ディープラーニングの訓練向けインテル® プロセッサー

この記事は、インテル® デベロッパー・ゾーンに公開されている「Intel® Processors for Deep Learning Training」の日本語参考訳です。

はじめに

2017 年 11 月 7 日、カリフォルニア大学バークレー校、テキサス大学オースティン校、カリフォルニア大学 デイビス校の研究者が、CPU 上において最先端の精度で ResNet-50 では 31 分 (論文発表時点の最速記録)、AlexNet では 11 分という記録的な訓練結果を発表しました [1]。これらの結果は、インテル® Xeon® スケーラブル・プロセッサー (開発コード名 Skylake-SP) 上で達成されました。これらのパフォーマンスを達成できた主な要因は、以下のとおりです。

1. インテル® Xeon® スケーラブル・プロセッサーの計算処理能力とメモリー容量
2. ディープ・ニューラル・ネットワーク向けインテル® マス・カーネル・ライブラリー (インテル® MKL-DNN) とポピュラーなディープラーニング・フレームワークのソフトウェアの最適化
3. 教師ありディープラーニング・ワークロード向け分散訓練アルゴリズムの最近の進歩

このレベルのパフォーマンスは、インテル® Xeon® プロセッサーがディープラーニングの訓練のための優れたハードウェア・プラットフォームであることを示しています。データ・サイエンティストは、既存の汎用インテル® Xeon® プロセッサー・ベースのクラスターを、これまでのディープラーニングの推論、古典的なディープラーニング、およびビッグデータ・ワークロードに加えて、ディープラーニングの訓練にも使用できます。1 サーバーノードで優れたディープラーニングの訓練パフォーマンスが得られますが、サーバーノードの数を増やすことで数百ノードへ直線的に近いスケーリングを達成して訓練時間をさらに短縮できます。

この記事は 4 つのパートで構成されます。記録的なスピードを達成した 3 つの主な要因について 1 つずつ調査し、商用での利用例を紹介します。

パート 1: インテル® Xeon® スケーラブル・プロセッサーの計算処理能力とメモリー容量

ディープラーニングの訓練には、通常、非常に多くの計算が必要になります。例えば、ResNet-50 [2] の訓練には、エクサ (10^{18}) 規模の単精度演算 [1] が必要です。計算スループットの高いハードウェアは、高利用率を達成できれば訓練時間を短縮できます。高利用率を達成するには、チップ上で計算をビジーに保つため、高帯域幅メモリーと賢明なメモリー管理が必要です [3]。最新世代のインテル® Xeon® プロセッサーは、これらの機能 (高周波数の多数のコア、高速システムメモリー、大きなコアごとの中間レベルキャッシュ (MLC または L2 キャッシュ)、新しい SIMD 命令) を備えており、ディープラーニング・モデルの訓練に最適なプラットフォームと言えます。パート 1 では、計算とメモリーを含むインテル® Xeon® スケーラブル・プロセッサーの主なハードウェア機能について説明し、インテル® Xeon® スケーラブル・プロセッサーと前世代のインテル® Xeon® プロセッサーのディープラーニング・ワークロードのパフォーマンスを比較します。

2017 年 7 月、インテルは、14nm 製造プロセス (インテル® 14nm テクノロジー) によって製造されたインテル® Xeon® スケーラブル・プロセッサーをリリースしました。インテル® Xeon® スケーラブル・プロセッサーは、ソケットごとに最大 28 物理コア (56 スレッド) (最大 8 ソケット)、ベース周波数 2.50GHz、最大ター

ボ周波数 3.80GHz、6 メモリーチャネル、最大 1.5TB DDR4 メモリー 2666MHz をサポートします。上位のインテル® Xeon® Platinum 8180 プロセッサーは、2 ソケットのシステム上で最大 199GB/s の STREAM Triad パフォーマンスを実現します [a,b]。インテル® Xeon® スケーラブル・プロセッサーは、ソケット間のデータ転送に新しいインテル® ウルトラ・パス・インターコネクト (インテル® UPI) を採用しています。インテル® UPI は、インテル® QuickPath インターコネクト (インテル® QPI) に代わるコヒーレント・インターコネクトで、データ転送速度は UPI ポートごとに 10.4GT/s、2 ソケットのシステム構成では最大 3 UPI ポートをサポートします [4]。

さらに、38.5MB の非包含的な (つまり、メモリー読み取りは直接 L2 に書き込まれ、L2 と L3 の両方には書き込まれない) 共有最終レベルキャッシュ (LLC または L3 キャッシュ)、コアごとに 1MB のプライベート L2 キャッシュがあります。インテル® Xeon® スケーラブル・プロセッサーのコアには、512 ビット・ベクトル・エンジンの一部として 512 ビットの FMA (Fused Multiply Add) 命令が含まれており、コアごとに最大 2 つの 512 ビット FMA ユニットを並列に計算できます (これは、インテル® Xeon Phi™ 製品ファミリーで追加されました)[1] [4]。これにより、以前のインテル® Xeon® プロセッサー v3 ファミリー (開発コード名 Haswell) およびインテル® Xeon® プロセッサー v4 ファミリー (開発コード名 Broadwell) の 256 ビットのインテル® アドバンスト・ベクトル・エクステンション 2 (インテル® AVX2) 命令と比較して、訓練と推論の両方のワークロードでパフォーマンスの大幅な向上をもたらします。

インテル® Xeon® Platinum 8180 プロセッサーは、パケットごとに最大 3.57TFLOPS (FP32) と最大 5.18TOPS (INT8)[2] を達成しました。512 ビットの FMA は、基本的にインテル® Xeon® スケーラブル・プロセッサーの FLOPS を 2 倍にし、単精度行列演算を大幅にスピードアップします。以前のインテル® Xeon® プロセッサー v4 ファミリーの SGEMM と IGEMM のパフォーマンスと比較して、それぞれ 2.3 倍と 3.4 倍の向上が見られました [c,e]。完全なディープラーニング・モデルを使用したパフォーマンスの比較では、ResNet-18 とインテル® Neon™ Framework のほうが、FP32 を使用する以前のインテル® Xeon® プロセッサー v4 ファミリーよりも、訓練と推論のスループットがそれぞれ 2.2 倍と 2.4 倍に向上しました [d,f]。

パート 2: インテル® MKL-DNN とメイン・フレームワークのソフトウェアの最適化

高利用率を達成し、パフォーマンスを向上するには、ソフトウェアの最適化が不可欠です。インテルにより最適化された Caffe (インテルの Caffe とも呼ばれます)、TensorFlow*、Apache* MXNet、インテル® Neon™ Framework は、訓練と推論向けに最適化されています。Caffe2、CNTK、PyTorch*、PaddlePaddle など、ほかのフレームワークの最適化も現在進行中です。パート 2 では、インテルにより最適化されたモデルとそうでないモデルのパフォーマンスを比較し、インテル® MKL-DNN ライブラリーにより CPU 利用率を高める方法、インテル® MKL とインテル® MKL-DNN の相違点、そしてさらなるパフォーマンス向上をもたらすフレームワーク・レベルのその他の最適化について説明します。

2 年前、インテル® プロセッサーのディープラーニング・パフォーマンスは、ソフトウェアの最適化が制限され、CPU 利用率が低かったため、最適とは言えませんでした。このため、ディープラーニング・サイエンティストは、CPU はディープラーニング・ワークロードには適していないと誤解していました。この 2 年間に、インテルはディープラーニング関数の最適化に取り組み、CPU 利用率を向上し、ディープラーニング・サイエンティストが既存の汎用インテル® プロセッサーをディープラーニングの訓練に使用できるようにしました。一般的なディープラーニング・フレームワークをビルドするときに設定フラグをセットするだけで (デフォルトでは、フレームワークがインテル® MKL-DNN を自動でダウンロードしてビルドするため)、データ・サイエンティストはインテル® プロセッサーの最適化を利用できます。

インテル® Xeon® プロセッサー上でインテル® MKL-DNN を利用することで、100 倍を超えるパフォーマンスの向上が可能です。

- **インテル® Xeon® プロセッサー E5-2699 v3 上で Apache* MXNet とすべての利用可能な CPU コアを利用した AlexNet、GoogLeNet v1、ResNet-50、GoogLeNet v3 の推論 (c4.8xlarge AWS* EC2* インスタンス) では、スループットがそれぞれ 111 倍、109 倍、66 倍、92 倍に向上しました [5]。**
- **インテル® Xeon® プロセッサー E5-2699 v4 上で Caffe2 とすべての利用可能な CPU コアを利用した AlexNet の推論では、39 倍のスループットの向上が得られました [6]。**
- **インテル® Xeon® プロセッサー E5-2699 v4 上で TensorFlow* を利用した AlexNet、GoogLeNet、VGG の訓練では、スループットがそれぞれ 17 倍、6.7 倍、40 倍向上しました [7]。**
- **インテル® Xeon® Platinum 8180 プロセッサー上でインテルにより最適化された Caffe およびインテル® MKL-DNN とすべての CPU コアを利用した AlexNet の訓練では、インテル® Xeon® プロセッサー E5-2699 v3 上でインテル® MKL-DNN なしで BVLC-Caffe を利用した場合よりもスループットが 113 倍向上しました [d,g]。図 1 は、複数の世代のインテル® Xeon® プロセッサー上でインテル® MKL-DNN ライブラリーを利用した訓練のスループットを比較したものです。**

