプリケーションの全体の実行時間は GPU の学習時間によって律速される．そのため，入力パイプラインが機能する限り，(1)∼(3) の処理時間が増加してもアプリケーションの実行時間は悪化しないことが予想される．

そこで本研究では，深層学習用 GPU サーバにおける NVDIMM の利用可能性を検討する．具体的には，TensorFlow アプリケーションの実行時間を増加させない範囲で，GPU サーバのメインメモリをどこまで低性能なメモリに変更できるかを検討する．上記の目的を達成するために，本論文では TensorFlow の入力パイプラインが機能するための境界条件式を提案する．

入力パイプラインが機能するための条件は式 (1) によって表すことができる．

$$\frac{D}{T_1} + \frac{D}{T_2} + T_{CPU} \leq T_{GPU} \qquad (1)$$

D：ミニバッチのデータサイズ (GB)
T_1：ストレージと CPU 間の転送速度 (GB/s)
T_2：CPU と GPU 間の転送速度 (GB/s)
T_{CPU}：CPU におけるミニバッチの前処理時間 (s)
T_{GPU}：GPU におけるミニバッチの学習時間 (s)

上記の式において，D/T_1 はストレージからメインメモリへのミニバッチの転送に要する時間，D/T_2 はメインメモリから GPU へのミニバッチの転送に要する時間を表す．

ⓒ 2021 Information Processing Society of Japan

Vol.2021-ARC-244 No.16
Vol.2021-SLDM-194 No.16
Vol.2021-EMB-56 No.16
2021/3/25

CPU ではミニバッチに対してさまざまな前処理が行われるが，その中でも特に次の 3 つの関数が処理時間の大きな割合を占める．1 つ目は訓練データである JPEG 画像をデコードする decode jpeg である．2 つ目は訓練データのサイズを変更する resize である．3 つ目は型を変換する cast である．そこで，上記 3 つの関数 1 回分の処理時間をそれぞれ T_{decode},T_{resize},T_{cast} としてモデル化する．これらの 3 つの関数は 3 つで 1 セットとなっており，必ず上記の順序で実行される．また，CPU による前処理はマルチスレッド化されており，各スレッドがこれらの関数を複数回実行することで 1 つのミニバッチに対する前処理を行う．したがって，T_{CPU} は，各関数のスレッドあたりの実行回数を n とすると，以下の式 (2) で表される．

$$T_{CPU} = n * (T_{decode} + T_{resize} + T_{cast}) \tag{2}$$

次に，CPU による前処理時間とメインメモリ性能の関係をモデル化する．前処理中にメインメモリに割り当てられたデータサイズを $d_{allocation}$ とする．メモリに割り当てられたデータには必ず 1 回はメモリアクセスが行われると考える．メモリに割り当てられたデータサイズを前処理中の関数の実行回数で割ることで，関数が 1 回の実行あたりに処理するデータのサイズである d_{mem} を求めることができる．これは式 (3) で表すことができる．

$$d_{mem} = \frac{d_{allocation}}{n \times \text{スレッド数}} \tag{3}$$

以後，d_{mem} を各関数が 1 回のメモリアクセスで読み出すデータサイズと仮定し，単一の関数内では同じデータに複数回のメモリアクセスが行われると考える．

メモリ性能はバンド幅とレイテンシによって定義される．サイズ d_{mem} のデータがメモリ上の連続する領域に配置されているとき，このデータへのアクセスに要する時間は式 (4) で表される．

$$T_{mem} = L + \frac{d_{mem}}{B} \tag{4}$$

d_{mem} : メモリから読み出すデータサイズ (byte)
B : メモリバンド幅 (GB/s)
L : メモリレイテンシ (ns)

Tensor Board を用いて CPU の前処理のプロファイリングを行ったところ，CPU の処理時間の大部分がメモリアクセス時間によって占められていることが確認できた．そこで, 3 つの関数の 1 回の実行時間である T_{decode},T_{resize},T_{cast} は，全てメモリアクセスに要する時間と考える．このように考えると，各関数の 1 回あたりの実行時間は式 (5)〜(7) で表される．

