ディープラーニングのデータ並列学習における少精度浮動小数点数を用いた通信量の削減

大山 洋介[1,a)]　野村 哲弘[1]　佐藤 育郎[2]　松岡 聡[1]

概要：Deep Neural Network を用いた学習手法であるディープラーニングは他の機械学習手法と比較して高い認識精度を発揮することから近年非常に重要視されている．一方でディープラーニングはネットワークの計算量や学習に使用するデータ量が膨大であることから GPU クラスタを用いた場合でも学習に非常に長い時間を要する．また，特にパラメータ数が多いネットワークを一定のミニバッチサイズで学習する場合は勾配の GPU 間・ノード間通信がスケーラビリティのボトルネックとなり，現存する GPU スパコンで利用可能な並列数よりもはるかに小さな規模でしか学習できないことが指摘されている．本論文では単精度よりも更に bit 数の少ない浮動小数点数型を用いた通信量の削減手法を提案する．提案手法では通信するデータを半精度浮動小数点数の上位 8bit により表現し，レイヤーごとに動的に表現範囲を調整することにより高速かつ単精度と比較して学習後の認識精度を大きく損なわない通信を実現する．提案手法は TSUBAME-KFC/DL の 2 ノード (16 GPU) を用いた CaffeNet と GoogLeNet の学習において，既存の単精度浮動小数点型を用いる場合と比較して認識精度を損なわずにそれぞれ 2.71 倍，2.19 倍の高速化を達成した．

1. 背景

ディープラーニング (以下 DL) とは Deep neural network(以下 DNN) と呼ばれる計算モデルを用いた機械学習手法であり，他の古典的な手法と比較して高い認識精度を発揮することから近年非常に重要視されている．DL は DNN の複雑さやデータセットの量・質を増大させることで認識精度を向上させることができると考えられている [1–3] ことから，これらの規模は年々増加する傾向にある．DL ではデータセットによって定義されるコスト関数を用いて DNN のパラメータの最適化 (学習) を行うが，最急降下法のようなコスト関数の値や勾配を直接用いる最適化手法は巨大なデータセット全体を反復的に処理する必要があるため現実的ではなく，代わりに 1 反復でデータセットのごく一部 (ミニバッチ) のみを必要とする Stochastic Gradient Descent (SGD) が広く用いられる [2–11]．また，DL は学習が収束するまでの計算量が膨大であることから GPU 等を複数用いた学習が行われることが多く [2, 8–11]，この中でも別々のサンプルについての勾配を並列に計算しパラメータを同期するデータ並列学習は実装が容易であり多くの実装が存在する [2, 8–11]．一方で特にパラメータ数が多いネットワークを一定のミニバッチサイズで学習する場合は勾配の GPU 間・ノード間通信がスケーラビリティのボトルネックとなり，現存する GPU スパコンで利用可能な並列数よりもはるかに小さな規模でしか学習できないことが指摘されている [8, 10–14]．また，通信時間を隠蔽するためにミニバッチサイズが大きく設定されることがある [2, 10] が，ミニバッチサイズが増大すると学習時間および認識精度が悪化することが指摘されている [10, 11, 15–17]．

本論文では単精度よりも更に bit 数の少ない浮動小数点数型を用いた通信量の削減手法を提案する．提案手法では通信するデータを半精度浮動小数点数の上位 8 bit により表現し，レイヤーごとに動的に表現範囲を調整する．また，勾配を直接通信するのではなく勾配とパラメータの比を用いることで通信データの分散をより小さくし，高速かつ単精度と比較して学習後の認識精度を大きく損なわない通信を実現する．提案手法は TSUBAME-KFC/DL の 2 ノード (16 GPU) を用いた CaffeNet と GoogLeNet の学習において，既存の単精度浮動小数点型を用いる場合と比較して認識精度を損なわずにそれぞれ 2.71 倍，2.19 倍の高速化を達成した．また，CaffeNet を学習する場合の GPU ごとのバッチサイズを一定とした weak scale の評価において，192 GPU で 168 倍 (単精度を用いる場合の 1.47 倍) のスケーラビリティを達成した．

[1] 東京工業大学
[2] デンソーアイティーラボラトリ
[a)] oyama.y.aa@m.titech.ac.jp

ⓒ 2017 Information Processing Society of Japan

2. 関連研究

2.1 通信データの量子化に関する研究

[12] では DNN のデータ並列学習において GPU 間で通信される勾配を 1 bit で表現する手法が提案されている．この手法では勾配の値はその符号によって 0 または 1 で表現され，これに加えてパラメータ行列の列ごとに正負の勾配それぞれの平均値を通信することで 1 bit での量子化・復号化を実現する．また，同符号の値すべての平均値を代表値とすると量子化誤差が大きくなることから，量子化の際に GPU ごとに量子化誤差を計算しておき，次回の反復の量子化前の勾配に加算することで誤差が累積することを防ぐ．この手法は 40 個の NVIDIA Tesla K20Xm GPU を用いた 160×10^6 パラメータの DNN の学習において，わずかな認識精度の低下を含むものの単一 GPU の 10 倍の学習速度を達成した．

[13] では通信される勾配をある閾値を超えた場合に制限し，疎ベクトルとして通信する手法が提案されている．勾配の値はインデックス (31 bit) および符号 (1 bit) の計 32 bit で表現され，閾値が代表値として使用される．この手法でも [12] と同様に量子化誤差を次回の反復に繰り越す手法が用いられている．この手法は 80 個の NVIDIA GRID K520 GPU を用いた 14.6×10^6 パラメータの音声認識を行う DNN の学習において，わずかな認識精度の低下を含むものの単一 GPU の 54 倍の学習速度を達成した．

[14] では [13] で閾値をユーザが指定する必要があるという問題に対し，通信する勾配の割合をユーザが指定して閾値を勾配値のランダムサンプリングにより決定する手法が提案されている．また，復号時には閾値ではなく閾値を超えた勾配の平均値が使用される．128 ノード (ノードあたり 2 MPI プロセス) を用いた 16384×16384 サイズの行列の all-reduce について，前述の 1 bit 通信・疎ベクトル通信はデフォルトの all-reduce よりも低速だった一方で，この手法は all-reduce の 1.76 倍高速だった．

これらの量子化通信はノード間で通信されるメッセージサイズが一定ではなく，プロセス間の通信スケジュールを考慮した計算が必要とされることから，メッセージサイズが削減される場合でもナイーブな all-reduce より低速である場合がある [14]．一方で本論文の提案手法では勾配が単純に少精度で表現されるだけであり，CPU 上での加算処理も SIMD 命令を用いて高速に実行されることから，プロセス数やメッセージサイズに関わらず通信時間を削減することができる．