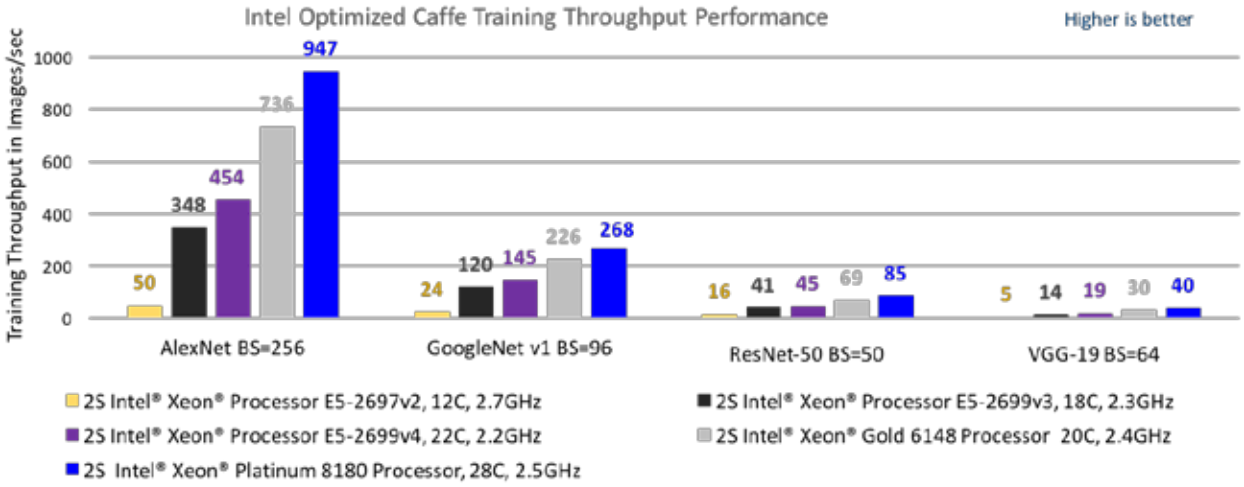

図 1: インテル® Xeon® プロセッサー v2 (開発コード名 Ivy Bridge)[h]、インテル® Xeon® プロセッサー v3 (開発コード名 Haswell)[g]、インテル® Xeon® プロセッサー v4 (開発コード名 Broadwell)[e]、インテル® Xeon® Gold プロセッサー[f] およびインテル® Xeon® Platinum プロセッサー (開発コード名 Skylake)[d] 上で AlexNet バッチサイズ (BS) = 256、GoogLeNet v1 BS=96、ResNet-50 BS=50、VGG-19 BS=64 を使用したインテルにより最適化された Caffe の訓練スループット。インテル® MKL-DNN は、インテル® AVX-2 命令が採用されたインテル® Xeon® プロセッサー v3 ファミリー以降でパフォーマンスの大幅な向上をもたらし、インテル® AVX-512 命令が採用されたインテル® Xeon® スケーラブル・プロセッサーではパフォーマンスがさらに向上しています。

これらの最適化で重要な役割を果たしているのが、インテル® マス・カーネル・ライブラリー (インテル® MKL) [8] とインテル® MKL-DNN ライブラリー [9] です。さまざまなディープラーニング・モデルがあり、それらは非常に異なって見えるかもしれません。しかし、ほとんどのモデルは、テンソルを操作するプリミティブと呼ばれるビルディング・ブロックの少数のセットから構築されています。これらのプリミティブには、内積、畳み込み、正規化線形関数 (ReLU)、バッチ正規化などに加えて、テンソル操作に必要な関数があります。これらのビルディング・ブロック (低水準のディープラーニング関数) は、インテル® MKL ライブラリーの内部でインテル® Xeon® プロセッサー・ファミリー向けに最適化されています。インテル® MKL には多くの数学関数が含まれていますが、ディープラーニングに使用されるのはその一部のみです。ディープラーニングに特化したライブラリーを提供し、ディープラーニング開発者と連携するため、複雑なモデルの構築に必要なすべての主要ビルディング・ブロックを含むインテル® MKL-DNN が Apache License 2.0 の下にオープンソースとしてリリースされました。

業界および学術分野のディープラーニング開発者は、インテル® MKL-DNN を配布したり、新しい関数や改善された関数を提供することができます。インテル® MKL-DNN では、すべての新しい最適化が最初に実装されるため、最先端のパフォーマンスが見込めます。

インテル® MKL-DNN では、プリフェッチ、データレイアウト、データ再利用、ベクトル化、レジスター・ブロッキング手法を組込むことで、ディープラーニング・プリミティブが最適化されています [9]。高 CPU 利用率を実現するには、実行ユニット (EU) が必要なときにデータが利用可能でなければなりません。そのためには、メインメモリーから同じデータを複数回フェッチするのではなく、データのプリフェッチとキャッシュにあるデータの再利用が必要になります。キャッシュ・ブロッキングの目的は、キャッシュ (通常 MLC) に収まるデータブロックに対する計算を最大化することです。データレイアウトは、メモリー上で連続するように配置されます。これは、最内ループへのアクセスができるだけ連続するようにして、不要なギャザー/スキャッター操作を回避するためです。これにより、キャッシュライン (そして帯域幅) の効率が改善され、プリフェッチのパフォーマンスが向上します。ブロックをループする際に、ブロックの外側の次元を SIMD 幅の倍数にし、最も内側の次元が SIMD 幅のグループをループすることで、効率良くベクトル化できます。FMA 命令のレイテンシーを隠蔽するため、レジスター・ブロッキングが必要になることがあります [3]。

高 CPU 利用率には、OpenMP* を使用してバッチ全体を並列化するなど、すべてのコアにわたる並列化も重要です。そのためには、各コアの作業量が等しくなり、コア間の同期イベントが減るように、ロードバランスを改善する必要があります。バランスよくすべてのコアを効率的に利用するには、レイヤー内の並列化が必要です。

これらの最適化は、畳み込み、行列乗算、バッチ正規化などのすべての主要なディープラーニング・プリミティブが、最新の SIMD 向けにベクトル化され、すべてのコアにわたって並列化されるようにします [10]。インテル® MKL-DNN プリミティブは、一般的なディープラーニング関数向けの C/C++ API を利用して C で実装されています [11]。

- 直接バッチ畳み込み
- 内積
- プーリング: 最大、最小、平均
- 正規化: チャネル全体にわたる局所反応正規化 (LRN)、バッチ正規化
- 活性化: 正規化線形関数 (ReLU)、ソフトマックス
- 融合プリミティブ: 畳み込み + ReLU
- データ操作: 多次元転置 (変換)、分割、連結、合計、スケール
- 将来実装予定: LSTM (Long Short-Term Memory)、GRU (Gated Recurrent Units)

Caffe、TensorFlow*、MXNet、PyTorch* など、複数のディープラーニング・フレームワークがあります。インテル® MKL-DNN プリミティブを効率良く利用するには、フレームワーク・レベルでの変更 (コードのリファクタリング) が必要です。フレームワークは、フレームワークとインテル® MKL-DNN ライブラリーが同じスレッドで競合しないように、既存のディープラーニング関数の呼び出しを適切なインテル® MKL-DNN API に置換します。セットアップ中、フレームワークは、インテル® MKL-DNN のレイアウトへ変換し、出力と入力のデータレイアウトが一致しない場合は一時配列を割り当てます。パフォーマンスを向上するには、異なるデータレイアウト間の変換を最小限に抑えるため、グラフの最適化が必要になる場合があります。実行フェーズでは、データは BCWH (バッチ、チャネル、幅、高さ) などのプレーンなレイアウトでネットワークに渡され、SIMD 対応レイアウトに変換されます。レイヤーとレイヤーの間のデータ伝達では、データレイアウトが保持され、インテル® MKL-DNN でサポートされていない操作を実行する必要がある場合は変換されます [10]。

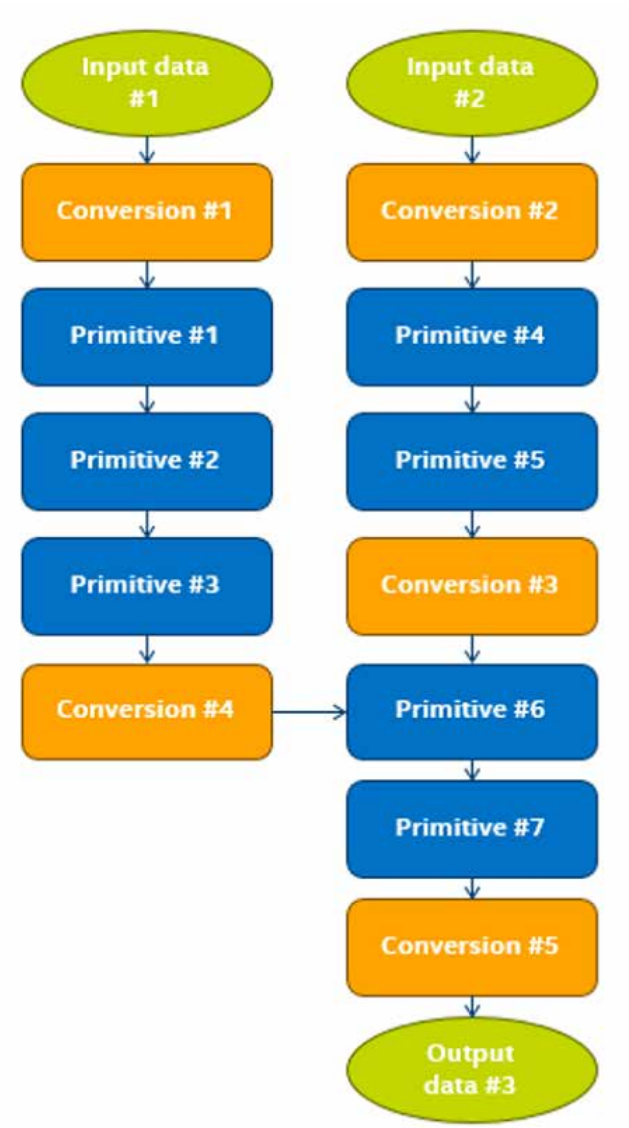

図 2: 操作フロー。インテル® MKL-DNN プリミティブは青色で示されています。フレームワークの最適化は、連続するプリミティブ操作でインテル® MKL-DNN のデータレイアウトを保持できるように、レイアウト変換の軽減を試みます。イメージの出典: [10]。

