Vol.2019-HPC-170 No.6
2019/7/24

大規模GPUクラスタにおけるResNet-50/ImageNet学習の高速化

本田 巧[1,a)] 三輪 真弘[1] 山崎 雅文[1] 笠置 明彦[1] 田渕 晶大[1] 福本 尚人[1] 田原 司睦[1] 池 敦[1] 中島 耕太[1]

概要：ディープニューラルネットワークの学習に必要な計算量は急速に増加しており, 単一プロセッサによる学習では多くの時間を要するようになった. そのため, クラスタコンピュータを用いた大規模分散学習による学習時間短縮のための研究が行われている. 大規模分散学習ではデータ並列でスケールするに従い, 1 回の学習処理で使用するデータ数 (ミニバッチサイズ) をより大きくする必要があるが, この場合最終的に到達する学習精度が低下するという問題がある. 我々は, 新たに巨大ミニバッチでの学習精度向上技術を開発し, 従来よりも大きなミニバッチサイズ 81,920 で ResNet-50 の学習精度 75.08%を達成した. また, 大規模分散学習の高速化技術を開発, 深層学習フレームワークに適用し, GPU クラスタである ABCI の 2,048 基の GPU を用いた ResNet-50/ImageNet の分散学習において, 1.73 million images/sec のスループットを達成し, 74.7 秒で学習を完了した.

1. はじめに

深層学習 (Deep Learning) では, 様々な Deep Neural Network (DNN) が提案されており, それらのネットワークは自然言語処理や画像識別, 物体検出などの分野で素晴らしい成果を上げている. しかし, DNN の構成や DNN 学習に用いるデータセットが大きくなるにつれ, 学習に必要な計算量が増大しており, 単一プロセッサによる学習では膨大な時間が必要になっている. この問題の解決策として, HPC システム上でのデータ並列による分散深層学習が注目されている. このアプローチでは, システム上に起動された全てのプロセスが同じ DNN と初期重みで学習を開始する. 各プロセスは並列に異なる入力データ (ミニバッチ) による学習を行うが, 重みの更新には全プロセスでの Allreduce 処理による重み更新量の集約が必要になる. HPC システムでの大規模分散学習では, Allreduce 処理のための通信オーバーヘッドが学習速度の低下をもたらす. ミニバッチサイズは通信量と独立であるため, ミニバッチサイズを増加させることで Allreduce 処理あたりの計算量を増やすことができ, 通信のオーバーヘッドを隠蔽することができる. しかし, 巨大なミニバッチによる大規模分散学習は DNN の精度低下につながることが知られている [1] [2].

[1] 株式会社富士通研究所
Fujitsu Laboratories Ltd.
[a)] honda.takumi@fujitsu.com

巨大なミニバッチを用いた分散学習での精度低下を防ぐ手法として様々な手法が提案されている [2] [3] [4]. これらの手法を適用することで, 精度低下を起こさず学習可能なミニバッチのサイズは増加し, ミニバッチサイズ 65,536 の巨大なミニバッチでの学習が実現した.

我々は新たに巨大なミニバッチを用いた大規模分散学習のための精度向上技術を開発した. 従来の技術に加え, 我々の技術を適用することで, より巨大なミニバッチでの学習を精度低下なしで行うことができる. また, HPC システム上で大規模分散学習を効率的に行うための高速化技術も開発した. これらの技術を適用することで, GPU クラスタ ABCI [5] 上の 2,048 基の GPU を用いたミニバッチサイズ 81,920 での ResNet-50 [6] の大規模分散学習で, 75.08%の精度を 74.7 秒で達成した.

本論文の構成は以下のとおりである. 2 章で, 関連研究について述べ, 3 章で DNN 学習について説明する. 4 章では, 提案手法を述べ, 5 章で性能評価の結果を示す. 最後に 6 章で結論を述べる.

2. 関連研究

AlexNet [7] は, Convolution Layer からなる DNN が, 実用的な画素数からなる画像を対象にした ILSVRC で高い性能を実現できることを示した. Batch Normalization [8] は, 識別処理における伝達情報を正規化することで, また ResNet [6] は誤差逆伝播の勾配消失を防ぐ手法により, 100

© 2019 Information Processing Society of Japan

層を超える DNN の学習を実現した.

大規模分散学習では, 多数のプロセッサを有効利用するために, ミニバッチサイズを大きくする必要がある. 巨大なミニバッチを用いた学習では, Goyal らが学習率の調整と Warm up を提案し, ミニバッチサイズ 8,192 まで精度の劣化なく学習できることを示した [2]. Ying らと三上らは, ミニバッチサイズを学習の進捗に合わせて変化させることで最終的なミニバッチサイズを大きくしている [4] [9]. You らは, 学習時に算出される勾配の大きさ（重みパラメータとのノルム比）が層毎に大きく異なる事に着目し, 学習率を層毎に正規化する手法（LARS）を提案し, ミニバッチサイズ 32,768 での学習を可能にした [3].