2.2 少 bit 数の計算型を用いる学習に関する研究

[18] では固定小数点型を用いた DNN の学習手法が提案されている．この手法では，実数を表現する型として整数部 IL bit, 小数部 FL bit の計 IL+FL bit の固定小数点型を用いる．また，積和計算等の後に行う丸め手法として，最も近い値に丸める round-to-nearest と近い大小 2 つの値との距離の比によって確率的に丸めを行う stochastic-rounding が提案されている．この手法は画像データセットである MNIST [19] や CIFAR10 [20] を用いた全結合 DNN や CNN の学習において 16 bit の型と stochastic-rounding を用いることで認識精度の低下なく学習が行えることができた．

この手法では計算の高速化を目的として学習全体の計算精度を下げているが，本論文の提案手法はスケーラビリティを上げるために通信部分の計算精度のみを下げているという点が異なる．また，この手法で提案されている stochastic-rounding や固定小数点型による計算は非決定的であり近年の CPU・GPU では一般的でない一方で，本論文の提案手法は決定的かつ近年の CPU・GPU で実装されている計算方法のみを用いる．

2.3 データ並列学習に関する研究

[10] では MPI を用いたデータ並列による DNN 学習の高速化手法が提案されている．この手法では勾配の計算と MPI 通信の並列処理や通信の細分化・追い抜き等を用いて通信時間を隠蔽しており，AlexNet [4] を 256 個の NVIDIA K20X を用いてミニバッチサイズ 65536 で学習した場合に 217 倍の処理速度の高速化を達成した．

この手法では weak scale(GPU ごとのバッチサイズを一定とした学習) の評価のみが行われているが，ミニバッチサイズが大きいと学習後の認識精度が悪化することが指摘されている．本論文の提案手法は weak scale での評価も行う一方で，strong scale(学習全体のミニバッチサイズを一定とした学習) に関しても認識精度の悪化なしに高速化が達成されている．

3. 提案手法

3.1 8bit 浮動小数点数型 (fp8)

本論文で提案する 8bit の浮動小数点数型 (以下 fp8) の概要を表 **1** に示す．fp8 は符号部 1 bit, 指数部 5 bit, 仮数部 2 bit で全体で 8 bit の長さを持つ型である．fp8 の指数部のオフセットは半精度浮動小数点数型 (以下 half) [21] と同じ 15 とし，最大値は $(1.11)_2 \times 2^{30-15} = 1.75 \times 2^{15} = 57344$, 最小の最小値は $(0.01)_2 \times 2^{-15+1} = 2^{-16} \sim 1.53 \times 10^{-5}$ である．NaN の場合を除いて fp8 の値 x は half の上位 8 bit を x, 下位 8 bit を 0 としたときの値と等しく，これにより half をハードウェアまたはソフトウェアレベルでサポートしている計算機上で容易に扱うことができる．

CNN の一種である CaffeNet を用いた場合の GPU ごとに計算される勾配の分布を図 **1** に示す．勾配の大部分は $10^{-10} \sim 10^{0}$ 程度に分布しており，各テンソルの 10% ∼ 100% 分位数は fp8 で表現できる最大値と最小値の比である $10^{9.5}$ 内の範囲に分布していた．よって fp8 を用

表 1　浮動小数点型の bit 長と表現範囲

	符号 [bit]	指数 [bit]	仮数 [bit]	合計 [bit]	最大値	正の最小値
fp8	1	5	2	8	5.73×10^{4}	1.53×10^{-5}
half	1	5	10	16	6.55×10^{4}	5.96×10^{-8}
float	1	8	23	32	3.40×10^{38}	1.40×10^{-45}
double	1	11	52	64	1.80×10^{308}	4.94×10^{-324}

いた場合でも勾配の大部分は飽和することはなく，3.2.1 節で述べる手法を用いることでさらに飽和しにくくなると考えられる．

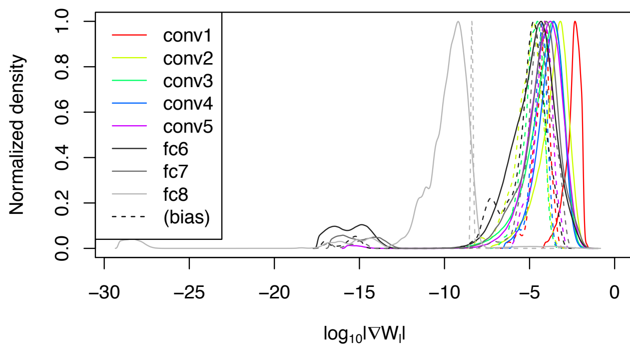


図 1　CaffeNet の学習中におけるパラメータテンソルごとの勾配の分布．学習条件は図 9 と同じ設定を用いた．分布はテンソルごとに 1%〜99%分位数内の非ゼロの値を用いてガウス関数によるカーネル密度推定により推定し，y 軸は推定された確率密度の最大値で正規化した．

3.2　提案する GPU 間・ノード間通信手法

提案する GPU 間・ノード間通信手法を Algorithm 1 に示す．なお，$\circ$ はベクトル同士の要素ごとの積，$\circ^{-1}$ は要素ごとの商を表し，アルゴリズムでは以下の定数 (明記しない限り下記の値を用いる) と変数を使用する．また，ミニバッチサイズ B はプロセス数 P の倍数であると仮定する．パラメータテンソルは DNN のパラメータを構成するテンソルであり，本研究では同じレイヤーの重みとバイアスを別のテンソルとして考える．

- 定数
  - B: ミニバッチサイズ
  - P: プロセス数 (=総 GPU 数)
  - P': ノード内プロセス数 (=ノード内 GPU 数)
  - $Q = 0.95$: 表現範囲を調整するための値
  - $N = 100$: Q 分位数を計算するステップ周期
  - $R = 1024$: 分位数を計算するための最大ランダムサンプリング数
  - $\varepsilon = 10^{-5}$: 比計算の際のゼロ除算を防ぐための補正値
- プロセス変数
  - p: プロセス ID
  - p': ノード内プロセス ID

Algorithm 1 提案する通信手法

1: // ノード内・同 GPU ID 同士のコミュニケータを作成
2: $C_{Node} = Comm_split(p\%P', p/P')$
3: $C_{GPUID} = Comm_split(p/P', p\%P')$
4: $i = 0$
5: **repeat**
6: 　// 担当する B/p サンプルについての勾配を計算
7: 　$\nabla W \leftarrow Backprop(W, B/p)$
8: 　**for all** $l \in$ (パラメータテンソル) **do**
9: 　　$D_l = \nabla W_l \circ^{-1} (|W_l| + \varepsilon)$
10: 　　**if** $i\%N = 0$ **then**
11: 　　　// R 要素のランダムサンプリングにより $|D_l|$ の Q 分位数を計算
12: 　　　$q_l \leftarrow quantile_R(|D_l|, Q)$
13: 　　**end if**
14: 　　// fp8 に変換しホストメモリに転送
15: 　　$H_l \leftarrow D2H(float_to_fp8(D_l \times (57344/q_l)/P'))$
16: 　　// ノード内加算
17: 　　// $H_l^{(j)}$ は H_l を P' 等分したときの j 個目の配列
18: 　　$H_l \leftarrow Alltoall(H_l, C_{Node})$
19: 　　$H'_l \leftarrow \sum_{j=0}^{P'} H_l^{(j)}/(P/P')$
20: 　　// ノード間加算
21: 　　$H'_l \leftarrow Allreduce_sum(H'_l, C_{GPUID})$
22: 　　// 加算結果を共有
23: 　　$H_l \leftarrow Allgather(H'_l, C_{Node})$
24: 　　// デバイスメモリに転送し float に変換
25: 　　$D_l \leftarrow fp8_to_float(H2D(H_l)) \times (q_l/57344)$
26: 　　$\nabla W_l \leftarrow D_l \circ (|W_l| + \varepsilon)$
27: 　　// パラメータを更新
28: 　　$W_l = SGD_update(W_l, \nabla W_l)$
29: 　**end for**
30: 　$i \leftarrow i + 1$
31: **until** プロセスが終了する

