ホワイトペーパー

インテル® Xeon® プロセッサーとインテル® Xeon Phi™ プロセッサーにおけるディープラーニング訓練と推論パフォーマンスの向上

Vikram Saletore Ph.D.、Deepthi Karkada、Vamsi Sripathi、
Kushal Datta Ph.D.、Ananth Sankaranarayanan

目次

このホワイトペーパーでは、フレームワークのコードを変更することなく、インテル® Xeon® プロセッサーおよびインテル® Xeon Phi™ プロセッサー上でオープンソースの TensorFlow* と Caffe のディープラーニング訓練/推論のパフォーマンスを (最大 2 倍/最大 2.7 倍) 向上する方法を紹介します。システムレベルの最適化により、画像認識畳み込みニューラル・ネットワーク (CNN) ワークロードのスループットの向上と訓練時間の削減を実現します。

1. はじめに

インテル® Xeon® プロセッサーおよびインテル® Xeon Phi™ プロセッサーは、ディープラーニング・アプリケーションやハイパフォーマンス・コンピューティング・アプリケーションで広く利用されています。インテルのソフトウェア・チームは、インテル® プラットフォーム上で最適なパフォーマンスが得られるように、TensorFlow*、Caffe、MxNet のような一般的なディープラーニング・フレームワークの訓練および推論ワークフローを最適化しました。インテルと Google* は、インテル® マス・カーネル・ライブラリー (インテル® MKL) およびディープ・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) を利用することにより、TensorFlow* のパフォーマンスを大幅に向上しました。同様に、インテル® Distribution of Caffe も、インテル® Xeon® プロセッサーおよびインテル® Xeon Phi™ プロセッサー上でパフォーマンスが大幅に向上するように最適化されています。

ResNet-50、GoogLeNet-v1、Inception-3、およびその他の畳み込みニューラル・ネットワーク (CNN) の訓練では、2 次元の畳み込み、行列乗算、ReLU 活性化、最大プーリングやソフトマックスのような計算負荷が高い多くの関数を実行して、膨大な反復計算を行います。これらの関数カーネルは、インテル® プラットフォーム向けに高度に最適化されたインテル® MKL やインテル® MKL-DNN のようなライブラリーにマップされます。CNN アプリケーションのパフォーマンスを測定したところ、インテルにより最適化されたディープラーニング・フレームワークがマルチスレッドに対応している場合でも、CNN の実行中に CPU コアが十分に活用されていないことが分かりました。

フレームワークではユーザーが設定可能な構成パラメーターが提供されていますが、最適なパフォーマンスを達成するにはそれだけでは不十分です。例えば、TensorFlow* は *intra-op* と *inter-op* の並列処理を使用します。*intra-op* は指定された操作でカーネルを並列化するスレッドプールのサイズを制御し、*inter-op* は操作を並列に実行するスレッドプールのサイズを制御します。

しかし、これらのユーザーレベルの設定では、マルチソケットのインテル® Xeon® プロセッサー・ベースのプラットフォームの NUMA 構成について十分なマイクロアーキテクチャーの情報が提供されません。

また、CPU ソケットや NUMA 構成を考慮しないで (スレッドプールのように) 単純なスレッド・アフィニティーを設定しても最適なパフォーマンスは得られません。実際に、ソケット 0 のコアがソケット 1 のメモリーバンクから頻繁にキャッシュラインにアクセスしてインテル® ウルトラ・パス・インターコネクト (インテル® UPI) の帯域幅の負荷が増加すると、スループットは大幅に低下します。この状況は、4、8、および 16 ソケットのシステムでソケット数が多くなるとともに悪化します。CNN ワークロードで最高のパフォーマンスを得るには、ユーザーがフレームワーク固有の構成パラメーターに加えてシステムレベルの最適化を考慮する必要があります。

2. ディープラーニング・パフォーマンスの向上

このセクションでは、マルチソケットのインテル® Xeon® プラットフォームでディープラーニング・ワークロードを適切に実行する次の方法 (最も一般的な手法) を説明します。

- シングルノード、マルチソケット + パラメーター・サーバー (必要な場合) のディープラーニング訓練
- マルチノード、マルチソケット + パラメーター・サーバー (必要な場合) の分散ディープラーニング訓練
- シングルノード、マルチストリームのディープラーニング推論

後のセクションで、これらの最も一般的な手法はインテル® Xeon Phi™ プロセッサー・ベースのプラットフォームにも適していることを示します。

3. 画像認識のパフォーマンス・メトリック

A. 訓練のパフォーマンス・メトリック

特定の反復数と指定されたワーカーあたりのバッチサイズで収束に達する、画像データセットの訓練済みニューラル・ネットワーク・モデルの開発には、パフォーマンス・メトリックとして訓練時間 (TTT) を使用します。指定されたバッチサイズ (*BSize*/ワーカー)、画像スループット (画像/秒) で、チューニングされたハイパーパラメーターおよび指定された *Epochs* 数での収束を仮定します。