秋葉らは, 1,024 基の GPU を用いて 15 分 [10], Jia らは最大 2,048 基の GPU を用いて 6.6 分で ResNet-50 の学習を完了させた [11]. また, Ying らは, TPU v3 プロセッサを用いて, 1.05 million images/sec をミニバッチサイズ 32,768 で実現し 2.2 分で学習を完了させ, さらにミニバッチサイズ 65,536 では 1.8 分で学習を完了させたと報告している [4].

3. DNN 学習

本章では, DNN 学習における処理の流れ, 学習において重要な要素となるハイパーパラメータについて説明する.

3.1 学習プロセス

DNN は, 多数のニューロンで構成される層が複数積み重なり構成されている. これらの層の間で相互にデータを伝播させ, 入力データの認識や学習を行っている. シングルプロセスでの学習の流れを図 1 に示す. 学習のプロセスは, フォワード処理とバックワード処理, 内部パラメータの更新処理から成っている.

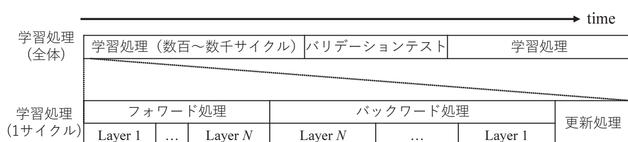

図 1 シングルプロセスでの DNN 学習の流れ

フォワード処理は認識処理とも呼ばれ, 第一の層 (ボトム層) から最終段の層 (トップ層) に向けて処理を行う. ボトム層は入力データ D と重みパラメータ w_0 を用いて演算を行い演算結果を出力する. ボトム層以降では, 前段の層の出力を入力データとし, 同様に重みパラメータ w_i (i は層のインデックス) を用いて演算を行う. トップ層の出力は, 入力データ D に対する認識結果となる.

バックワード処理では, フォワード処理で得られた認識結果と正解データから誤差関数 E を求め, 各層のパラメータの勾配情報 ∇E_i を算出する. この処理は, フォワード処理の向きとは逆向き (トップ層からボトム層) に処理が進む. バックワード処理がボトム層まで完了すると, 勾配情報 ∇E_i から内部パラメータの更新量 Δw_i を計算する.

最後に, 更新処理では, バックワード処理で得られた各層の内部パラメータの更新量 Δw_i を用いて, 重み w_i を更新する. この一連の処理を繰り返し DNN の学習が行われる.

また, 繰り返し行われる学習処理による DNN の認識精度の変化を評価するために, 定期的にバリデーションテストと呼ばれる処理が行われる. バリデーションテストでは, 学習用の入力データとは異なるデータでフォワード処理を行う. このフォワード処理での認識結果の正解率, または, 損失関数の値で DNN の精度を評価する.

3.2 ハイパーパラメータ

DNN の学習処理には, 調整が必要なハイパーパラメータが多く存在する. 学習によって DNN の認識精度を高めるためには, パイパーパラメータを適切に設定する必要がある. 本節では, 学習速度に関わる学習率とミニバッチサイズについて述べる.

3.2.1 学習率 (Learning Rate)

学習率とは, バックワード処理により算出された勾配情報を用いて重み更新量を算出する際の係数である. 一般に, 学習率が大きくなると重みが発散してしまう. また, 小さすぎると DNN の精度が収束するまでに多くの学習サイクルが必要になる.

3.2.2 ミニバッチサイズ

DNN の学習処理では, 複数の入力データをまとめて処理することで, プロセッサの演算性能を引き出している. プロセッサで一度に処理する入力データの数を b, プロセッサ数を n とすると, データ並列により一度の更新で使用されるミニバッチサイズは $b \times n$ である. ミニバッチサイズを大きくすると入力データが多くなり勾配情報が平均化されるため, 重みの更新量が安定し学習率を大きくすることが可能になる. しかし, ミニバッチサイズを一定の値以上に大きくした場合, 学習の進みが遅くなるという問題と, 最終的に到達する学習精度が低くなるという問題がある.

3.3 データ並列 DNN 学習

本節では, DNN 学習の並列化手法であるデータ並列と, データ並列を行う場合に必須となる Allreduce 処理について説明する.

データ並列による DNN 学習では, 学習を行う全てのプロセスが同じ DNN モデルを保持し, 異なる入力データで学習を行う. 学習のフォワード処理とバックワード処理を行うと各プロセスは異なる勾配情報を得る. そのため, データ並列 DNN 学習の学習サイクルでは図 2 に示すように, バックワード処理で得られる勾配情報を全プロセスで集約する. そして, 集約した勾配情報に基づき各プロセスは重

© 2019 Information Processing Society of Japan

みを更新する.