提案手法では GPU での計算には float を，ホスト-デバイス間・MPI 通信や CPU での計算では fp8 を用いる．これにより，デバイスメモリと比較して帯域幅の小さい PCI Express およびインターコネクト上では常に fp8 が使用され，float のみで通信する場合よりも通信時間の短縮が期待される．

float と fp8 の型の変換は CUDA 7.5 [22] よりサポートされた half の上位 8 bit を用いて GPU 上で行う．この際，変換前の値に無限大および NaN は出現しないと仮定する．all-reduce 中で fp8 の数同士の加算により無限大が出現する場合の処理については 3.2.4 節で述べる．

3.2.1　通信データの相対化

提案手法では直接勾配を通信するのではなく，勾配 ∇W とパラメータ W の要素ごとの比 $D_l = \nabla W_l \circ^{-1} (|W_l| + \varepsilon)$ を通信する．DNN の重み行列は列方向に似た値が分布することが指摘されており [23]，勾配を量子化して通信する手法でも列ごとに代表値をとる手法がとられている [12, 14]．よって提案手法では勾配とパラメータの大きさに相関があると仮定し，この比を fp8 で通信することで fp8 による計

ⓒ 2017 Information Processing Society of Japan

算で計算誤差が大きくなることを防ぐ．各プロセスは同じステップで計算誤差を除き同一の W をもつため，比の計算は他のプロセスとの通信なしに行うことができる．比を用いる場合と勾配を直接用いる場合の学習への影響は 4.5.1 節で述べる．

3.2.2 fp8 の表現範囲の動的な調整

fp8 は float よりも表現できる値の範囲が狭いため，通信される大部分の値が fp8 の表現範囲に入るように Q 分位数 q_l を用いて調整する．q_l は一定のステップ周期 N ごとにランダムサンプリングした $|D_l|$ の要素を用いて計算する (Algorithm1 の 12 行目)．ランダムサンプリングした結果はプロセスごとに異なることから，プロセスごとに計算した値を最大値 all-reduce することにより全プロセスで同一の値を得る．

また，提案手法では通信データをノード内・ノード間の 2 段階で加算する．これは fp8 の表現範囲が float と比較して狭いため，1 回の all-reduce ですべてのデータを加算する場合にオーバーフロー・アンダーフローが発生し認識精度が悪化する可能性があるためである (4.5.2 節)．ノード内加算の前には値 q_l をノード内 GPU 数分加算した場合でもオーバーフローしないように通信データに $(57344/q_l)/P'$ を乗算し，次のノード間加算の前には同様の理由から P/P' を除算する．これらの係数は加算後に GPU 上で $q_l/57344$ を乗算することでキャンセルされ，通信データの平均が得られる．

3.2.3 非同期通信

DNN の誤差逆伝播法では一度出力レイヤーが計算されたのちは出力から入力に向かって 1 レイヤーずつ勾配が計算されるため，あるレイヤーの勾配はそのひとつ前のレイヤーの勾配計算中に通信することができる．提案手法では後述する Caffe [24] の実装を改変し，ホスト-デバイス間転送と MPI 通信部分 (Algorithm 1 の 14〜25 行目) を元のスレッド (以下 GPU スレッド) とは別のスレッド (以下通信スレッド) に以下のように担当させることで非同期通信を実装した．

(1) GPU スレッドはあるレイヤーの勾配を計算したあと，レイヤー毎に定義された cudaStream で非同期にホストメモリに転送し，すぐにその前のレイヤーの勾配を計算する．
(2) 通信スレッドは通信データがホストメモリに転送されたことを確認するとそのレイヤーに関する MPI 通信を行いミニバッチ全体の勾配を計算し，同じ cudaStream でデバイスメモリに転送する．
(3) GPU スレッド全てのレイヤーの誤差逆伝播が終わったのちに転送用の cudaStream を待機し，レイヤーごとにパラメータを更新する．

このアルゴリズムでは，勾配計算・ホスト-デバイス間転送・MPI 通信のすべてがオーバーラップする余地がある．

3.2.4 Intel AVX を用いたベクトル加算演算

現在の MPI では 8 bit 浮動小数点数型による演算は定義されていない [25] ため，提案手法では fp8 の加算を行う MPI データ型と演算を独自に定義した．

MPI データ型は fp8 の数を 16 個パックした 128 bit のベクトルデータ型とし，通信する fp8 の要素数が $16 \times P'$ の倍数でない場合はそれより大きい最小の倍数にパディングする．

fp8 の加算には Intel AVX [26] を用いた SIMD 演算を実装した (Algorithm 2)．fp8 の加算演算では fp8 の値を half から float への変換命令である VCVTPH2PS 命令により float に変換し，float として加算後に再び逆の VCVTPS2PH 命令により fp8 に変換する．また，fp8 では表現範囲が小さいために VCVTPS2PH 命令による変換結果が正負の無限大となる場合があり得る [26] が，fp8 の指数部の全 bit が 1 である場合に符号なし整数として 1 を減算することで正負の最大値に変換し，無限大の値とならないようにする．この処理の行わない場合，学習が正常に進行しない場合があることを確認している．

図 2 fp8 ベクトル加算の擬似コード．可読性のためにベクトルの要素ごとの加算やシフト等の低レベルな処理は算術記号で表した．

```
// Input: a8, b8 (fp8*16 要素, 128 bit)
// Output: sum = a8 + b8 (fp8*16 要素, 128 bit)
ODD = 0xFF00FF00FF00FF00FF00FF00FF00FF00; // 奇数バイト
EXP = 0x7C7C7C7C7C7C7C7C7C7C7C7C7C7C7C7C; // 指数部マスク
ONE = 0x01010101010101010101010101010101; // "1"ベクトル
// 0, 2, 4, ..., 14 番目の要素同士を加算
a_float_e = _mm256_cvtph_ps((a8 << 8) & ODD);
b_float_e = _mm256_cvtph_ps((b8 << 8) & ODD);
sum_half_e = _mm256_cvtps_ph(a_float_e + b_float_e, 0) >> 8;
// 1, 3, 5, ..., 15 番目の要素同士を加算
a_float_o = _mm256_cvtph_ps(a8 & ODD);
b_float_o = _mm256_cvtph_ps(b8 & ODD);
sum_half_o = _mm256_cvtps_ph(a_float_o + b_float_o, 0);
// 奇数目・偶数目の要素を結合
sum = _mm_blendv_epi8(sum_half_e, sum_half_o, ODD);
// 無限大を同符号の最大値に変換
sum -= (_mm_cmpeq_epi8(sum & EXP, EXP) & ONE);
```