パート 3: ディープラーニング向け分散訓練アルゴリズムの進歩

大規模なディープラーニング・モデルの訓練には、数日あるいは数週間かかることがあります。複数のサーバーノードに計算処理を分散することで、訓練時間を短縮できます。これには、使用するハードウェアに関係なく、アルゴリズムの課題が伴いますが、分散アルゴリズムに関する最近の進歩により、これらの課題のいくつかを軽減できます。パート 3 では、勾配降下法 (GD) および確率的勾配降下法 (SGD) アルゴリズムについて考察し、大規模なバッチを利用する訓練の制約事項を説明します。モデルとデータ並列性について述べ、同期 SGD (SSGD)、非同期 SGD (ASGD)、オールレデュース/ブロードキャスト・アルゴリズムを検証します。そして、大きなバッチサイズの SSGD 訓練と最先端の結果を可能にする最新の進歩を紹介します。

教師ありディープラーニングでは、入力データはモデルを介して渡され、出力は正解または予測される結果と比較されます。そして、ペナルティーや損失が計算されます。モデルの訓練には、この損失を抑えるため、モデル・パラメーターの調整が含まれます。勾配降下法やその変形である確率的勾配降下法、Adagrad、Adadelta、RMSprop、Adam など、損失関数を最小限に抑えるさまざまな最適化アルゴリズムがあります。

最急降下法とも呼ばれる勾配降下法 (GD) では、重みのセットによって定義された特定のモデルの損失関数が、データセット全体に対して計算されます。重みは、勾配の反対方向に (つまりローカル最小値に向かって) 移動することで更新されます。

更新後の重み = 現在の重み – 学習率 * 勾配

ミニバッチ勾配降下法とも呼ばれる確率的勾配降下法 (SGD) では、データセットはいくつかのバッチに分割されます。バッチの損失を計算し、勾配降下法と同じ更新規則に従って重みが更新されます。このほかにも、勾配の反対方向に速度 (モーメンタムとも呼ばれる) を蓄積することで訓練プロセスをスピードアップしたり、勾配のノルムに応じて学習率を自動的に変更することでデータ・サイエンティストが適切な学習率を選択しなくても済むようにする手法があります。各手法の詳細は、[12] を参照してください。

バッチサイズが大きくなると SGD の動作は GD の動作に近づき、バッチサイズがデータセット全体と等しくなると 2 つの動作は同じになります。GD には主に 3 つの課題があります (これらは、バッチサイズが非常に大きい SGD の課題でもあります)。

- 1 つ目は、各ステップまたは反復がデータセット全体の損失を計算する必要があるため、計算負荷が高くなることです。
- 2 つ目は、勾配がゼロに近く、曲率が切り替わる鞍点付近では学習率が下がることです。
- 3 つ目は、インテルとノースウェスタン大学の研究者 [13] の報告によると、最適化領域には多くの鋭い最小値があるように見受けられることです。

勾配降下法は、最適化領域を調査せず、開始点の直下にある局所的な最小値 (多くの場合は鋭い最小値) に向かいます。鋭い最小値は一般化されません。損失関数はテスト・データセットと訓練データセットで似ていますが、鋭い最小値の実際のコストは大きく異なる可能性があります。これを視覚化したものが 図 3 です。テスト・データセットの損失関数は、訓練データセットの損失関数からわずかにシフトしています。このシフトにより、鋭い最小値に収束するモデルは、テスト・データセットではコストが高く、訓練セット以外のデータでは一般化しません。一方、平らな最小値に収束するモデルは、テスト・データセットではコストが低く、訓練セット以外のデータで一般化します。

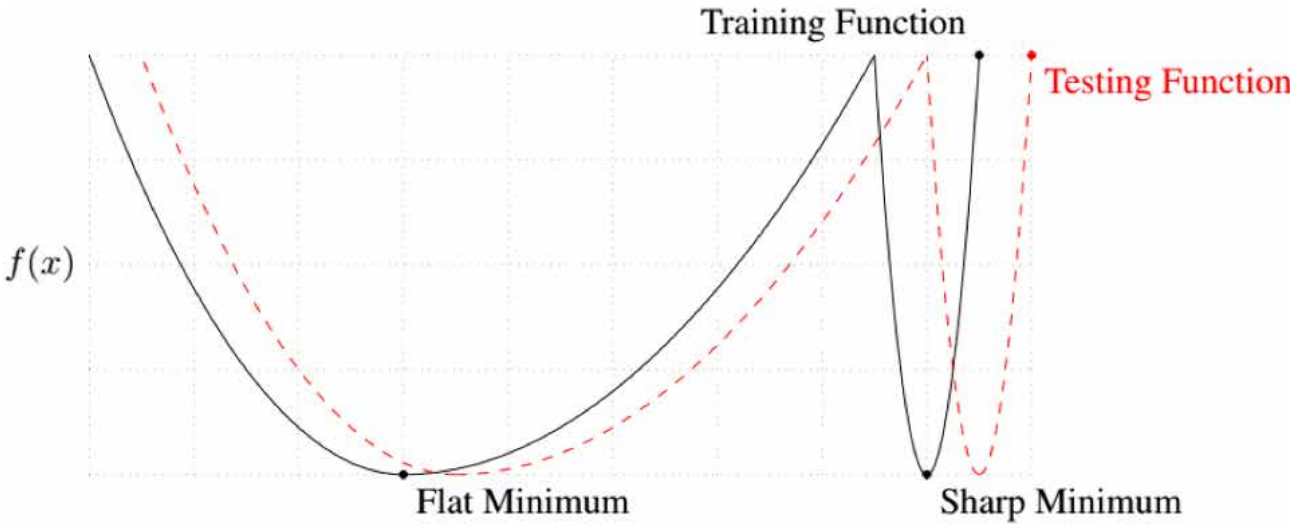

図 3: テスト・データセットの損失関数は、訓練データセットの損失関数からわずかにシフトしている。鋭い最小値は、テスト損失関数ではコストが高くなる。イメージの出典: [13]。

スモールバッチ SGD (SB-SGD) を利用してこれらの課題に対応できます。

- まず、SB-SGD は計算コストが低く、各反復を高速に実行できます。通常、データセット全体に対して多数の反復があります (GD では 1 反復のみ)。SB-SGD では、多くの場合、データセット全体の通過回数が少なくて済むため、短時間で訓練できます。
- 2 つ目に、訓練セット全体の勾配がゼロであっても、訓練セットのいくつかのバッチの勾配はゼロでない可能性が高いため、SB-SGD では鞍点で停滞することは極めてまれです。
- 3 つ目に、SB-SGD は開始点の直下にある局所的な最小値に向かう代わりに、解空間を詳しく検索するため、平らな最小値を見つけやすくなります。一方、非常に小さなバッチは、高い CPU (または GPU) 使用率が得られないため理想的とは言えません。これは特に、計算ワークロードを複数のワーカーノードに分配する際に問題となります。そのため、高 CPU 使用率を維持するのに十分な大きさでありながら、GD の問題を回避できる大きすぎないバッチサイズを見つけることが重要です。これは、後述の同期データ並列 SGD ではさらに重要になります。

複数のワーカーノードにワークロードを効率良く分配することで、全体の訓練時間を短縮できます。使用される主な手法は、モデル並列性とデータ並列性の 2 つです。

- モデル並列性では、ワーカーノード間でモデルが分割され、各ノードは同じバッチを処理します。モデル並列性は、メモリー要件がワーカーのメモリー容量を超える場合に使用されます。データ並列性は、より一般的なアプローチであり、重み付けの少ないモデルに最適です。
- データ並列性では、ワーカーノード間でバッチが分割され、各ノードは完全なモデルを持ち、バッチの一部 (ノードバッチと呼ばれる) を処理します。各ワーカーノードは、ノードバッチの勾配を計算します。計算された勾配は、レデュース・アルゴリズムを使用して集約され、バッチ全体の勾配が計算されます。そして、モデルの重み付けが更新され、更新された重み付けが各ワーカーノードにブロードキャストされます。これは、レデュース/ブロードキャストまたはオールレデュース (allreduce) 手法として知られています (オールレデュース・オプションのリストは後述します)。バッチの各反復またはサイクルの最後に、ノードが同期され、すべてのワーカーノードは同じ更新済みモデルを持ちます。これは、同期 SGD (SSGD) として知られています。