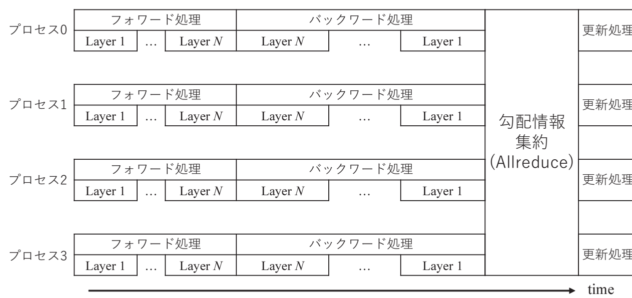

図 2 データ並列による DNN 学習の流れ

各プロセスがフォワード処理とバックワード処理を独立に実行できるため, データ並列のアプローチは高い並列性を実現できる. しかし, 勾配情報の集約には重みパラメータのサイズに比例するデータ通信が発生する. プロセス間の勾配情報の集約では Allreduce 処理を行い要素ごとの総和を算出する. 一般に, Allreduce 処理では各プロセスのベクトルデータに対して要素ごとの集約演算 (総和, 最大値, 最小値, 論理演算等) を行い, その結果を全プロセスで共有する処理である. 重みパラメータのサイズが大きい DNN の場合, また, プロセス数が多い場合, 集約処理の負荷が大きくなる.

4. 提案手法

本章では, 大規模分散学習における精度向上技術と学習速度向上技術について説明する. ターゲットの DNN は, 深層学習のベンチマークである MLPerf [12] でも使用される ResNet-50 と呼ばれる畳み込みニューラルネットワーク (Convolutional Neural Network, CNN) である.

4.1 大規模分散学習における精度向上

我々はオプティマイザとして広く知られている確率的勾配降下法 (Stochastic Gradient Descent, SGD) を用いた. ミニバッチサイズが増加するにつれて SGD の更新回数が減少するため, 巨大なミニバッチサイズによる学習では, 少ない更新回数で精度を向上させるという課題がある. 我々は, 巨大なミニバッチサイズでの学習の精度を向上させるために以下の技術を適用した.

4.1.1 適用した既存技術

ミニバッチサイズが 32,768 での学習において, Label smoothing [13] を適用することによる精度向上が報告されている [9]. この技術は, ミニバッチサイズ 81,920 においても効果があることを確認した.

Batch Normalization 層におけるバッチの平均と分散の値は, 本来全プロセスの値から計算されなければならないが, プロセス毎に独立して計算される. 巨大なミニバッチにおいては推論時に用いられる平均と分散の移動平均値は不正確になるため [10], それらの値を最適化するようにハイパーパラメータを調整した.

4.1.2 学習率の調整

少ない更新回数で学習を行うためには, 高い学習率を設定して学習を加速する必要がある. しかし, 学習率を高く設定すると学習初期にモデルが不安定になり発散しやすくなる. 学習率を小さな値から徐々に増加させる Warm up [2] を用いることで SGD を安定化させた. さらに, ミニバッチサイズが大きいことから特定の層に対する誤差値が強くなり学習が不安定になるため, Layer-wise Adaptive Rate Scaling (LARS) [3] により重みと勾配の L2 ノルムから各層の学習率を調整するようにした. 加えて, より高い精度を巨大なミニバッチサイズで実現するために, Step, Polynomial, Linear などさまざまな学習率の減衰を試し, 我々は Arc-cotangent と Warm up を組み合わせた独自の学習率スケジューリングを採用した.

Arc-cotangent スケジューリング：大規模分散学習を効率よく行うために, 我々は Arc-cotangent を用いた学習率スケジューリングを提案する. Arc-cotangent スケジューリングは, 学習率の高い期間と低い期間をそれぞれ合わせ持っており, 更新回数が少ない大規模分散学習においても, 最適値の大域的な探索と局所的な探索を行うことができる. 本スケジューリングにおける t エポック目の学習率 lr_t は以下の式で求められる.

$$lr_t = \left(arccot(f(t-e)) \times \frac{lr_0}{\pi} \right) + \frac{lr_0}{2}$$

ここで, f は原点を通る単調増加関数であり, lr_0 は初期の学習率である. e は Arc-cotangent スケジューリングにおける学習率の変更点であり, e エポック目に学習率は $\frac{lr_0}{2}$ となる. 変化量の大きい f を用いた場合, e エポック目を中心に前後で学習率が lr_0 に近い期間と 0 に近い期間が得られる.