3.3 提案手法を用いた Caffe の拡張

本論文では Caffe を MPI を用いてデータ並列学習するよう以下のように改変し，さらに提案手法を適用したバージョンを実装した．

- SGD ソルバで Algorithm 1 に従った更新を行う．
- データサンプル読み込み時に自プロセスが担当するサンプルのみをシークする．
- 全プロセスの accuracy の平均を学習全体での代表値として出力する．

© 2017 Information Processing Society of Japan

4. 評価

4.1 評価環境

実行環境を表 **2** に示す．なお，FLOP/s は各設定下の単精度浮動小数点数の理論計算性能を表す．また，Tesla K80 は 2 つの GK210 チップを持ち個別の 2 GPU として動作するため，ノード内 GPU 数は 8 である．

表 **2** TSUBAME-KFC/DL のスペック・実行環境

ノード数	42
CPU	Intel Xeon E5-2620 v2 × 2 2.1 GHz, 6 コア 64GB DDR3 メモリ
GPU	NVIDIA Tesla K80 × 4 8.74 TFLOP/s 24 GB GDDR5 メモリ ECC 無効, Auto boost 有効
SSD	Intel SSDSC2BB480G4 × 2 480GB SATA3
インターコネクト	4X FDR InfiniBand (7 GB/s)
OS	CentOS 7.3
コンパイラ	GCC 4.8.5 (`-O2 -march=core-avx-i` `-mtune=core-avx-i` オプション)
CUDA	CUDA 8.0
MPI	OpenMPI 2.0.1
Caffe	1.0.0-rc3

4.2 評価するデータセット

評価に使用するデータセットとしては ILSVRC2012 データセット [27] のうちの 16 クラスのみを取り出した部分集合を用いた．学習用画像は 20571 枚，テスト用画像は 800 枚 (50 枚 ×16 クラス) である．データセットは共有ファイルシステムでなく全ノードの SSD に配置することで読み出しの高速化を図った．

4.2.1 評価するモデル・学習条件

モデルは AlexNet [4] をベースとしたモデルである CaffeNet と GoogLeNet [5] を用いた (表 **3**)．また，本研究で使用したデータセットは 16 クラスである一方でこれらのモデルの出力は 1000 クラスであるが，先行研究との比較を容易にするためにモデルのクラス数は変更せずに用いた．

モデルの詳細やハイパーパラメータは明示しない限り Caffe に付属して配布されている設定ファイルにより決定した．ミニバッチサイズは CaffeNet では 256，GoogLeNet では 32 とした．また，学習終了までの反復回数や学習係数が減少する反復数はデータセットのクラス数の比によって決定した．特に GoogLeNet に関しては，学習係数を $0.01 \times \{1 - (\text{反復回数})/38400\}^{0.5}$ として 60 エポック学習する設定 (以下 quick) と学習係数を 0.01 から 5120 反復ごとに 0.96 倍して 250 エポック学習する設定 (以下 slow) の二つを用いた．

表 **3** 評価に使用したモデル

	CaffeNet	GoogLeNet	
		(quick)	(slow)
パラメータ数 [10^6]	61.0	13.4	
パラメータテンソル数	16	128	
畳み込みレイヤー数	5	59	
全結合レイヤー数	3	5	
ミニバッチサイズ	256	32	
初期学習係数	0.01	0.01	
エポック数	90	60	250
反復回数	7200	38400	160000

同じネットワークを複数の異なる条件で学習する場合，最初の 100 反復を float を用いて事前に学習し，それ以降の反復を異なる条件で学習することでパラメータの初期化に使用される乱数の違いによる影響を除いた．

4.3 通信性能の評価

TSUBAME-KFC/DL 上で fp8 を用いた all-reduce の実行時間を評価した．評価は各評価点について `MPI_Allreduce` の実行を連続して 10 回行い，それらの実行時間の中央値を代表値とした．

16 ノード (1 ノード 1MPI プロセス) で要素数を $1, 2, 4, \cdots, 256 \times 10^6$ とした時の実行時間を図 **3** に示す．なお，“float(MPI_SUM)” は MPI で定義されている加算演算を用いた場合であり，“float,half,fp8(SIMD)” はそれぞれの型についての加算演算を独自に定義した場合である．“float,half,fp8(SIMD)” ではそれぞれ 8,8,16 要素 (256 bit, 128 bit, 128 bit) をパックした型を用いている．結果より，256×10^6 要素の all-reduce を行う場合 fp8(SIMD) は float(MPI_SUM) の 3.21 倍，half(SIMD) の 1.56 倍のスピードアップとなった．

$1, 2, 4, \cdots, 32$ ノードで要素数を 256×10^6 とした時の実行時間を図 **4** に示す．実行時間がノードが増加するにつれて一定に近づく傾向にあるが，これはメッセージサイズが大きいために時間計算量が $O(n\frac{p-1}{p})$ (n はメッセージサイズ，p はプロセス数) である pairwise exchange reduce-scatter と ring-based allgather によるアルゴリズム [14, 28] が使用されるためであると考えられる．すべてのノード数について，fp8(SIMD) は float(MPI_SUM) の 3.19 倍以上のスピードアップだった．

4.4 ステップ時間の内訳

4.4.1 CaffeNet

float および fp8 を all-reduce に用いて CaffeNet の学習を行った場合の 1 反復の実行時間の内訳を図 **5** に示す．実行は 100 反復行い，全プロセスの 1 反復の実行時間の