表 1: TSUBAME3.0 のシステム構成

計算ノード	540 台
CPU	Intel Xeon E5-2680 v4 2.4GHz × 2CPU
GPU	NVIDIA TESLA P100 for NVlink-Optimized Servers × 4
メモリ	256GB (DDR4-2400 32GB × 8)
ローカルストレージ	SSD 2TB (NVMe, PCI-E 3.0 x4)
外部ストレージ	DDN EXAScalerx3 合計 15.9PB, 150GB/s
CPU と GPU の接続	× 16 PCIe 16GB/s

$$T_{decode} = \alpha_{decode} * T_{mem} \tag{5}$$

$$T_{resize} = \alpha_{resize} * T_{mem} \tag{6}$$

$$T_{cast} = \alpha_{cast} * T_{mem} \tag{7}$$

α_{decode} : decode jpeg 1 回の実行で行われるメモリアクセスの回数
α_{resize} : resize 1 回の実行で行われるメモリアクセスの回数
α_{cast} : cast 1 回の実行で行われるメモリアクセスの回数

以上のモデル式を用いて，メモリ性能 (バンド幅とレイテンシ) を変更した場合に式 (1) の境界条件式を満たすか否かを判定できる．

4. 実験方法

4.1 実験環境

本実験は東工大が運用するスーパーコンピュータ TSUBAME3.0 上で, TensorFlow を用いて行った．TSUBAME3.0 のシステム構成を表 1 に示す．評価に使用した TensorFlow のバージョンは 2.3.1 である．今回学習に使用したデータセットは ImageNet(147.5GB,1,281,167 枚)[3] である．

4.2 評価方法

TensorFlow のプロファイリングツールである TensorBoard[4] を使用し，深層学習アプリケーションのプロファイリングを行った．今回の実験では EfficientNet[5] と ResNet50[6] の 2 つのネットワークモデルで評価を行った．使用した GPU 数は 1 つである．

最初に，ミニバッチサイズを 64，128，256 としたときの 3 つの関数（decode jpeg，resize，cast）1 回分の処理時間である T_{decode}，T_{resize}，T_{cast} を計測した．

続いて，それぞれのミニバッチにおける 3 つの関数のスレッド 1 つあたりの実行回数である n を計測した．また，TesorBoard 用に生成された JSON ファイルから学習中に CPU メモリに割り当てられたデータサイズ $d_{allocation}$ を計測した．上記のようにして求めた $d_{allocation}$ と n から各関数が 1 回の実行で処理の対象とするデータのサイ

ⓒ 2021 Information Processing Society of Japan

ズである d_{mem} を求め，d_{mem} と DDR4-2400 の仕様から TSUBAME3.0 におけるメモリアクセス 1 回あたりにかかる時間 T_{mem} を求めた．

続いて，3 つの関数それぞれの 1 回の実行時間である T_{decode}，T_{resize}，T_{cast} とメモリアクセス 1 回あたりにかかる時間 T_{mem} より，関数 1 回の実行中に行われるメモリアクセスの回数である α_{decode}，α_{resize}，α_{cast} を求めた．

最後に，以上の結果から T_{CPU} を求め，3 章で示した境界条件式である式 (1) による検討を行った．それぞれのミニバッチの学習時間 T_{GPU} を計測した．続いて，式 (1) をもとに，メモリバンド幅 B とレイテンシ L について，メモリ性能をどこまで許容できるのかについて確認を行った．B=152,76,38,19,10， L=15,30,60,120,240 として，全ての組み合わせ 25 通りについて評価を行った．また，異なる L の値に対し，B=19 で固定した場合，異なる B の値に対し，L=120 で固定した場合の 2 つについての評価を行った．

なお，評価には [7] よりダウンロードしたコードを使用した．このコードには畳み込みニューラルネットワークの TensorFlow による実装があり，Tensor Board をサポートしている．

5. 評価結果

5.1 前処理のプロファイリング

ミニバッチサイズを 64，128，256 として計測した．T_{decode}，T_{resize}，T_{cast} の結果，各ミニバッチに対する 3 つの関数のスレッド 1 つ当たりの実行回数 n をまとめた結果，学習中にメモリに割り当てられたデータサイズ $d_{allocation}$ を求めた結果，メモリアクセス 1 回あたりメモリから読み出すデータサイズ d_{mem} を求めた結果，それぞれの関数 1 回の実行中のメモリアクセス回数 α_{decode}，α_{resize}，α_{cast} を求めた結果を表 2 に示す．α_{decode}，α_{resize}，α_{cast} を求める際は式 (4) を使用したが，同式で用いる値 B，L に関しては，実験環境である TSUBAME3.0 のハードウェア構成より，B=152，L=15 とした．また，Tensor Board で確認した結果，異なるサイズのミニバッチを学習した場合でも使用されるスレッドの数は変わらず，全ての学習において等しく 13 であった．