4.2 バッチ LARS 計算の GPU 実装

我々は深層学習フレームワークとして MXNet [14] を採用した. LARS 計算では, 層ごとに重みと勾配の L2 ノルムを算出し, 学習率を調整する係数を求める. 単純にノルム計算を実装すると, 重みと勾配の L2 ノルムを GPU で計算し, ノルムに基づいた学習率のスケーリングを CPU で実行するという処理フローが各層で実行される. ResNet-50 の層の多くは要素数が少ないため層毎に LARS 計算を行うと並列性が低くなり GPU の演算性能を活かすことができない. また, ResNet-50 の各層の L2 ノルムの計算コストは小さいため, GPU の関数起動のオーバーヘッドと, 学習率のスケーリングを CPU 側で行うための GPU-CPU 間のデータの転送のオーバーヘッドが学習速度を低下させる. そのため, L2 ノルムと学習率のスケーリングを複数の層まとめて算出する LARS の GPU 関数を実装し, GPU 関数の起動

© 2019 Information Processing Society of Japan

とデータ転送のオーバーヘッドを削減した.

4.3 データ通信フレームワークの初期化最適化

本研究では, 分散学習のプロセス間のデータ通信のフレームワークとして Horovod [15] を, GPU 間の集合通信ライブラリとして NCCL [16] を採用した. Horovod では, プロセス間のデータ通信を管理する専用のスレッドをプロセス毎に起動する. Horovod のスレッドは, プロセス間のデータ通信のために, はじめに MPI のコミュニケータを初期化する. しかし, NCCL のコミュニケータの初期化は, 各プロセスの GPU 上にある勾配情報の集約のための Allreduce 処理がはじめて実行されるときに行われる. NCCL の初期化では, ルートとなるプロセスがユニークな ID を生成, 全てのプロセスに Broadcast し, 全プロセスが NCCL のコミュニケータを初期化する. ノード数が少ない場合, Broadcast のコストは小さいが, 多数のノードを使用する大規模分散学習では, Broadcast のコストが大きくなり最初の学習サイクルの学習時間が長くなる. そこで, NCCL の初期化を Horovod のスレッドの初期化フェーズで行えるようにした. 初期化フェーズで行うことで, DNN モデルの初期化や入力データの読み込みとオーバーラップさせることができ, NCCL の初期化コストを隠蔽することができる.

4.4 通信最適化

データ並列による DNN 学習では, 全プロセスで勾配情報を集約するために Allreduce 処理が必要になる. 大規模分散学習では, プロセスあたりのミニバッチサイズが小さいため, フォワード処理とバックワード処理の計算コストが小さくなるのに対して, プロセス数が増加することにより Allreduce 処理のコストは増加する. そのため, 大規模分散学習では Allreduce 処理による通信オーバーヘッドが無視できない. 通信オーバーヘッドを軽減するために以下の 2 つの通信最適化を行った.

通信のデータサイズ調整：DNN は多数の層で構成されており, 各層の要素数も様々である. Allreduce 処理を層毎に行うと, 小さいデータサイズに対して頻繁に集合通信を行うことになり, 通信ネットワークのバンド幅を活かすことができない. 我々は, 複数層の勾配情報をまとめて 1MB 以上のデータに対して Allreduce 処理を行うようにした.

通信スケジューリング：各層の勾配情報の Allreduce 処理は, 層のバックワード処理が完了すると実行可能になる. そのため, DNN のトップ層からボトム層に向けて順に Allreduce 処理が可能になる. Allreduce 処理が実行可能になった層から順に行っていくことで, バックワード処理とのオーバーラップ実行が可能である. 我々は, できる限りバックワード処理と Allreduce 処理をオーバーラップさせるために DNN の層を複数のグループに分け, グループ単位で Allreduce 処理を実行するようにした. バックワード処理とオーバーラップしやすくするために, トップ層に近い層で構成されるグループの層の数は少なく, ボトム層に近い層で構成されるグループの層の数は多くなるようにした. 最初のグループの層数を少なくすることで, 数層のバックワード処理後に最初の Allreduce 処理が実行できるようになる.

5. 性能評価

本章では, 評価環境と提案手法の性能評価の結果について報告する.

5.1 評価環境

評価環境として, 産総研の AI 橋渡しクラウド (ABCI) を利用した. ABCI は, 図 3 に示すように, 34 ノードを搭載した 32 ラックで構成されている. 同一ラック内の計算ノード間は, 図 3 に示すように 7 つのスイッチによる 2 段構成でフルバイセクションのバンド幅 7200Gbps 相当で接続されている. 一方で, 異なるラック間のスイッチの接続はオーバーサブスクリプション構成であり, ラック間の計算ノード間は 1/3 である 2400Gbps 相当のバンド幅である.