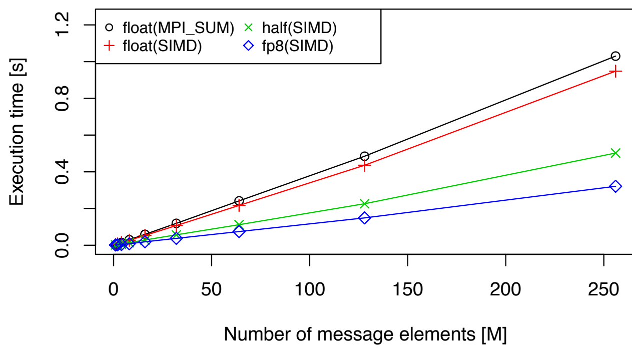


図 3 TSUBAME-KFC/DL の 16 ノードでの all-reduce の実行時間

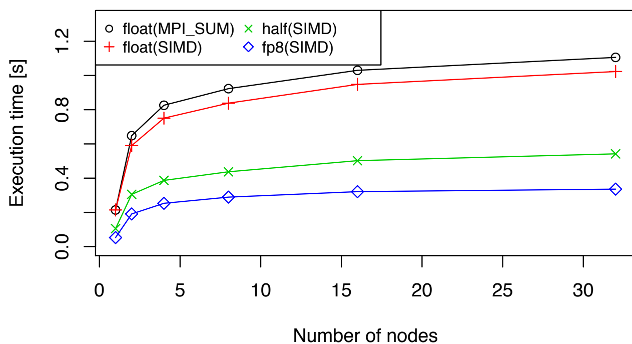


図 4 TSUBAME-KFC/DL の $1, 2, 4, \cdots, 32$ ノードでの 256×10^6 要素の all-reduce の実行時間

中央値を代表値として表示した．実行時間は勾配計算時間 (“ForwardBackward”)，ホストメモリへの転送時間の合計 (“D2H”)，通信時間 (“Communication”)，デバイスメモリへの転送時間の合計 (“H2D”) を計測し，反復全体の実行時間とこれらの時間の差を “Other” として表示した．“Other” にはパラメータを更新するカーネルの実行時間等が含まれる．また，fp8 では 1 回の分位数の計算時間を N で割った時間を “Quantile” として表示した．

図 5 より，float を用いる場合は 2 ノード (非同期の場合は 1 ノード)，fp8 を用いる場合は 2 ノード (非同期の場合は 4 ノード) が最も高速であり，2 ノードで fp8 を用いる場合は float を用いる場合 (ともに非同期) の 2.67 倍のスピードアップだった．また，float を用いる場合は 16 ノードの時に ForwardBackward が全体に占める割合が 3.91%である一方で，fp8 では 10.6%であり，計算と通信を非同期化した場合により効率的に隠蔽できると考えられる．

4.4.2 GoogLeNet

同様に GoogLeNet の学習を行った場合の 1 反復の実行時間の内訳を図 **6** に示す．

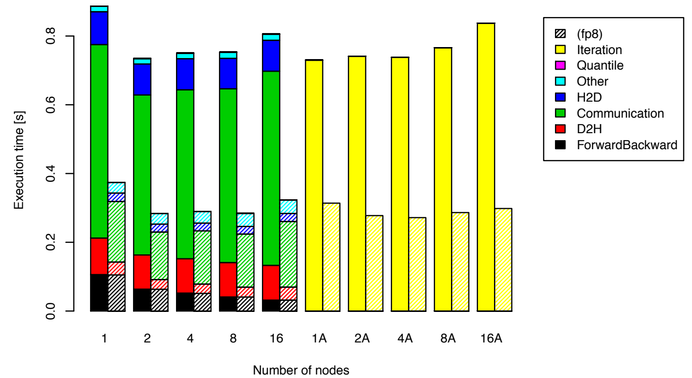


図 5 TSUBAME-KFC/DL で CaffeNet の学習を行った場合の 1 反復の内訳．塗り潰しは float を用いた場合，斜線部は fp8 を用いた場合の実行時間を表す．ノード数末尾の “A” は通信を非同期にした実装を表し，反復全体の時間のみを “Iteration” として表示した．

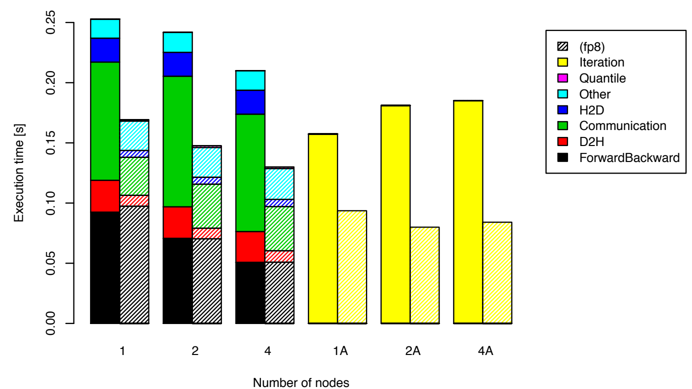


図 6 TSUBAME-KFC/DL で GoogLeNet の学習を行った場合の 1 反復の内訳

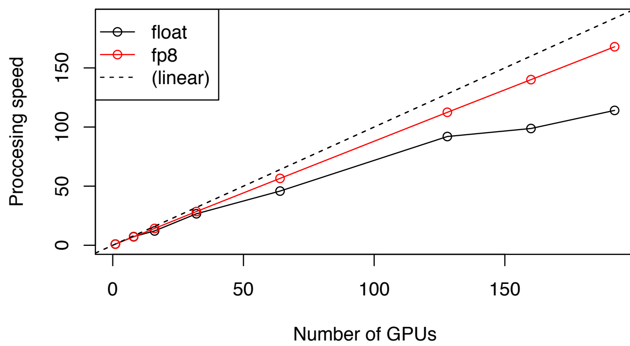
図 6 より，float を用いる場合は 4 ノード (非同期では 2 ノード)，fp8 を用いる場合は 4 ノード (非同期では 2 ノード) が最も高速であり，2 ノードで fp8 を用いる場合は float を用いる場合 (ともに非同期) の 2.26 倍のスピードアップだった．

4.4.3 weak scale

GPU ごとのバッチサイズを 256 (CaffeNet), 32(GoogLeNet) で固定した場合の float と fp8 の処理速度を図 **7** に示す．処理速度は 1GPU と比較した場合のサンプルあたりの実行時間で表した．

CaffeNet を学習する場合，float の処理速度は 192 GPU(8 GPU×24 ノード) で 114 倍となった一方で，fp8 は 168 倍 (float と比べて 1.47 倍) だった．これは float を用いる場合 CaffeNet のパラメータ数が多く通信時間が計算で隠蔽できない一方で，fp8 では通信時間が float よりも短いために 192 GPU を用いた場合でも十分に隠蔽できるためである

ⓒ 2017 Information Processing Society of Japan



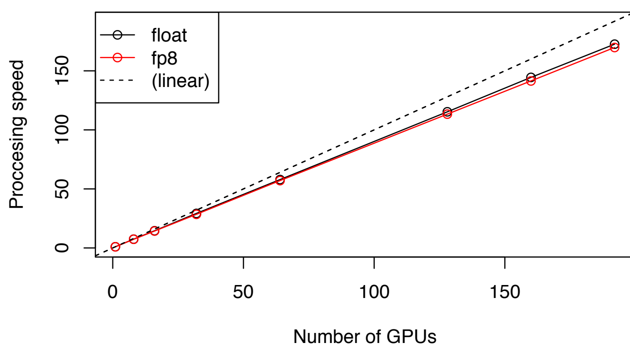


図 7　TSUBAME-KFC/DL で CaffeNet(上)，GoogLeNet(下) の学習を行った場合の weak scale．y 軸は 1 GPU で MPI 通信を行わずに学習を行う場合と比較したデータサンプルの処理速度を表す．