非同期 SGD (ASGD) は、同期のオーバーヘッドを軽減します。ただし、ASGD にはさらなる課題があります。ASGD では、モーメンタムなどのハイパーパラメーターをさらにチューニングする必要があり、訓練により多くの反復が必要です。その上、単一ノードのパフォーマンスに一致しないため、デバッグが困難です。実際には、ASGD は大規模なモデルにおいてスケーリングと精度の維持をまだ実現できていません。スタンフォード大学、ローレンス・バークレー国立研究所 (LBNL)、インテルの研究者は、ノードが最大 8 つのグループにクラスター化されている場合、ASGD/SSGD ハイブリッド・アプローチが機能することを示しました。グループ内の更新は同期で、グループ間の更新は非同期です。8 グループを超えると、ASGD の課題によりパフォーマンスが低下します [14]。

勾配の伝達方法の 1 つとして、各ノードの勾配を合計し、モデルを更新して、更新済みの重み付けを各ワーカーに送信するパラメーター・サーバーとして、1 つのノードを指定します。ただし、1 つのパラメーター・サーバーですべての勾配の送受信を行うことはボトルネックにつながります。ASGD を使用しない場合、パラメーター・サーバーの使用は推奨しません。

オールレデュースとブロードキャスト・アルゴリズムを使用して、ノードの勾配の伝達と追加を行い、更新済みの重み付けをブロードキャストします。オールレデュース・アルゴリズムには、ツリー、バタフライ、リングなどさまざまな種類があります。バタフライは、レイテンシーに優れており、*O(log(P))* 反復でスケーリングします。*P*

はワーカーノードの数で、レデュース-スキャッターとブロードキャストの組み合わせです。リングは帯域幅に優れており、大規模なデータの伝達では、ノード数に応じて *O(1)* でスケーリングします。帯域幅が制限されるクラスターでは、通常、オールレデュース・リング・アルゴリズムが適しています。オールレデュース・リング・アルゴリズムの詳細は、[15] を参照してください。

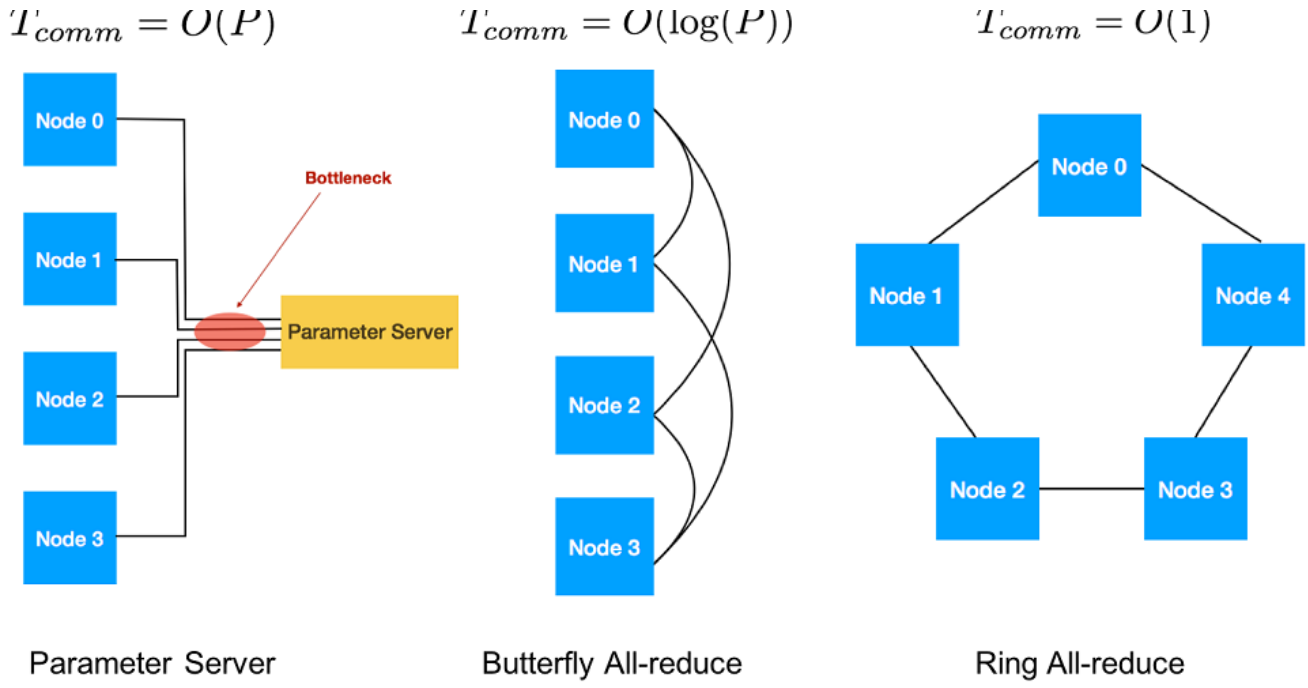

図 4: さまざまな伝達手法。バタフライ・オールレデュースはレイテンシーに優れており、リング・オールレデュースは帯域幅に優れている。イメージの出典: Amir Gholami、Peter Jin、Yang You、Kurt Keutzer、カリフォルニア大学バークレー校の PALLAS グループ

2014 年 11 月、Jeff Dean 氏は、訓練期間を 6 週間から 1 日に短縮するという Google* の研究目標について述べました [16]。そして 3 年後、CPU を使用した AlexNet の訓練が 11 分で行われました! これは、計算ワークロードを 1000 を超えるノードに分配する大きなバッチサイズを使用することで達成できました。効率良くスケーリングするには、勾配の計算において、勾配と更新済み重み付けの伝達を隠蔽する必要があります。

次の手法により、全体のバッチサイズを増やすことができます。

1) バッチサイズと学習率を比例して増やします。
2) 訓練の初期段階の学習率 (ウォーミングアップ学習率) を徐々に上げます。
3) LARS (Layer-wise Adaptive Rate Scaling) アルゴリズムを使用してモデルの各層が異なる学習率になるようにします。それぞれの手法について詳しく見てみましょう。

バッチサイズが大きいほうが勾配の信頼度が高くなるため、より大きな学習率を使用できます。経験則では、学習率はバッチサイズに比例して増加します [4] [17]。この手法により、カリフォルニア大学バークレー校の研究者は、GoogLeNet モデルでバッチサイズを 256 から 1024 に増やし、128 の K20-GPU ノードへスケーリングして、訓練期間を 21 日から 10.5 時間に短縮しました [18]。また、インテルの研究者は、VGG-A でバッチサイズを 128 から 512 に増やし、128 のインテル® Xeon® プロセッサー E5-2698 v3 ベースのノードにスケーリングしました [19]。

大きな学習率は、初期訓練段階で発散 (反復ごとに損失が減るのではなく増える) につながる可能性があります。これは、初期訓練段階では、勾配のノルムが重み付けのノルムよりもはるかに大きいためです [20]。目標

の学習率に達するまで、初期訓練段階 (例えば、最初の 5 エポック) では学習率を徐々に増やすことでこれを軽減できます。この手法により、Facebook* の研究者は、ResNet-50 でバッチサイズを 256 から 8096 に増やし、256 の P100-GPU ノードへスケーリングして、訓練期間を 29 時間 (8 つの P100-GPU を使用) から 1 時間に短縮しました [21]。また、SURFsara とインテルの研究者は、512 の 2 基のインテル® Xeon® Platinum プロセッサーにスケーリングして、ResNet-50 の訓練期間を 40 分に短縮しました [22]。

NVIDIA* の研究者は、モデル内の各レイヤーで重み付けに対する勾配の比率が大きく異なることに気付きました [20]。そして、レイヤーごとにこの比率に反比例する学習率を使用することを提案しました。この手法 (と前述の手法の組み合わせ) により、彼らはバッチサイズを 32K に増やしました。

カリフォルニア大学バークレー校、テキサス大学オースティン校、カリフォルニア大学デイビス校の研究者は、これらの手法を使用して、インテル® プロセッサー上において最先端の精度で、AlexNet では 11 分、ResNet-50 では 31 分という、訓練期間の最高記録を達成しました (2017 年 11 月 7 日の論文発表時点) [1]。彼らは、それぞれ 1024 台と 1600 台の 2 基のインテル® Xeon® Platinum 8160 プロセッサーベースのサーバーで、インテル® MKL-DNN ライブラリーとインテルにより最適化された Caffe フレームワーク [5] をし使用ました (SURFsara とインテルの研究者が、1024 の 2 基のインテル® Xeon® Platinum 8160 プロセッサー上において最先端の精度で、ResNet-50 の訓練を 42 分で行った [23] も参照してください)。[1] の検証では、インテル® Omni-Path アーキテクチャーにより接続された 1000 を超えるノードが使用されました。

多くのデータセンターでは、高帯域幅インターコネクトや数千のノードを利用できない可能性があります。しかし、10Gbps イーサネットを使用するだけで、ResNet-50 で 32 台の 2 基のインテル® Xeon® Gold 6148 プロセッサーベースのサーバーへの完璧な線形スケーリングと 64 台の同サーバーへの 99.8% のスケーリングを達成できます [i]。この 64 台のサーバーを使用した 90 エポックの訓練では、7.3 時間で 75.9% の Top-1 精度に達します。