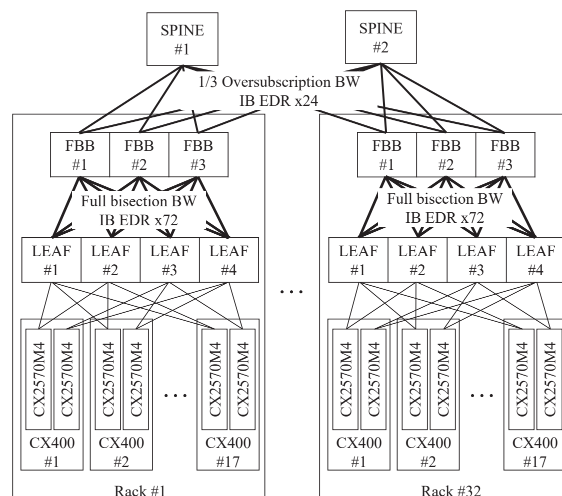

図 3 ABCI の計算ノードのインターコネクト

ABCI の計算ノードは, 図 4 に示すように Intel Xeon Gold 6148 を 2 基搭載しており, アクセラレータとしては, HPC 向けの GPU である NVIDIA Tesla V100 を 4 基搭載している. 図 4 からも分かるように, 同じ CPU に接続されている 2 基の GPU は PCI スイッチを介して PCIe Gen3 ×16 を共有している. また, 4 基の GPU は互いに 2 本の NVLink2 で接続されており, GPU 間の高速なデータ転送が可能である. 計算ノードは 2 つの IB HCA(InfiniBand Host Channel Adapter) を持ち, 2 本の Infiniband EDR が異なる Leaf スイッチに接続されている.

4 章で述べたように, 我々は深層学習フレームワークとして MXNet, プロセス間のデータ通信フレームワークと

© 2019 Information Processing Society of Japan

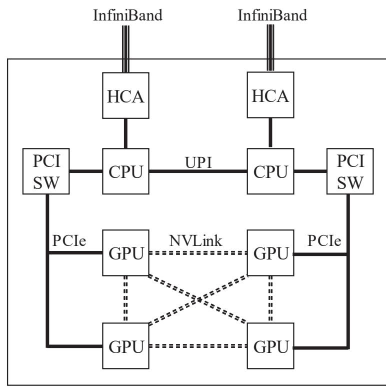

図 4 ABCI の計算ノード構成

して Horovod を採用し, 提案手法を適用した. Horovod で使用される集合通信ライブラリ NCCL は, NVIDIA が開発している GPU マルチノード環境向けの高速通信ライブラリである. Allreduce 通信について従来 Ring 方式のアルゴリズムのみ実装されていたが, NCCL 2.4 より Tree 方式のアルゴリズムが実装された. Tree 方式では, 通信に参加するプロセスで二分木を構成し, リーフからルートに向かってデータを集約し, その後ルートからリーフに向かってブロードキャストすることでツリーの高さ, すなわちプロセス数の対数オーダに比例する遅延時間となる.

本計測で用いたソフトウェアを表 1 に示す. また, 本実装では 1 プロセスに 1 基の GPU を割り当て, 各プロセスが異なる GPU で学習処理を行う. 入力のデータセットとしては ImageNet [17] を用いた.

表 1 計測で使用した主なソフトウェア

	Version
Python	3.6
GCC compiler	7.3.0
OpenMPI	2.1.6
CUDA	9.2
cuDNN	7.5
NCCL	2.4.2
DALI	0.6.1

5.2 学習率の推移と精度

図 5 に, 2,048 基の GPU によるミニバッチサイズ 81,920 での ResNet-50 の学習の精度の推移を示す. 本評価の学習率スケジューリングは Arc-cotangent であり, 学習率の推移も図 5 に示している. 15 エポックあたりまでは Warm up の期間とすることで, 重みの発散を防止している. その後, 60 エポックを超えるまで高い学習率を維持することで, 大域的探索のために多くのエポックを費やしている. 70 エポックあたりから急激に学習率を低下させることで, 急速に局所的な最適解に近づける. このスケジューリングによって, 我々はミニバッチサイズ 81,920 で 74.9%以上の学習精度を実現した.