と考えられる．

一方，GoogLeNet を学習する場合，float の処理速度は 192 GPU で 173 倍となった一方で，fp8 は 170 倍だった．これは GoogLeNet は CaffeNet よりもパラメータ数が少ないため float でも通信時間を十分に隠蔽でき，float と fp8 の変換等を行うオーバーヘッドの分だけ fp8 の処理速度が低下するためであると考えられる．

4.5　提案手法の学習曲線

4.5.1　通信データによる認識精度の変化

float と fp8 を用いて 2 ノードで CaffeNet の学習を 90 エポック行った際の top-1 accuracy を図 **8** に示す．"fp8($\varepsilon = 10^{-3}, 10^{-4}, 10^{-5}$)" はそれぞれ ε を指定の値にした場合の結果であり，"fp8($D_l = \nabla W_l$)" は通信データとして勾配を直接用いる場合 (Algorithm 1 の 9 行目で $D_l = \nabla W_l$ とする場合) の結果である．top-1 accuracy は 100 反復ごとに計算した．

float の最良 top-1 accuracy は 73.25%，fp8($\varepsilon = 10^{-5}$) は 73.75%であり，fp8($\varepsilon = 10^{-5}$) の 90 エポックの学習速度は float の 2.71 倍のスピードアップだったことから，精度を

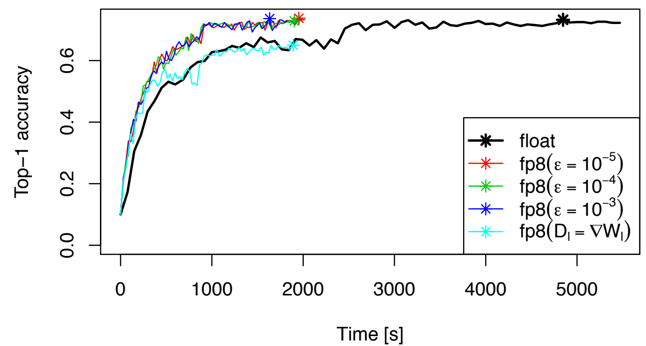


図 8　TSUBAME-KFC/DL で CaffeNet の学習を行った場合の実行時間に対する top-1 accuracy．星印は最良の top-1 accuracy を表す．

損なわずに学習を高速化できたと考えられる．また，勾配とパラメータの比を通信する場合は $\varepsilon = 10^{-3}, 10^{-4}, 10^{-5}$ のどの場合でも最終的な top-1 accuracy が float と同様となったものの，通信データとして勾配を直接用いる場合は最良 top-1 accuracy が 8.25%低下した．これは 3.2.1 節で述べたように勾配はパラメータとの比よりも分散が大きくアンダーフローが発生しやすいためであると考えられる．

float を用いた場合の 3000 反復目の 1 つの GPU で計算された conv2 の勾配 (16 サンプルの平均)，conv2 の重みとこれらの比 ($\varepsilon = 10^{-5}$) のヒストグラムを図 **9** に，可視化した行列を図 **10** に示す．

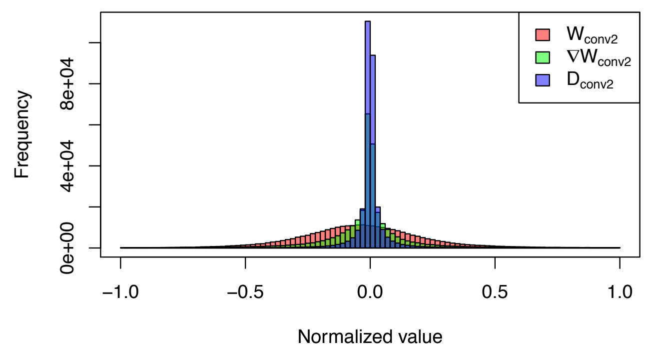


図 9　CaffeNet の 3000 反復目の conv2 の重み W_{conv2}，勾配 ∇W_{conv2}，それらの比 $D_{conv2} = \nabla W_{conv2}/(|W_{conv2}| + \varepsilon)$ のヒストグラム．それぞれの項目について，絶対値の 0.99-分位数以下の要素のみをその分位数との比で正規化して表示した．

図 9 より，D_{conv2} は ∇W_{conv2} よりも分散が小さい．また，図 10 より重みや勾配には行・列ごとに局所性があることが観察される．提案手法では勾配中に現れるパラメータの行・列ごとの局所性を除算によってキャンセルするこ

ⓒ 2017 Information Processing Society of Japan

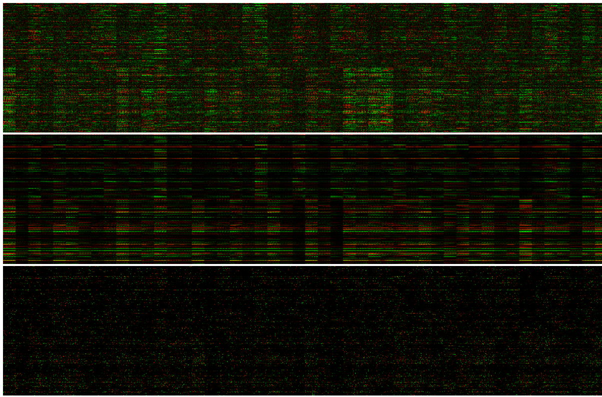


図 10　CaffeNet の 3000 反復目の conv2 の重み W_{conv2}(上)，勾配 ∇W_{conv2}(中)，それらの比 D_{conv2}(下) の行列の可視化．列方向は入力ニューロン $(5 \times 5 \times 48)$，行方向はフィルタ (128×2) を表す．赤色は正の値，緑色は負の値を表す．それぞれの項目について，絶対値の 0.99-分位数以上または以下の要素をその分位数に丸めて表示した．

とで分散が小さくなり，fp8 でも精度を損なわずに通信できたと考えられる．

4.5.2　通信手法による認識精度の変化

同様の条件で CaffeNet の学習を行う際，提案手法のノード間・ノード内の階層的な通信を 1 回の all-reduce で行った場合の top-1 accuracy を図 **11** に示す．

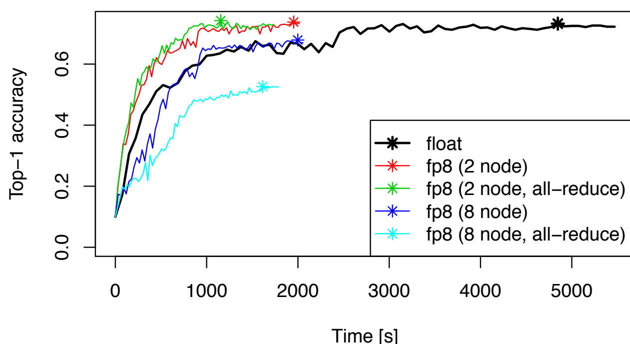


図 11　TSUBAME-KFC/DL で CaffeNet の学習を 2 種類の通信手法で行った場合の実行時間に対する top-1 accuracy．“all-reduce” は加算を 1 回の all-reduce で行った場合の結果であり，それ以外の fp8 は階層的に加算を行った場合の結果である

fp8 を用いて 2 ノードで学習した場合はどちらの通信手法でも float と同等の top-1 accuracy が得られたものの，8 ノードで学習した場合は all-reduce を用いて通信する場合に大きく accuracy が低下した．これは 3.2.2 節で述べたように all-reduce を用いる場合はオーバーフロー・アンダーフローが発生しやすくなるためであると考えられる．