表より，各ミニバッチに対する 3 つの関数の n の値について，関数による値の変化がほとんどないため，以降の計算ではミニバッチサイズが 64 の場合は $n = 5$，ミニバッチサイズが 128 の場合は $n = 10$，ミニバッチサイズが 256 の場合は $n = 20$ とする．また，T_{decode}，T_{resize}，T_{cast} の結果より，ミニバッチを大きくした場合は，1 回の関数でより多くのデータにアクセスするのではなく，関数の実行回数を増やすことでより多くのデータにアクセスしていることがわかる．

表 2: 前処理のプロファイリング結果

	64	128	256
T_{decode}(ns)	2,625,371	2,547,397	2,299,896
T_{resize}(ns)	862,558	869,313	866,363
T_{cast}(ns)	694,072	729,368	695,602
n(decode)	5.00	9.92	19.5
n(resize)	4.85	10.0	19.5
n(cast)	4.93	10.0	19.6
$d_{allocation}$(byte)	58,965,606	114,330,214	209,371,750
d_{mem}(byte)	907,163	879,463	805,273
α_{decode}	439	439	433
α_{resize}	144	150	163
α_{cast}	116	126	131

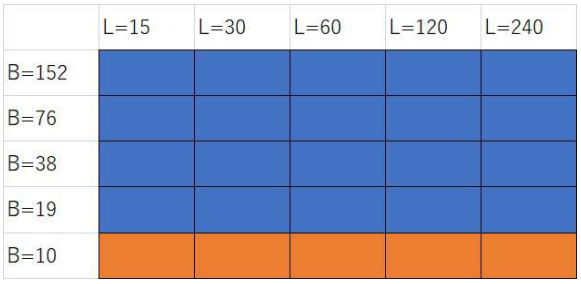

図 4: EfficientNet によるメモリ性能の評価

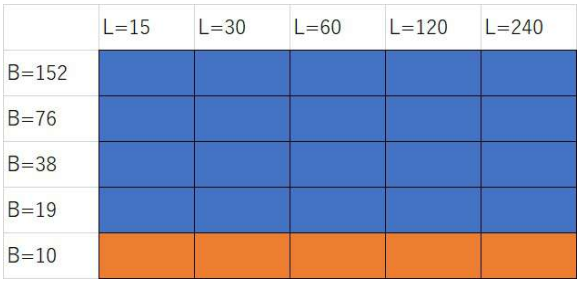

図 5: ResNet50 によるメモリ性能の評価

5.2 境界条件式によるメモリ性能の検討

提案する境界条件式を用いて，入力パイプラインが機能する限界メモリ性能を評価した．EfficientNet と ResNet50 の結果をそれぞれ，図 4 と図 5 に示す．全てのミニバッチにおいて結果が同様となったため，図は 1 つにまとめている．境界条件式が成り立つ場合は青，成り立たない場合はオレンジになっている．また，異なる L の値に対し，B=19 で固定した場合，異なる B の値に対し，L=120 で固定した場合の結果を，それぞれ表 3，表 4 に示す．

図 4，図 5 より，EfficientNet と ResNet50 どちらのネットワークモデルの場合でも $B = 10$ のときで境界条件式が成り立たなくなっている．よって，今回の実験環境ではメモリバンド幅 `B=19GB/s`，レイテンシ `L=240ns` がメモリの性能として最低限必要であることが確認できた．これは，TSUBAME3.0 のメインメモリ性能（DDR4-2400,8 チャネル）に対して，メモリバンド幅は 8 分の 1，レイテンシは 16 倍の性能である．

ⓒ 2021 Information Processing Society of Japan

Vol.2021-ARC-244 No.16
Vol.2021-SLDM-194 No.16
Vol.2021-EMB-56 No.16
2021/3/25

表 3: 異なる L の値に対して B=19 で固定した場合

	64	128	256
L=15	0.1675292	0.3322693	0.6188969
L=30	0.1675816	0.3323765	0.6191004
L=60	0.1676864	0.3325910	0.6195366
L=120	0.1678961	0.3330200	0.6204090
L=240	0.1683155	0.3338780	0.6221538
EfficientNet の学習時間	0.2088965	0.3870324	0.7387873
ResNet50 の学習時間	0.2790608	0.4614249	0.8920058