パート 4: 商用での利用例

計算処理能力とメモリー容量の増加、ソフトウェアの最適化、分散訓練の進歩により、既存のインテル® Xeon® プロセッサーをディープラーニングに利用することができます。パート 4 では、インテルの組み立ておよび試験工場、Facebook* サービス、deepsense.ai* の強化学習 (RL) アルゴリズム、OpenAI* の進化戦略、Honeywell* のドローン検査サービス、BigDL [24] を採用しているさまざまなクラウドサービスプロバイダーを含むいくつかの商用での利用例を紹介します。

インテルの組み立ておよび試験工場では、インテル® Xeon® プロセッサー上でインテルにより最適化された Caffe を使用することで、シリコン製造パッケージの欠陥検出を改善しました。このプロジェクトの目的は、人間と同レベルの誤検出率を維持しつつ、最終検査において人間による表面的な損傷の検査率を減らすことでした。入力はパッケージ写真のセットで、それぞれに対してパッケージの合格/不合格を示すバイナリー分類を実行することが目標でした。このタスク向けに GoogLeNet モデルを改良しました。10Gb イーサネットにより接続された 8 つのインテル® Xeon® Platinum 8180 プロセッサーを使用して、訓練は 1 時間以内に完了しました。誤検出率は、一貫して目標の人間レベルの精度に達しました。このプロセスの自動化により、検査担当者の時間の 70% を節約できました [25]。

Facebook* のサービスで使用されているマシンラーニングアルゴリズムは、推論に CPU を使用し、訓練に CPU と GPU の両方を使用しています。訓練の頻度は、推論と比較すると非常に少ないです。推論フェーズは、1 日に数億回実行され、一般にリアルタイムで実行されなければなりません。マシンラーニングを利用する主なサービスには、ニュースフィード、広告、検索、Sigma (異常検出)、Lumos (特徴抽出器)、Facer (顔認識)、言

語翻訳、音声認識が含まれます。ニュースフィードと Sigma サービスは CPU で訓練し、Facer と検索アルゴリズムは CPU と GPU の両方で訓練しています [26]。

deepsense.ai* は、強化学習 (RL) にインテル® プロセッサーを使用しています。彼らは、64 のインテル® プロセッサー (12 コア) 上でさまざまな Atari 2600* ゲームをプレイするエージェントを、ときには完璧な線形スケーリングで、1 ゲームあたりわずか 20 分で訓練しました (注: 記事では、使用された特定のインテル® プロセッサーとインターコネクトが明記されていません) [27]。

OpenAI* は、RL ベンチマーク (Atari*/MuJoCo*) で優れたパフォーマンスを達成する進化戦略 (ES) を訓練するため、CPU を使用しています。80 台のマシンと 1,440 の CPU コアで構成されるクラスターを使用して、OpenAI* のエンジニアは 3D MuJoCo* ヒューマノイド・ウォーカーをたった 10 分で訓練することができました [28]。

Honeywell* は最近、商用無人航空機 (UAV) の検査サービスを開始しました [29]。彼らは、ソーラーパネルの欠陥検出のタスクに Faster-RCNN とインテルにより最適化された Caffe を使用しました。300 のソーラーパネルのイメージを 36° 回転したものを使用した訓練は、インテル® Xeon® Platinum 8180 プロセッサー上で 6 時間かかり、悪影響下において 96.3% の検出精度を達成しました。推論のパフォーマンスは、1 秒あたり 188 イメージでした。これは、石油/ガスの検査、パイプラインの浸水/漏れ、ユーティリティー検査、送電線と変電所、緊急時の危機対応を含む、市場におけるさまざまな検査サービスに使用できる一般的なソリューションです。

インテルが提供するオープンソースの分散ディープラーニング・フレームワークである BigDL [24] は、Apache Spark* で包括的なディープラーニング・サポートを提供します。高度にスケーラブルな Apache Spark* プラットフォームをベースに構築された BigDL は、数百または数千のサーバーへ容易にスケーリングすることができます。また、BigDL は、インテル® Xeon® プロセッサー・ベースのサーバーで優れたパフォーマンスを達成できるように、インテル® MKL と並列コンピューティング手法を使用しています。BigDL は、インテル® Xeon® プロセッサー・ベースの Hadoop*/Spark* クラスターにおいて、単純さ、スケーラビリティー、低 TCO などの利点により、業界と開発者コミュニティーによって幅広く採用されています。BigDL の導入に関する詳細なチュートリアルは、AWS* EMR (英語)、Microsoft* Azure* HDInsight (英語)、Microsoft* Data Science Virtual Machines (英語)、Alibaba* Cloud E-MapReduce (英語)、Databricks (英語)、Cray* Urika-XC* Analytics Software (英語)、Cloudera* Data Science Workbench (英語) などを参照してください。

JD.com とインテルのチームは共同で、BigDL と Apache Spark* で SSD と DeepBit モデルを使用して、大規模な画像特徴検出パイプラインを構築しました。このユースケースでは、BigDL はオブジェクト検出や分類などのさまざまなディープラーニング・モデルのサポートを提供しました。また、Caffe、Torch、TensorFlow* の事前に訓練されたモデルを再利用できるようにしました。インテル® Xeon® プロセッサー・ベースの Hadoop* クラスターで大幅なスピードアップを達成するため、アプリケーション・パイプライン全体が最大限に最適化されました。その結果、同じソリューションを 20 枚の K40 GPU カードで実行した場合と比較して、約 3.83 倍高速になりました [30]。

MLSListings* とインテルのチームは共同で、Microsoft* Azure* のインテル® Xeon® プロセッサー上で BigDL を使用して画像の類似性ベースの住宅推奨システムを構築しました。Places365 データセットを使用して GoogLeNet v1 と VGG-16 をチューニングし、画像の類似性の計算に使用される画像埋め込みを生成しました [31]。

Gigaspace* は、BigDL を Gigapsace* の InsightEdge プラットフォームに統合しました。インテル® Xeon® プロセッサー上で BigDL を使用して訓練した自然言語処理 (NLP) モデルにより、ユーザー要求をコールセンターの適切なスペシャリストに自動転送するシステムを構築しました [32]。

まとめ

最新のインテル® Xeon® スケーラブル・プロセッサーとインテル® MKL-DNN の最適化されたディープラーニング関数は、(推論、古典的なマシンラーニング、その他の AI アルゴリズムに加えて) ディープラーニングの訓練ワークロードに十分な計算を提供します。一般的なディープラーニング・フレームワークは、これらの最適化を組込んでおり、シングルサーバーのパフォーマンスを、一部のケースでは 100 倍以上に向上しています。また、分散アルゴリズムに関する最近の進歩により、数百台のサーバーを使用して訓練期間を数週間から数分へ短縮できるようになりました。データ・サイエンティストは、既存の汎用インテル® Xeon® プロセッサー・ベースのクラスターを、これまでのディープラーニングの推論、古典的なディープラーニング、およびビッグデータ・ワークロードに加えて、ディープラーニングの訓練にも使用できます。1 つのインテル® プロセッサーで優れたディープラーニングの訓練パフォーマンスが得られますが、複数の CPU ノードを使用することで数百ノードへ直線的に近いスケーリングを達成して訓練時間をさらに短縮できます。

脚注

1. インテル® Xeon® Platinum プロセッサー、インテル® Xeon® Gold 6xxx/5122 プロセッサーでは、コアごとに 2 つの 512 ビット FMA ユニット (並列に処理) を利用できます。その他のインテル® Xeon® スケーラブル・プロセッサーでは、コアごとに 1 つの FMA ユニットを利用できます。
2. 計算能力は次の式で求められます: インテル® AVX-512 の周波数 * コア数 * コアごとの FMA ユニット数 * FMA ごとに 2 操作 * SIMD ベクトル長 / 数値形式のビット数。インテル® Xeon® Platinum 8180 の場合、インテル® AVX-512 の周波数 * 28 * 2 * 2 * 512 / 32 = (1792 * インテル® AVX-512 の周波数) ピーク TFLOPS になります。各 SKU のインテル® AVX-512 の周波数は、https://www.intel.com/content/www/us/en/processors/xeon/scalable/xeon-scalable-spec-update.html (英語) にあります。記載されている周波数は、FP64 操作のものです。FP32 の周波数は、記載されている周波数よりもやや高い可能性があります。インテル® AVX-512 のインテル® ターボ・ブースト・テクノロジー利用時の最大周波数は、非常に大きな FLOPS のワークロードの実行では達成できない可能性があります。
3. 2 つの FMA ユニットを搭載したプロセッサーの 512 ビット・レジスター・ポート・スキームで実行する場合、ポート 0 FMA のレイテンシーは 4 サイクル、ポート 5 FMA のレイテンシーは 6 サイクルになります。バイパスには、-2 (高速なバイパス) ～ +1 サイクルの遅延が伴います。FMA ユニットは、すべての命令ソースが FMA ユニットからの場合、高速なバイパスをサポートします。FP32 でディープラーニング・ワークロードに使用される命令はバイパスをサポートしており、ポート 0 と 5 のレイテンシーはどちらも 4 サイクルです。https://www.isus.jp/technical-document/ にある「インテル® 64 および IA-32 アーキテクチャー最適化リファレンス・マニュアル参考訳」の 15.17 節を参照してください。2 FMA ユニット搭載のインテル® Xeon® スケーラブル・プロセッサーでは、データと重み付けのレイテンシーを隠蔽するため、少なくとも 8 つのレジスターが必要になります。
4. 8K 以下のバッチサイズでは、効果が見られません [23]。8K を超えると、学習率はバッチサイズの増加の平方根に比例して向上するはずです。
5. 研究者達はいくつかの変更を追加しました。それらは、インテルにより最適化された Caffe のメインブランチにコミットされる予定です。