4.5.3　学習中の分位数の変化

4.5.1 節の fp8($\varepsilon = 10^{-5}$) でのパラメータテンソルごとの Q 分位数の変化を図 **12** に示す．

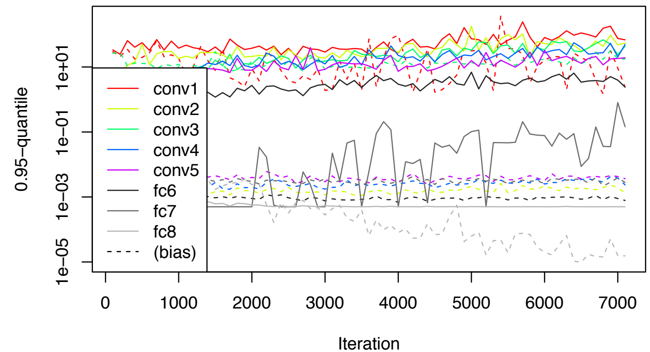


図 12　TSUBAME-KFC/DL で CaffeNet の学習を行った場合のパラメータテンソルごとの Q 分位数の変化．conv は畳み込みレイヤー，fc は全結合レイヤーの重みを表し，点線は対応する色のレイヤーのバイアスを表す．

図 12 より，テンソルごとに分位数が最大で 10^6 程度異なった．fp8 の正の最大・最小値の比は $10^{9.5}$ 程度であるから，テンソルごとに異なる係数を用いて表現範囲を調整しない場合には値が飽和しやすくなると考えられる．

4.6　GoogLeNet の認識精度

float と fp8 を用いて 2 ノードで GoogLeNet の学習を行った際の top-1 accuracy を図 **13** に示す．top-1 accuracy は 100 反復ごとに計算した．

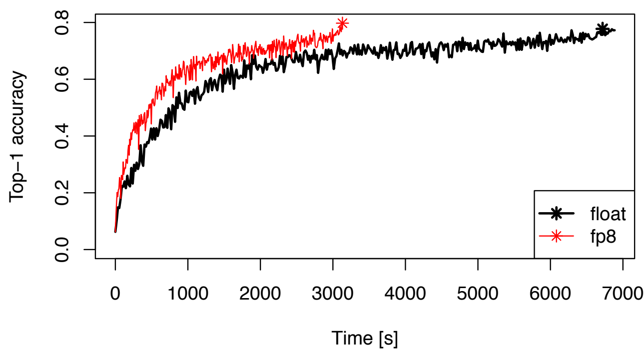
quick では float の最良 top-1 accuracy は 77.75%，fp8 は 79.75%であり，fp8 の 60 エポックの学習速度は float の 2.19 倍のスピードアップだった．

一方で slow では float の最良 top-1 accuracy は 83.125%，fp8 は 80.75%であり，accuracy が 2.375% 低下した．これは fp8 を用いることにより float と比較して計算誤差が大きくなったためであると考えられる．ただし，fp8 の 250 エポックの学習速度自体は float の 2.22 倍のスピードアップであり，また特定の top-1 accuracy に到達するまでの実行時間も 1.7 倍程度高速だった (図 **14**)．よってこのような accuracy の低下が許容できるかどうかは DNN やデータセットの構成，GPU 数などが要因となる実際の実行時間とのトレードオフにより異なるといえる．

5.　まとめと今後の課題

本論文の提案手法では 16 GPU を用いた CaffeNet と GoogLeNet の学習において認識精度を悪化させることなく 2 倍以上の高速化を達成した．一方で，提案手法は weak scale(GPU ごとのバッチサイズを固定する場合) において

ⓒ 2017 Information Processing Society of Japan



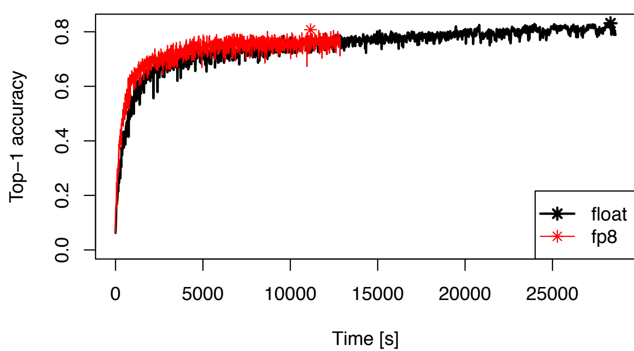


図 13　TSUBAME-KFC/DL で GoogLeNet の学習を行った場合の実行時間に対する top-1 accuracy (上:quick，下:slow)

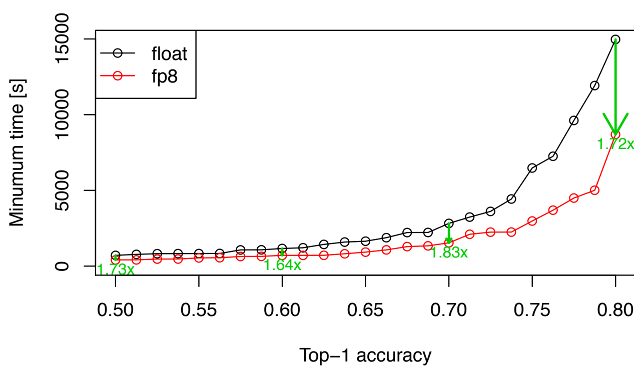


図 14　TSUBAME-KFC/DL で GoogLeNet の学習を行った場合の特定の top-1 accuracy に到達するまでの最小実行時間 (slow)．それぞれの点に対応する実行時間は top-1 accuracy の実測値を線形補間して求めた．また，10%ごとに fp8 と float の実行速度比を表示した．

は 192 GPU 程度まではよくスケールしたものの，strong scale(全体のミニバッチサイズを固定する場合) ではいぜん 2〜4 ノード程度で最も 1 反復の実行時間が短く，スケーラビリティの向上には至っていない．また，図 11 からも分かるように通信手法や並列数による悪化の影響は自明ではない．機械学習・DL の適応的な高速化では認識精度と学習時間がトレードオフの関係にあり，特にミニバッチサイズ (並列数) と認識精度の関係は先行研究で指摘されているが，本提案手法のような少精度化を行う場合は学習時間を短縮するための並列数や通信手法等の学習条件は自明ではない．よって，このような最適な学習条件を予測するためには通信されるデータの計算誤差の評価や性能モデリング等を用いたより精緻な分析が不可欠であると考えられる．

謝辞　本研究の一部は JST, CREST の支援を受けたものである．

参考文献

[1] He, K., Zhang, X., Ren, S. and Sun, J.: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, *CoRR*, Vol. abs/1502.01852 (online), available from ⟨http://arxiv.org/abs/1502.01852⟩ (2015).