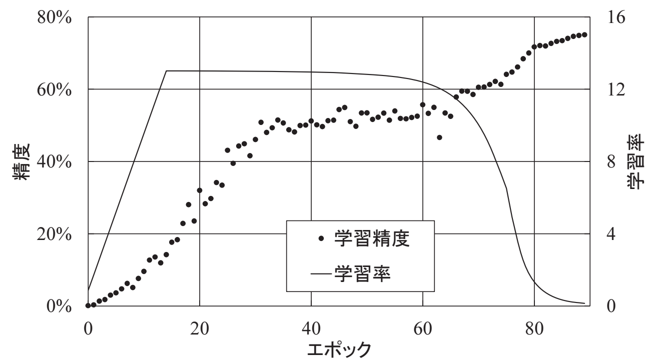

図 5 学習率と学習精度の推移

2,048 基の GPU での分散学習時のミニバッチサイズと学習精度の関係を図 6 に示す. 図 6 の横軸はミニバッチサイズで, 縦軸が学習精度であり, 学習率のスケジューリングには Arg-cotangent を用いた. 学習用の入力データは約 128 万枚であり, ミニバッチサイズが 81,920 まで大きくなるとエポックあたりの学習の反復が 16 回にまで少なくなる. そのため, 90 エポックでも総学習回数は 1,440 回しかなく, SGD による DNN 学習としてはとても少ない. しかし, 図 6 からわかるように, Arc-cotangent スケジューリングを用いることで, 我々はミニバッチサイズ 81,920 までの ResNet-50 の学習で 74.9%以上の学習精度を達成した.

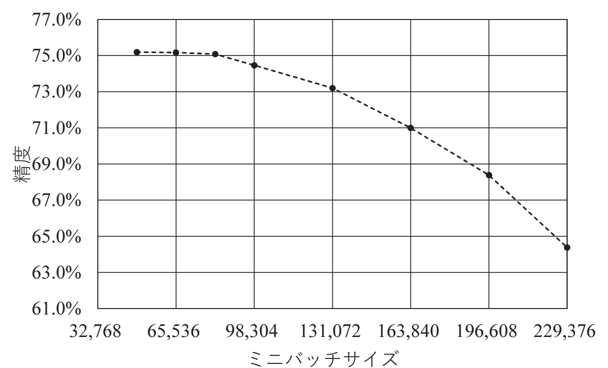

図 6 ミニバッチサイズと精度

5.3 バッチ LARS 計算の評価

本節では, 実装した LARS のバッチ計算の GPU 実装の性能を評価する. LARS なしの SGD と, MXNet に実装されているノルム計算を利用した LARS の naïve 実装ありの SGD, GPU 実装した LARS のバッチ計算ありの SGD の 3 つの更新処理による ResNet-50 の学習速度の違いを図 7 に示す. 本評価では, 通信の影響を除くために 1 基の GPU での平均学習速度で比較した. LARS 計算の naïve 実装では層毎にノルム計算と CPU-GPU 間のデータ転送が発生し, LARS なしの更新処理と比較して学習速度が大きく低下している. 一方で, 我々が実装した LARS の GPU 関数

© 2019 Information Processing Society of Japan

では，1 度の LARS 計算で複数層のノルム計算を GPU で行い，学習率のスケーリングも GPU で計算する．そのため，頻繁な GPU 関数の起動オーバーヘッドとデータ転送のオーバーヘッドを除くことができ，LARS なしの SGD と同程度の学習速度を達成した．

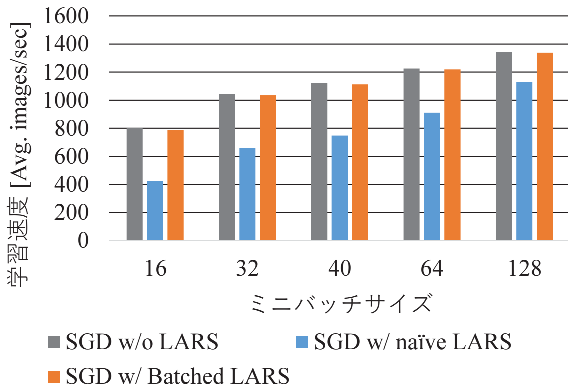

図 7　更新処理の違いによる学習速度の違い (GPU 1 基)

5.4　通信フレームワークの初期化最適化の評価

Horovod スレッドによる NCCL のコミュニケータ初期化コストの隠蔽による性能向上を評価する．2,048 基の GPU での ResNet-50 の分散学習のときの 10 エポック目までのエポック毎の学習時間を図 8 に示す．図 8 は，ミニバッチサイズが 81,920 のときのエポックあたり学習時間である．NCCL の初期化を Allreduce 処理実行時に行う実装を図 8 では，Baseline としている．Baseline の実装では最初の Allreduce 実行時に初期化が行われるため，1 エポック目の学習時間が 2 エポック目以降より非常に長くなることが分かる．一方で，初期化部分を隠蔽することで 1 エポック目に生じるコミュニケータの初期化オーバーヘッドを無くし，1 エポック目の学習時間を大きく短縮した．