謝辞

データの収集と検証に協力してくれたパフォーマンス・チームのメンバーに感謝します: Deepthi Karkada、Adam Procter、William (Prune) Wickart、Vikram Saletore、Banu Nagasundaram。また、技術的な正確さをチェックし、記事の明晰さと内容の改善に協力してくれた素晴らしいレビューアー達にも感謝します: Mike Pearce、Alexis Crowell、Chase Adams、Eric Dahlen、Akhilesh Kumar、Dheevatsa Mudigere、Mikhail Smorkalov、Wei Li、Dave Hill、R. Vivek Rane、Mike Ferron-Jones、Allyson Klein、Ransford Hyman、Brian Golembiewski、Israel Hirsh。

著者紹介

Andres Rodriguez 博士: インテル コーポレーションの AI 製品グループ (AIPG) のシニア主席エンジニア。インテルの顧客向けに AI ソリューションの設計に取り組み、AI 製品についてインテルで技術的なリーダーシップをとっています。AI 分野で 13 年の経験があり、マシンラーニングの研究でカーネギーメロン大学から博士号を取得しています。マシンラーニングに関して、雑誌、会議、書籍の章で 20 を超える査読論文を発表しています。

Jason Dai: インテル コーポレーションのソフトウェア & サービスグループ (SSG) のシニア主席エンジニア兼ビッグデータのチーフアーキテクト。高度なビッグデータ解析 (分散マシン/ディープラーニングを含む) の開発を指揮し、主要研究機関 (カリフォルニア大学バークレー校の AMPLab など) との共同プロジェクトを担当しています。ビッグデータ、クラウド、分散マシンラーニングの分野で国際的に認知されているエキスパートで、Strata Data Conference Beijing プログラムの共同議長、Apache Spark* プロジェクトのコミッター兼 PMC メンバー、Apache Spark* 上の分散ディープラーニング・フレームワークである BigDL プロジェクトの創始者でもあります。

Frank Zhang: インテル コーポレーションのソフトウェア & サービスグループ (SSG) のインテルにより最適化された Caffe 製品マネージャー。インテルにより最適化された Caffe ディープラーニング・フレームワークの開発、製品リリース、カスタマーサポートを管理しています。NEC、TI、Marvell を含む複数の企業において 20 年以上のソフトウェア開発分野での経験があります。テキサス大学ダラス校から電気工学の修士号を取得しています。

Jiong Gong: インテル コーポレーションのソフトウェア & サービスグループ (SSG) のシニア・ソフトウェア・エンジニア。インテルにより最適化された Caffe のアーキテクチャーを設計しており、シングルノードとマルチノードの両方の IA プラットフォームでパフォーマンスの利点をもたらす最適化に取り組んでいます。システム・ソフトウェアと AI 分野で 10 年以上の経験があります。上海交通大学でコンピューター・サイエンスの修士号を取得しています。AI とマシンラーニングに関する 4 つの米国特許を保有しています。

Chong Yu: インテル コーポレーションのソフトウェア & サービスグループ (SSG) のソフトウェア・エンジニア。インテルにより最適化された Caffe フレームワークの開発と IA プラットフォームでの最適化に取り組んでいます。インテル・フェローシップを経て、5 年前にインテルに入社しました。復旦大学から情報科学技術の修士号を取得しています。20 の論文を雑誌で発表しており、2 つの中国特許を保有しています。人工知能、ロボット工学、3 次元再構成、リモートセンシング、ステガノグラフィーなどの分野の研究に取り組んでいます。

文献目録

1. **Y. You, et al., "ImageNet training in minutes." Nov. 2017. https://arxiv.org/pdf/1709.05011.pdf (英語)**

2. K. He, at al., "Deep residual learning for image recognition." NIPS. Dec. 2015. https://arxiv.org/abs/1512.03385 (英語)
3. N. Rao, "Comparing dense compute platforms for AI." June 2017. https://www.intelnervana.com/comparing-dense-compute-platforms-ai/ (英語)
4. D. Mulnix, "Intel® Xeon® processor scalable family technical overview." Sept. 2017. https://software.intel.com/en-us/articles/intel-xeon-processor-scalable-family-technical-overview (英語)
5. A. Rodriguez and J. Riverio, "Deep learning at cloud scale: improving video discoverability by scaling up Caffe on AWS." AWS Re-Invent, Nov. 2016. https://www.slideshare.net/AmazonWebServices/aws-reinvent-2016-deep-learning-at-cloud-scale-improving-video-discoverability-by-scaling-up-caffe-on-aws-mac205 (英語)
6. A. Rodriguez and N. Sundaram, "Intel and Facebook collaborate to boost Caffe2 performance on Intel CPUs." Apr. 2017. https://software.intel.com/en-us/blogs/2017/04/18/intel-and-facebook-collaborate-to-boost-caffe2-performance-on-intel-cpu-s (英語)
7. E Ould-Ahmed-Vall, et al., "TensorFlow optimizations on modern Intel® architecture." Aug. 2017. https://software.intel.com/en-us/articles/tensorflow-optimizations-on-modern-intel-architecture (英語)
8. インテル® MKL https://www.isus.jp/intel-mkl/
9. Intel® MKL-DNN https://github.com/01org/mkl-dnn (英語)
10. V. Pirogov and G. Federov, "Introducing DNN primitives in Intel® Math Kernel Library." Oct. 2016. https://www.isus.jp/machine-learning/introducing-dnn-primitives-in-intelr-mkl/
11. https://github.com/01org/mkl-dnn/blob/master/include/mkldnn.hpp (英語)
12. S. Ruder, "An overview of gradient descent optimization algorithms." June 2017. http://ruder.io/optimizing-gradient-descent/ (英語)
13. N. Keskar, et al., "On large-batch training for deep learning: generalization gap and sharp minima." Feb. 2017. https://arxiv.org/abs/1609.04836 (英語)
14. T. Kurth, et al., "Deep learning at 15PF: supervised and semi-supervised classification for scientific data." Aug. 2017. https://arxiv.org/pdf/1708.05256.pdf (英語)
15. A. Gibiansky, "Bringing HPC techniques to deep learning." Feb. 2017. http://research.baidu.com/bringing-hpc-techniques-deep-learning/ (英語)
16. J. Dean, "Large scale deep learning." Nov. 2014. https://www.slideshare.net/hustwj/cikm-keynotenov2014 (英語)
17. A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks." Apr. 2014. https://arxiv.org/pdf/1404.5997.pdf (英語)
18. F. Iandola, et al., "FireCaffe: near-linear acceleration of deep neural network training on compute clusters." Jan. 2016. https://arxiv.org/abs/1511.00175 (英語)
19. D. Das, et al., "Distributed deep learning using synchronous stochastic gradient descent." Feb. 2016. https://arxiv.org/abs/1602.06709 (英語)
20. Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks." https://arxiv.org/pdf/1708.03888.pdf (英語)
21. P. Goyal, et al., "Accurate, large minibatch SGD: training ImageNet in 1 hour." June 2017. https://arxiv.org/abs/1706.02677 (英語)
22. V. Codreanu, D. Podareanu and V. Saletore, "Achieving deep learning Training in less than 40 minutes on ImageNet-1K & best accuracy and training time on ImageNet-22K & Places-365 with scale-out Intel® Xeon®/Xeon Phi™ architectures." Sep. 2017.

https://blog.surf.nl/en/imagenet-1k-training-on-intel-xeon-phi-in-less-than-40-minutes/ (英語)
23. V. Codreanu, D. Podareanu and V. Saletore, "Scale out for large minibatch SGD: Residual network training on ImageNet-1K with improved accuracy and reduced time to train." Nov. 2017. https://arxiv.org/abs/1711.04291 (英語)
24. BigDL: Distributed deep learning library for Apache Spark.7 https://github.com/intel-analytics/BigDL (英語)
25. Z. Yiqiang and J. Gong, "Manufacturing package fault detection using deep learning." Aug. 2017. https://software.intel.com/en-us/articles/manufacturing-package-fault-detection-using-deep-learning (英語)
26. K. Hazelwood, et al., "Applied machine learning at Facebook: A datacenter infrastructure perspective." International Symposium on High-Performance Computer Architecture (HPCA). Feb. 2018. https://research.fb.com/publications/applied-machine-learning-at-facebook-a-datacenter-infrastructure-perspective/ (英語)
27. I. Adamski, "Solving Atari games with distributed reinforcement learning." Oct. 2017. https://blog.deepsense.ai/solving-atari-games-with-distributed-reinforcement-learning/ (英語)
28. A. Karpathy, et al., "Evolution strategies as a scalable alternative to reinforcement learning." Mar. 2017. https://blog.openai.com/evolution-strategies/ (英語)
29. "Honeywell launches UAV industrial inspection service, teams with Intel on innovative offering." Sept. 2017. https://www.honeywell.com/newsroom/pressreleases/2017/09/honeywell-launches-uav-industrial-inspection-service-teams-with-intel-on-innovative-offering (英語)
30. J. Dai, X. Liu and Z. Wang, "Building Large-Scale Image Feature Extraction with BigDL at JD.com." Oct. 2017. https://software.intel.com/en-us/articles/building-large-scale-image-feature-extraction-with-bigdl-at-jdcom (英語)
31. J. Dai, Y. Yuhao and J. Wang, "Using BigDL to build image similarity-based house recommendations." Nov. 2017. https://software.intel.com/en-us/articles/using-bigdl-to-build-image-similarity-based-house-recommendations (英語)
32. R. Shah, "Insights into in-memory computing and real-time analytics." The official GigaSpaces technologies blog. Sep. 2017. https://blog.gigaspaces.com/gigaspaces-to-demo-with-intel-at-strata-data-conference-and-microsoft-ignite/ (英語)