[2] Amodei, D., Anubhai, R., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G., Elsen, E., Engel, J., Fan, L., Fougner, C., Han, T., Hannun, A., Jun, B., LeGresley, P., Lin, L., Narang, S., Ng, A., Ozair, S., Prenger, R., Raiman, J., Satheesh, S., Seetapun, D., Sengupta, S., Wang, Y., Wang, Z., Wang, C., Xiao, B., Yogatama, D., Zhan, J. and Zhu, Z.: Deep Speech 2: End-to-End Speech Recognition in English and Mandarin, *ArXiv e-prints* (2015).

[3] Simonyan, K. and Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition, *CoRR*, Vol. abs/1409.1556 (online), available from ⟨http://arxiv.org/abs/1409.1556⟩ (2014).

[4] Krizhevsky, A., Sutskever, I. and Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks, *Advances in Neural Information Processing Systems 25* (Pereira, F., Burges, C., Bottou, L. and Weinberger, K., eds.), Curran Associates, Inc., pp. 1097–1105 (online), available from ⟨http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf⟩ (2012).

[5] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A.: Going Deeper With Convolutions, *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).

[6] Chilimbi, T., Suzue, Y., Apacible, J. and Kalyanaraman, K.: Project Adam: Building an Efficient and Scalable Deep Learning Training System, *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, Broomfield, CO, USENIX Association, pp. 571–582 (online), available from ⟨https://www.usenix.org/conference/osdi14/technical-sessions/presentation/chilimbi⟩ (2014).

[7] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., Le, Q. V. and Ng, A. Y.: Large Scale Distributed Deep Networks, *Advances in Neural Information Proceessing Systems 25* (Bartlett, P., Pereira, F., Burges, C., Bottou, L. and Weinberger, K., eds.), pp. 1232–1240 (online), available from

ⓒ 2017 Information Processing Society of Japan

⟨http://books.nips.cc/papers/files/nips25/NIPS2012_0598.pdf⟩ (2012).

[8] Iandola, F. N., Ashraf, K., Moskewicz, M. W. and Keutzer, K.: FireCaffe: near-linear acceleration of deep neural network training on compute clusters, *CoRR*, Vol. abs/1511.00175 (online), available from ⟨http://arxiv.org/abs/1511.00175⟩ (2015).

[9] Wu, R., Yan, S., Shan, Y., Dang, Q. and Sun, G.: Deep Image: Scaling up Image Recognition, *CoRR*, Vol. abs/1501.02876 (online), available from ⟨http://arxiv.org/abs/1501.02876⟩ (2015).

[10] 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，情報処理学会研究報告， Vol. 2016-HPC-155 (2016).

[11] Oyama, Y., Nomura, A., Sato, I., Nishimura, H., Tamatsu, Y. and Matsuoka, S.: Predicting statistics of asynchronous SGD parameters for a large-scale distributed deep learning system on GPU supercomputers, *2016 IEEE International Conference on Big Data (Big Data)*, pp. 66–75 (online), DOI: 10.1109/BigData.2016.7840590 (2016).

[12] Seide, F., Fu, H., Droppo, J., Li, G. and Yu, D.: 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs, *Interspeech 2014*, (online), available from ⟨https://www.microsoft.com/en-us/research/publication/1-bit-stochastic-gradient-descent-and-application-to-data-parallel-distributed-training-of-speech-dnns/⟩ (2014).

[13] Strom, N.: Scalable distributed DNN training using commodity GPU cloud computing, *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*, pp. 1488–1492 (2015).

[14] Dryden, N., Jacobs, S. A., Moon, T. and Van Essen, B.: Communication Quantization for Data-parallel Training of Deep Neural Networks, *Proceedings of the Workshop on Machine Learning in High Performance Computing Environments*, MLHPC '16, Piscataway, NJ, USA, IEEE Press, pp. 1–8 (online), DOI: 10.1109/MLHPC.2016.4 (2016).

[15] Gupta, S., Zhang, W. and Milthorpe, J.: Model Accuracy and Runtime Tradeoff in Distributed Deep Learning, *ArXiv e-prints* (2015).

[16] Sallinen, S., Satish, N., Smelyanskiy, M., Sury, S. S. and R, C.: High Performance Parallel Stochastic Gradient Descent in Shared Memory, *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 873–882 (online), DOI: 10.1109/IPDPS.2016.107 (2016).

[17] Krizhevsky, A.: One weird trick for parallelizing convolutional neural networks, *CoRR*, Vol. abs/1404.5997 (online), available from ⟨http://arxiv.org/abs/1404.5997⟩ (2014).

[18] Gupta, S., Agrawal, A., Gopalakrishnan, K. and Narayanan, P.: Deep Learning with Limited Numerical Precision, *CoRR*, Vol. abs/1502.02551 (online), available from ⟨http://arxiv.org/abs/1502.02551⟩ (2015).

[19] LeCun, Y. and Cortes, C.: MNIST handwritten digit database, (online), available from ⟨http://yann.lecun.com/exdb/mnist/⟩ (2010).

[20] Krizhevsky, A., Nair, V. and Hinton, G.: CIFAR-10, (online), available from ⟨http://www.cs.toronto.edu/ kriz/cifar.html⟩.

[21] : IEEE Standard for Floating-Point Arithmetic, *IEEE Std 754-2008*, pp. 1–70 (online), DOI: 10.1109/IEEESTD.2008.4610935 (2008).

[22] Harris, M.: New Features in CUDA 7.5. `https://devblogs.nvidia.com/parallelforall/new-features-cuda-7-5/`.

[23] Han, S., Pool, J., Tran, J. and Dally, W. J.: Learning both Weights and Connections for Efficient Neural Networks, *CoRR*, Vol. abs/1506.02626 (online), available from ⟨http://arxiv.org/abs/1506.02626⟩ (2015).

[24] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. and Darrell, T.: Caffe: Convolutional Architecture for Fast Feature Embedding, *arXiv preprint arXiv:1408.5093* (2014).

[25] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard Version 3.0 (2012). Chapter author for Collective Communication, Process Topologies, and One Sided Communications.

[26] Intel Corporation: Intel(R) 64 and IA-32 Architectures Software Developer's Manual. `https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf`.

[27] 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 (online), DOI: 10.1007/s11263-015-0816-y (2015).

[28] Thakur, R., Rabenseifner, R. and Gropp, W.: Optimization of Collective Communication Operations in MPICH, *Int. J. High Perform. Comput. Appl.*, Vol. 19, No. 1, pp. 49–66 (online), DOI: 10.1177/1094342005051521 (2005).

ⓒ 2017 Information Processing Society of Japan