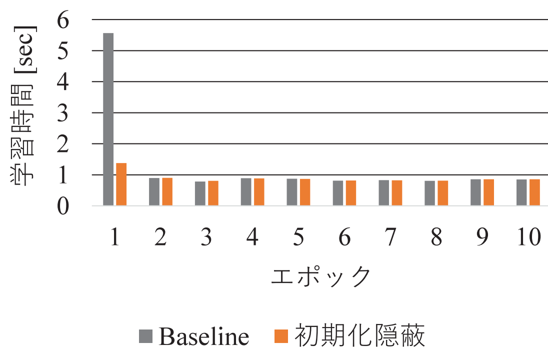

図 8　10 エポック目までのエポックあたり学習時間

5.5　通信最適化の評価

本節では，通信の最適化による学習速度の向上を評価する．図 9 に，全ての層の勾配情報の Allreduce 処理を 1 度に実行する実装の学習サイクル毎の学習速度を示す．図 9 の学習速度は 2048 基の GPU を用いたときの学習速度であり，この実装を Baseline とする．一方で，図 10 は，通信最適化を適用した場合の学習サイクル毎の学習速度を表している．これらの 2 つのグラフより，Allreduce 処理を複数層毎に分割し実行することで，ミニバッチサイズに関係なく学習速度が向上することがわかる．Baseline の実装では，全ての層のバックワード処理完了後に Allreduce 処理が行われるため，オーバーラップ実行による Allreduce 処理の隠蔽ができない．一方で，分割し複数層毎に Allreduce を実行することで，バックワード処理が完了する前に Allreduce 処理をオーバーラップ実行することができるため学習速度が Baseline に対して向上した．また，図 9, 10 の比較より，GPU あたりミニバッチサイズが 16 ととても小さいサイズでも分割による通信コストの隠蔽の効果があることがわかる．

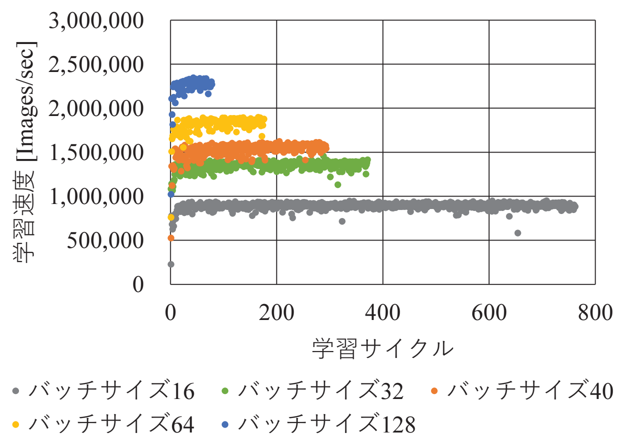

図 9　異なるミニバッチサイズにおける学習速度 （Baseline）

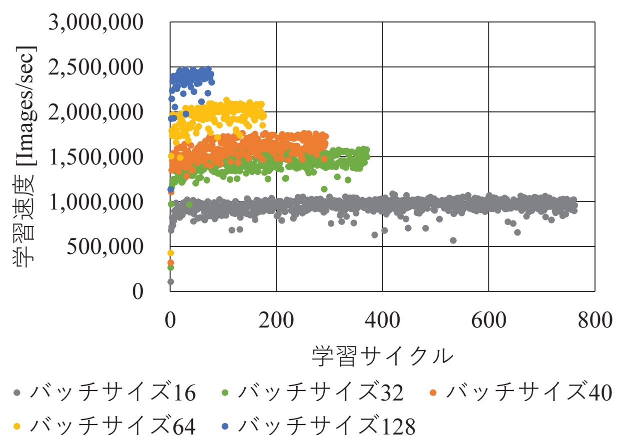

図 10　異なるミニバッチサイズにおける学習速度 (通信最適化適用)

© 2019 Information Processing Society of Japan

Vol.2019-HPC-170 No.6
2019/7/24

表 2　本研究と既存研究の ResNet-50 の学習時間と精度

	ミニバッチサイズ	プロセッサ	深層学習フレームワーク	学習時間	学習精度
He ら [6]	256	Tesla P100 × 8	Caffe	29 hours	75.3%
Goyal ら [2]	8,192	Tesla P100 × 256	Caffe2	1 hour	76.3%
Smith ら [18]	8,192 → 16,384	full TPU Pod	TensorFlow	30 mins	76.1%
秋葉ら [10]	32,768	Tesla P100 × 1,024	Chainer	15 mins	74.9%
Jia ら [11]	65,536	Tesla P40 × 2,048	TensorFlow	6.6 mins	75.8%
Ying ら [4]	65,536	TPU v3 × 1,024	TensorFlow	1.8 mins	75.2%
三上ら [9]	55,296	Tesla V100 × 3,456	NNL	2.0 mins	75.29%
本研究	**81,920**	**Tesla V100 × 2,048**	**MXNet**	**1.2 mins**	**75.08%**

5.6　関連研究との比較

本研究と既存研究の ResNet-50 の学習時間と学習精度を表 2 にまとめる. 我々の計測は, 深層学習のベンチマークである MLPerf v0.5.0 に従い, メモリ確保と学習のための初期化時間を含んでいる. 我々は, ミニバッチサイズ 81,920 ととても巨大なミニバッチで 2,048 基の GPU を用い, 74.7 秒で ResNet-50 の学習を完了した.