システム構成

a. **STREAM: 1 ノード、2x インテル®** Xeon® Platinum 8180 **プロセッサー、Neon City CRB、合計メモリー 384GB、Red Hat* Enterprise Linux* 7.2 カーネル 3.10.0-327、STREAM インテル®** AVX 512 **バイナリー。データの出典: 要求番号: 2500、ベンチマーク: STREAM - Triad、スコア: 199 (値が大きいほうが良い)**

b. **SGEMM: 1 ノード、1x インテル®** Xeon® Platinum 8180 **プロセッサー、38.5M キャッシュ、2.50GHz、28 コア、56 論理プロセッサー、Lightning Ridge SKX (開発コード名) プラットフォーム、12 スロット、合計メモリー 384GB (メモリー構成: 12 スロット、32GB、2666MT/s、DDR4 RDIMM メモリー)、インテル® ハイパースレッディング・テクノロジー無効、インテル® ターボ・ブースト・テクノロジー有効、Red Hat* Enterprise Linux* 7.3 カーネル 3.10.0-514.el7.x86_64、インテル® コンパイラー 17 Update 2、BIOS バージョン SE5C620.86B.01.00.0412.020920172159。データの出典: 要求番号: 2594、ベンチマーク: SGEMM、スコア: 3570.48 (値が大きいほうが良い)**

c. SGEMM、IGEMM: SKX (開発コード名): インテル® Xeon® Platinum 8180 プロセッサー、ソケットごとに 28 コア、2 ソケット (テストには 1 ソケットのみ使用)、TDP 周波数 2.50GHz、Wolf Pass プラットフォーム、メモリー 384GB、メモリー速度 2666 MHz。BIOS バージョン SE5C620.86B.01.00.0412.020920172159、Ubuntu* 16.04。

BDX (開発コード名): インテル® Xeon® プロセッサー E5-2699 v4、ソケットごとに 22 コア、2 ソケット (テストには 1 ソケットのみ使用)、TDP 周波数 2.20GHz、Cottonwood Pass プラットフォーム、メモリー 64GB、メモリー速度 2400MHz。BIOS バージョン GRRFSDP1.86B.0271.R00.1510301446、Red Hat* Enterprise Linux* 7.0。

d. 2x インテル® Xeon® Platinum 8180 プロセッサー、2.50GHz、28 コア、インテル® ハイパースレッディング・テクノロジー無効、インテル® ターボ・ブースト・テクノロジー無効、intel_pstate ドライバーによりスケーリング・ガバナーは "performance" に設定、384GB DDR4-2666 ECC RAM。インテル® Solid-State Drive DC S3700 シリーズ (800GB、2.5 インチ SATA 6Gb/s、25nm、MLC)。CentOS* 7.3.1611 (Core) カーネル 3.10.0-514.10.2.el7.x86_64。

パフォーマンス測定に使用した設定: KMP_AFFINITY='granularity=fine, compact'、OMP_NUM_THREADS=56、cpupower frequency-set -d 2.5G -u 3.8G -g performance。

Caffe (http://github.com/intel/caffe/ (英語)) revision f96b759f71b2281835f690af267158b82b150b5c。推論に使用したコマンド: "caffe time --forward_only"、訓練に使用したコマンド: "caffe time"。"ConvNet" トポロジーではダミー・データセットを使用。その他のトポロジーではデータをローカルストレージに格納し、訓練の前にメモリーにキャッシュ。トポロジーの仕様: https://github.com/intel/caffe/tree/master/models/intel_optimized_models (英語) (GoogLeNet、AlexNet、ResNet-50)、https://github.com/intel/caffe/tree/master/models/default_vgg_19 (英語) (VGG-19)、https://github.com/soumith/convnet-benchmarks/tree/master/caffe/imagenet_winners (英語) (ConvNet ベンチマーク。Caffe の新しい prototxt 形式を使用するようにファイルを更新したが機能は同等)。インテル® C++ コンパイラー 17.0.2 20170213、インテル® MKLML 2018.0.20170425。"numactl -l" で Caffe を実行。

TensorFlow* (https://github.com/tensorflow/tensorflow (英語)) commit id 207203253b6f8ea5e938a512798429f91d5b4e7e。ダミーデータを使用して 3 つの ConvNet ベンチマークのパフォーマンスを測定: AlexNet、GoogLeNet v1、VGG (https://github.com/soumith/convnet-benchmarks/tree/master/tensorflow (英語))。GCC 4.8.5、インテル® MKLML 2018.0.20170425、interop 並列処理スレッドの設定: AlexNet と VGG は 1、GoogLeNet は 2、intraop 並列処理スレッドの設定: 56、NCHW データ形式を使用、KMP_BLOCKTIME の設定: GoogLeNet と VGG は 1、AlexNet は 30。推論に使用したインテル® MKL 2017 オプション: --caffe time -forward_only -engine、訓練に使用したインテル® MKL 2017 オプション: --forward_backward_only。

MXNet (https://github.com/dmlc/mxnet/ (英語)) revision 5efd91a71f36fea483e882b0358c8d46b5a7aa20。ダミーデータを使用。推論には "benchmark_score.py"、訓練にはバックワード・プロパゲーションも実行可能な変更した benchmark_score.py を使用。トポロジーの仕様:

https://github.com/dmlc/mxnet/tree/master/example/image-classification/symbols (英語)。GCC 4.8.5、インテル® MKLML 2018.0.20170425。

インテル® Neon™ Framework: 社内バージョン。ダミーデータを使用。main.py スクリプトを mkl モードで使用。インテル® C++ コンパイラー 17.0.3 20170404、インテル® MKLML 2018.0.20170425。インテル® Neon™ Framework の最新バージョンの結果: https://www.intelnervana.com/neon-v2-3-0-significant-performance-boost-for-deep-speech-2-and-vgg-models/ (英語)。

e. 2x インテル® Xeon® Gold 6148 プロセッサー、2.40GHz、20 コア、インテル® ハイパースレッディング・テクノロジー無効、インテル® ターボ・ブースト・テクノロジー無効、intel_pstate ドライバーによりスケーリング・ガバナーは "performance" に設定、384GB DDR4-2666 ECC RAM。インテル® Solid-State Drive DC S3700 シリーズ (800GB、2.5 インチ SATA 6Gb/s、25nm、MLC)。CentOS* 7.3.1611 (Core) カーネル 3.10.0-514.10.2.el7.x86_64。

パフォーマンス測定に使用した設定: KMP_AFFINITY='granularity=fine, compact'、OMP_NUM_THREADS=40、cpupower frequency-set -d 2.4G -u 3.7G -g performance。

Caffe (http://github.com/intel/caffe/ (英語)) revision f96b759f71b2281835f690af267158b82b150b5c。推論に使用したコマンド: "caffe time --forward_only"、訓練に使用したコマンド: "caffe time"。"ConvNet" トポロジーではダミー・データセットを使用。その他のトポロジーではデータをローカルストレージに格納し、訓練の前にメモリーにキャッシュ。トポロジーの仕様: https://github.com/intel/caffe/tree/master/models/intel_optimized_models (英語) (GoogLeNet、AlexNet、ResNet-50)、https://github.com/intel/caffe/tree/master/models/default_vgg_19 (英語) (VGG-19)、https://github.com/soumith/convnet-benchmarks/tree/master/caffe/imagenet_winners (英語) (ConvNet ベンチマーク。Caffe の新しい prototxt 形式を使用するようにファイルを更新したが機能は同等)。インテル® C++ コンパイラー 17.0.2 20170213、インテル® MKLML 2018.0.20170425。"numactl -l" で Caffe を実行。

TensorFlow* (https://github.com/tensorflow/tensorflow (英語)) commit id 207203253b6f8ea5e938a512798429f91d5b4e7e。ダミーデータを使用して 3 つの ConvNet ベンチマークのパフォーマンスを測定: AlexNet、GoogLeNet v1、VGG (https://github.com/soumith/convnet-benchmarks/tree/master/tensorflow (英語))。GCC 4.8.5、インテル® MKLML 2018.0.20170425、interop 並列処理スレッドの設定: AlexNet と VGG は 1、GoogLeNet は 2、intraop 並列処理スレッドの設定: 40、NCHW データ形式を使用、KMP_BLOCKTIME の設定: GoogLeNet と VGG は 1、AlexNet は 30。推論に使用したインテル® MKL 2017 オプション: --caffe time -forward_only -engine、訓練に使用したインテル® MKL 2017 オプション: --forward_backward_only。

MXNet (https://github.com/dmlc/mxnet/ (英語)) revision 5efd91a71f36fea483e882b0358c8d46b5a7aa20。ダミーデータを使用。推論には "benchmark_score.py"、訓練にはバックワード・プロパゲーションも実行可能な変更した benchmark_score.py を使用。トポロジーの仕様: https://github.com/dmlc/mxnet/tree/master/example/image-classification/symbols (英語)。GCC 4.8.5、インテル® MKLML 2018.0.20170425。