表 4: 異なる B の値に対して L=120 で固定した場合

	64	128	256
B=152	0.0218855	0.0434379	0.0812077
B=76	0.0427437	0.0848078	0.1582406
B=38	0.0844635	0.1675476	0.3122919
B=19	0.1678961	0.3330200	0.6204090
B=10	0.3180798	0.6308819	1.1750519
EfficientNet の学習時間	0.2088965	0.3870324	0.7387873
ResNet50 の学習時間	0.2790608	0.4614249	0.8920058

現在，NVDIMM の実現方式はいくつか存在するが，最新技術の 1 つに DRAM と不揮発性メモリを混載した NVDIMM-P がある．NVDIMM-P 技術は NVDIMM のメリットである記憶容量の拡大，消費電力の低減を実現しながらも，DRAM に近い高速なアクセスが可能である．バンド幅は DRAM と同程度の性能である一方で，レイテンシは数百 ns まで性能が低下する [8].

今回の実験結果から考えると，メモリバンド幅が低下しない場合はレイテンシの性能が低下していても，TensorFlow アプリケーションにおいて性能低下を引き起こす可能性は少ないと言える．したがって，深層学習用 GPU サーバのメインメモリを DRAM から NVDIMM-P に変更するのは十分現実的な選択肢である．

表 3 より，異なるレイテンシ L に対してメモリバンド幅 B を固定した場合，境界条件式の左辺の値はほとんど変化していない．一方で，表 4 より，異なる B に対して L を固定した場合，大きな変化が見られる．ミニバッチ学習では，レイテンシよりもメモリバンド幅の方が実行時間に与える影響が大きいことが分かる．実験結果によると，今回の実験環境では 1 回のメモリアクセスにかかる時間のうち，レイテンシが 15ns であるのに対し，データを読み出す時間は 5,000ns 程になっている．また，表 2 より，ミニバッチの処理のために行われるメモリアクセスは高々 1,000 回ほどであり，たとえレイテンシが大きくなったとしても TensorFlow アプリケーションの実行時間に与える影響は小さいと言える．

今回の実験では 2 つのネットワークモデル EfficientNet と ResNet50 による評価を行った．ResNet50 は 50 層からなるネットワークモデルであり，パラメータ数は 2,600 万に及ぶ．一方で，EfficientNet のパラメータ数は ResNet50 の 5 分の 1 ほどであり [5]，ミニバッチ分の学習時間からも，そのことが確認できる．他のネットワークモデルでも同等の処理が CPU で行われると考えるのならば，ResNet50 よりパラメータ数の多いネットワークモデルを学習する場合は，GPU での学習により時間がかかると考えられる．つまり，今回の 2 つのネットワークモデルでは最低限 B=10, L=240 のメモリ性能が必要であったが，さらにメモリ性能の低下を許容できる可能性がある．

6. 結論

本論文では，TensorFlow アプリケーションの入力パイプラインが機能する条件を表す境界条件式を提案し，いくつかの TensorFlow アプリケーションに対する評価を行った．評価の結果，今回実験を行った環境ではメモリ性能の低下をメモリバンド幅 `19GB/s`，レイテンシ `240ns` まで許容できることを確認した．また，メモリ性能を低下させる場合，レイテンシの悪化がアプリケーションの実行時間に与える影響は軽微であることが確認できた．以上の結果より，TensorFlow アプリケーションを実行する GPU サーバにおいて，ホストマシンのメインメモリに NVDIMM が利用できる可能性が高いことを示した．

今後は本論文で提案した境界条件式を用いてより多くの TensorFlow アプリケーションに対する評価を行い，深層学習用 GPU サーバにおける NVDIMM の利用可能性の検討をさらに進める予定である．

謝辞 本研究の一部はキオクシア株式会社の支援を受けて実施したものです．

参考文献

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ”Imagenet classification withdeep convolutional neural networks” (2012)
[2] Better performance with the tf.data API — TensorFlow Core. `https://www.tensorflow.org/guide/data_performance`.
[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database” IEEE Computer Vision and Pattern Recognition (CVPR), (2009)
[4] Tensor Board `https://www.tensorflow.org/tensorboard?hl=ja`
[5] Mingxing Tan, Quoc V. Le: “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks” (2019)
[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun: “Deep Residual Learning for Image Recognition” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition , pp. 770-778 , (2016)
[7] tensorflow/benchmarks/perfzero at master `https://github.com/tensorflow/benchmarks/tree/master/perfzero`.
[8] A Sainio - Memory Computing Summit, “NVDIMM: changes are here so what’s next” (2016)

ⓒ 2021 Information Processing Society of Japan