我々が知る限り, 既存研究で最も速く ResNet-50 の学習を完了した記録は, Ying らのミニバッチサイズ 65,536 での 1,024 基の TPU による 1.8 分である. また, SGD による学習での最大ミニバッチサイズは Jia ら, Ying らの 65,536 である. 我々の記録は, 既存研究よりも大きなミニバッチサイズ 81,920 での ResNet-50 の学習完了と, 2,048 基の GPU による 1.2 分での学習完了の両方を達成した.

6.　おわりに

我々は, 大規模分散学習のための精度向上技術と学習速度向上技術を開発し, 深層学習フレームワーク MXNet と通信フレームワーク Horovod に適用した. 結果として, 2,048 基の GPU を用いてミニバッチサイズ 81,920 で ResNet-50 の学習を行い, 74.7 秒で学習精度 75.08%を達成した.

謝辞　本研究では主に産業技術総合研究所の AI 橋渡しクラウド (ABCI) を利用した. ABCI 関係者及びサポートチームに, 感謝の意を表する.

参考文献

[1] 山崎雅文，笠置明彦，田原司睦，中平直司：MPI を用いた Deep Learning 処理高速化の提案，技術報告 2016-HPC-155(6) (2016).
[2] Goyal, P., Dollar, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y. and He, K.: Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, *ArXiv:1706.02677* (2017).
[3] You, Y., Gitman, I. and Ginsburg, B.: Large Batch Training Of Convolutional Networks, *arXiv:1708.03888* (2017).
[4] Ying, C., Kumar, S., Chen, D., Wang, T. and Cheng, Y.: Image Classification at Supercomputer Scale, *arXiv:1811.06992v2* (2018).
[5] 小川宏高，松岡　聡，佐藤　仁，高野了成，滝澤真一朗，谷村勇輔，三浦信一，関口智嗣：世界最大規模のオープン AI インフラストラクチャ AI 橋渡しクラウド（ABCI）の概要，技術報告 2018-HPC-165(19) (2018).
[6] He, K., Zhang, X., Ren, S. and Sun, J.: Deep Residual Learning for Image Recognition, *in CVPR* (2016).
[7] Krizhevsky, A., Sutskever, I. and Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks, *in NIPS*, pp. 1097–1105 (2012).
[8] Ioffe, S. and Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, *arXiv:1502.03167v3* (2015).
[9] Mikami, H., Suganuma, H., U-chupala, P., Tanaka, Y. and Kageyama, Y.: Massively Distributed SGD: ImageNet/ResNet-50 Training in a Flash, *arXiv:1811.05233v2* (2019).
[10] Akiba, T., Suzuki, S. and Fukuda, K.: Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes, *arXiv:1711.04325* (2017).
[11] Jia, X., Song, S., He, W., Wang, Y., Rong, H., Zhou, F., Xie, L., Guo, Z., Yang, Y., Yu, L., Chen, T., Hu, G., Shi, S. and Chu, X.: Highly Scalable Deep Learning Training System with Mixed–Precision: Training ImageNet in Four Minutes, *arXiv:1807.11205* (2018).
[12] : MLPerf, `https://mlperf.org/`.
[13] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z.: Rethinking the Inception Architecture for Computer Vision, *arXiv:1512.00567v3* (2015).
[14] Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C. and Zhang, Z.: MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems, *arXiv:1512.01274* (2015).
[15] Sergeev, A. and Balso, M. D.: Horovod: fast and easy distributed deep learning in TensorFlow, *arXiv:1802.05799* (2018).
[16] Jeaugey, S.: Massively Scale Your Deep Learning Training with NCCL 2.4, `https://devblogs.nvidia.com/massively-scale-deep-learning-training-nccl-2-4/` (2019).
[17] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. and Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge, *International Journal of Computer Vision (IJCV)*, Vol. 115, No. 3, pp. 211–252 (2015).
[18] Smith, S. L., Kindermans, P.-J., Ying, C. and Le, Q. V.: Don't Decay the Learning Rate, Increase the Batch Size, *in NIPS* (2017).

© 2019 Information Processing Society of Japan