インテル® Neon™ Framework: 社内バージョン。ダミーデータを使用。main.py スクリプトを mkl モードで使用。インテル® C++ コンパイラー 17.0.3 20170404、インテル® MKLML 2018.0.20170425。インテル® Neon™ Framework の最新バージョンの結果: https://www.intelnervana.com/neon-v2-3-0-significant-performance-boost-for-deep-speech-2-and-vgg-models/ (英語)。

f. 2x インテル® Xeon® プロセッサー E5-2699 v4、2.20GHz、22 コア、インテル® ハイパースレッディング・テクノロジー有効、インテル® ターボ・ブースト・テクノロジー無効、acpi-cpufreq ドライバーによりスケーリング・ガバナーは "performance" に設定、256GB DDR4-2133 ECC RAM。インテル® Solid-State Drive DC S3500 シリーズ (480GB、2.5 インチ SATA 6Gb/s、20nm、MLC)。CentOS* 7.3.1611 (Core) カーネル 3.10.0-514.10.2.el7.x86_64。

パフォーマンス測定に使用した設定: KMP_AFFINITY='granularity=fine, compact,1,0'、OMP_NUM_THREADS=44、cpupower frequency-set -d 2.2G -u 2.2G -g performance。

パフォーマンス測定に使用した設定: KMP_AFFINITY='granularity=fine, compact'、OMP_NUM_THREADS=56、cpupower frequency-set -d 2.5G -u 3.8G -g performance。推論に使用したコマンド: "caffe time --forward_only"、訓練に使用したコマンド: "caffe time"。"ConvNet" トポロジーではダミー・データセットを使用。その他のトポロジーではデータをローカルストレージに格納し、訓練の前にメモリーにキャッシュ。トポロジーの仕様: https://github.com/intel/caffe/tree/master/models/intel_optimized_models (英語) (GoogLeNet、AlexNet、ResNet-50)、https://github.com/intel/caffe/tree/master/models/default_vgg_19 (英語) (VGG-19)、https://github.com/soumith/convnet-benchmarks/tree/master/caffe/imagenet_winners (英語) (ConvNet ベンチマーク。Caffe の新しい prototxt 形式を使用するようにファイルを更新したが機能は同等)。GCC 4.8.5、インテル® MKLML 2017.0.2.20170110。

TensorFlow* (https://github.com/tensorflow/tensorflow (英語)) commit id 207203253b6f8ea5e938a512798429f91d5b4e7e。ダミーデータを使用して 3 つの ConvNet ベンチマークのパフォーマンスを測定: AlexNet、GoogLeNet v1、VGG (https://github.com/soumith/convnet-benchmarks/tree/master/tensorflow (英語))。GCC 4.8.5、インテル® MKLML 2018.0.20170425、interop 並列処理スレッドの設定: AlexNet と VGG は 1、GoogLeNet は 2、intraop 並列処理スレッドの設定: 44、NCHW データ形式を使用、KMP_BLOCKTIME の設定: GoogLeNet と VGG は 1、AlexNet は 30。推論に使用したインテル® MKL 2017 オプション: --caffe time -forward_only -engine、訓練に使用したインテル® MKL 2017 オプション: --forward_backward_only。

MXNet (https://github.com/dmlc/mxnet/ (英語)) revision e9f281a27584cdb78db8ce6b66e648b3dbc10d37。ダミーデータを使用。推論には "benchmark_score.py"、訓練にはバックワード・プロパゲーションも実行可能な変更した benchmark_score.py を使用。トポロジーの仕様: https://github.com/dmlc/mxnet/tree/master/example/image-classification/symbols (英語)。GCC 4.8.5、インテル® MKLML 2017.0.2.20170110。

インテル® Neon™ Framework: 社内バージョン。ダミーデータを使用。main.py スクリプトを mkl モードで使用。インテル® C++ コンパイラー 17.0.3 20170404、インテル® MKLML 2018.0.20170425。

インテル® Neon™ Framework の最新バージョンの結果: https://www.intelnervana.com/neon-v2-3-0-significant-performance-boost-for-deep-speech-2-and-vgg-models/ (英語)。

g. 2x インテル® Xeon® プロセッサー E5-2699 v3、2.30GHz、18 コア、インテル® ハイパースレッディング・テクノロジー有効、インテル® ターボ・ブースト・テクノロジー無効、intel_pstate ドライバーによりスケーリング・ガバナーは "performance" に設定、256GB DDR4-2133 ECC RAM。Seagate* Enterprise ST2000NX0253 (2TB 、2.5 インチ、内蔵ハードディスク)。CentOS* 7.3.1611 (Core) カーネル 3.10.0-514.el7.x86_64。

パフォーマンス測定に使用した設定: KMP_AFFINITY='granularity=fine, compact,1,0'、OMP_NUM_THREADS=36、cpupower frequency-set -d 2.3G -u 2.3G -g performance。

インテルの Caffe: (http://github.com/intel/caffe/ (英語)) revision b0ef3236528a2c7d2988f249d347d5fdae831236.推論に使用したコマンド: "caffe time --forward_only"、訓練に使用したコマンド: "caffe time"。"ConvNet" トポロジーではダミー・データセットを使用。その他のトポロジーではデータをローカルストレージに格納し、訓練の前にメモリーにキャッシュ。トポロジーの仕様: https://github.com/intel/caffe/tree/master/models/intel_optimized_models (英語) (GoogLeNet、AlexNet、ResNet-50)、https://github.com/intel/caffe/tree/master/models/default_vgg_19 (英語) (VGG-19)、https://github.com/soumith/convnet-benchmarks/tree/master/caffe/imagenet_winners (英語) (ConvNet ベンチマーク。Caffe の新しい prototxt 形式を使用するようにファイルを更新したが機能は同等)。GCC 4.8.5、インテル® MKLML 2017.0.2.20170110。

BVLC-Caffe (https://github.com/BVLC/caffe (英語)) revision 91b09280f5233cafc62954c98ce8bc4c204e7475 (commit date 5/14/2017)。推論と訓練に使用したコマンド: "caffe time"。"ConvNet" トポロジーではダミー・データセットを使用。その他のトポロジーではデータをローカルストレージに格納し、訓練の前にメモリーにキャッシュ。BLAS: atlas 3.10.1。

h. 2x インテル® Xeon® プロセッサー E5-2697 v2、2.70GHz、12 コア、インテル® ハイパースレッディング・テクノロジー有効、インテル® ターボ・ブースト・テクノロジー有効、intel_pstate ドライバーによりスケーリング・ガバナーは "performance" に設定、256GB DDR3-1600 ECC RAM。インテル® Solid-State Drive DC S3500 シリーズ (240GB、2.5 インチ SATA 6Gb/s、25nm、MLC)。CentOS* 7.3.1611 (Core) カーネル 3.10.0-514.21.1.el7.x86_64。

パフォーマンス測定に使用した設定: KMP_AFFINITY='granularity=fine, compact,1,0'、OMP_NUM_THREADS=24、cpupower frequency-set -d 2.7G -u 3.5G -g performance。

Caffe (http://github.com/intel/caffe/ (英語)) revision b0ef3236528a2c7d2988f249d347d5fdae831236。推論に使用したコマンド: "caffe time --forward_only"、訓練に使用したコマンド: "caffe time"。"ConvNet" トポロジーではダミー・データセットを使用。その他のトポロジーではデータをローカルストレージに格納し、訓練の前にメモリーにキャッシュ。トポロジーの仕様: https://github.com/intel/caffe/tree/master/models/intel_optimized_models (英語) (GoogLeNet、AlexNet、ResNet-50)、https://github.com/intel/caffe/tree/master/models/default_vgg_19 (英語) (VGG-19)、https://github.com/soumith/convnet-benchmarks/tree/master/caffe/imagenet_winners (英語)

(ConvNet ベンチマーク。Caffe の新しい prototxt 形式を使用するようにファイルを更新したが機能は同等)。GCC 4.8.5、インテル® MKLML 2017.0.2.20170110。

i. 2x インテル® Xeon® Gold 6148 プロセッサー、2.40GHz、20 コア、インテル® ハイパースレッディング・テクノロジー有効、192GB DDR4-2666 ECC RAM。インテル® Solid-State Drive DC S3700 シリーズ (800GB、2.5 インチ SATA 6Gb/s、25nm、MLC)。CentOS* 7.3.1611 (Core) カーネル 3.10.0-514.21.1.el7.x86_64。パフォーマンス測定に使用した設定: KMP_AFFINITY='granularity=fine, compact'、OMP_NUM_THREADS=40。インテル® イーサネット・コネクション X722、10GBASE-T (rev 03)。ディープラーニング・フレームワーク: インテルの Caffe (社内バージョン)。トポロジーバージョン: https://github.com/intel/caffe/tree/master/models/intel_optimized_models (英語) (GoogLeNet、AlexNet、ResNet-50)、https://github.com/intel/caffe/tree/master/models/default_vgg_19 (英語) (VGG-19)。データセット・バージョン: ImageNet* ILSVRC 2012、256x256 にサイズ変更した JPEG。インテル® C++ コンパイラー 2017.4.196、インテル® MKLML 2018.0.1.20171007。インテル® MKL-DNN aab753280e83137ba955f8f19d72cb6aaba545ef。バッチサイズ: ResNet-50=128 x ノード数。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。