1 ワーカーの TTT は次のように指定されます。

$$TTT_1 = \frac{Images_{Total}}{ImageThroughput_1} * Epochs$$

W ワーカーの TTT は次のように指定されます。

$$TTT_w = \frac{Images_{Total}}{ImageThroughput_w} * Epochs$$

B. シングルノードおよびマルチノード訓練のベースライン・パフォーマンス

1 ワーカー/ノード、バッチサイズ *BSize* で訓練を行います。シングルノードのベースライン・パフォーマンスは、*1* ワーカー/ノードの TTT で測定します。マルチノード (*N* ノード) のベースライン・パフォーマンスは、*N* ワーカー、各ノード *1* ワーカーの TTT で測定します。

C. ベースライン推論パフォーマンス

1 入力ストリーム、1 ワーカー/ノードで推論を行います。ベースライン推論パフォーマンスとして、指定されたバッチサイズ *BSize* でシングルノードで達成されたスループット (画像/秒) を測定します。

4. ディープラーニング訓練: マルチソケットのインテル® Xeon® プロセッサー・ベースのシステムで処理を分割

マルチソケットのインテル® Xeon® プラットフォームで CNN ワークロードのコア使用率を向上して最高のパフォーマンスを得るため、プラットフォームのソケットとコアを個別の計算デバイスとして分割し、複数のディープラーニング訓練インスタンスを実行します。用語「インスタンス」は、マルチソケットまたはシングルソケット・システムで同期してローカル・バッチ・サイズの入力データをそれぞれ実行する、ディープラーニング・フレームワークのワーカープロセスを表します。各ワーカープロセスの境界は、システムが使用するコア/スレッド・アフィニティー設定の合計コア/スレッド数のサブセットです。

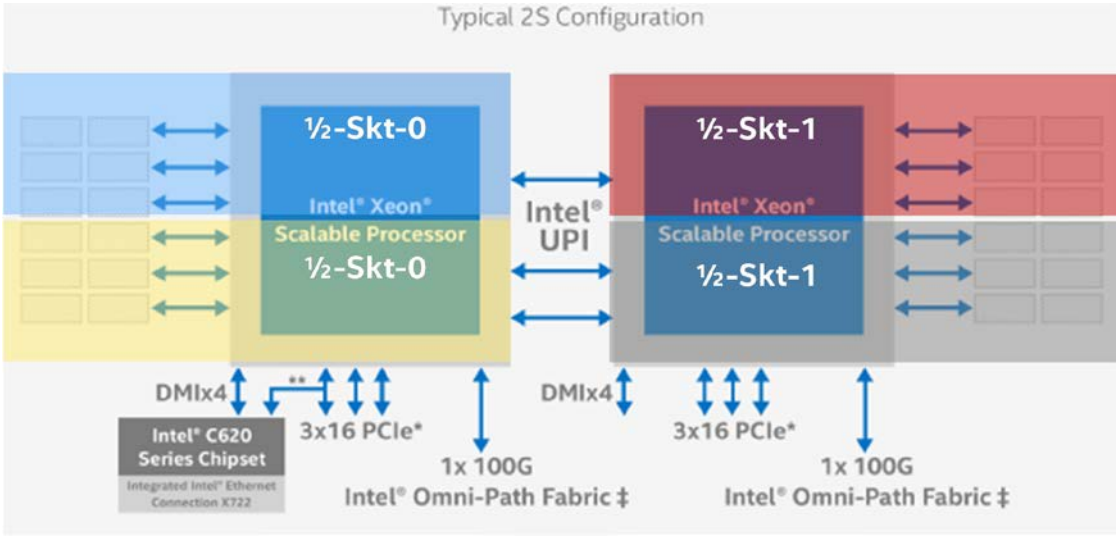

図 1. デュアルソケットのインテル® Xeon® プラットフォームで処理を分割

libnumactl を使用してターゲット NUMA ドメインのメモリー割り当てを制御し、OpenMP* ランタイム・ライブラリーで提供される KMP_AFFINITY 環境変数を使用してターゲットコアに対する OpenMP* スレッドのアフィニティーを設定します。

必要な場合、パラメーター・サーバーは、ホストサーバーで個別のスレッドとしてローカルにスポーンされた勾配または別のサーバーでネットワーク経由でリモートにスポーンされた勾配を集計して処理します。

A. シングルノードおよびマルチノード訓練のマルチワーカーで最適化されたパフォーマンス

このシナリオでは、シングルノードの最適化されたパフォーマンスは、*K* ワーカー/ノード、ワーカーあたりのバッチサイズ *BSize* の TTT で測定します。ノードごとのバッチサイズは *K*BSize* になります。マルチノード (*N* ノード) の最適化されたパフォーマンスは、*K*N* ワーカー、各ノード *K* ワーカーの TTT で測定します。ニューラル・ネットワーク・モデルのハイパーパラメーターは、シングルおよびマルチノードのマルチワーカー向けにチューニングされていると仮定しています。

5. ディープラーニング推論: マルチソケットのインテル® Xeon® プロセッサー・ベースのシステムで処理を分割

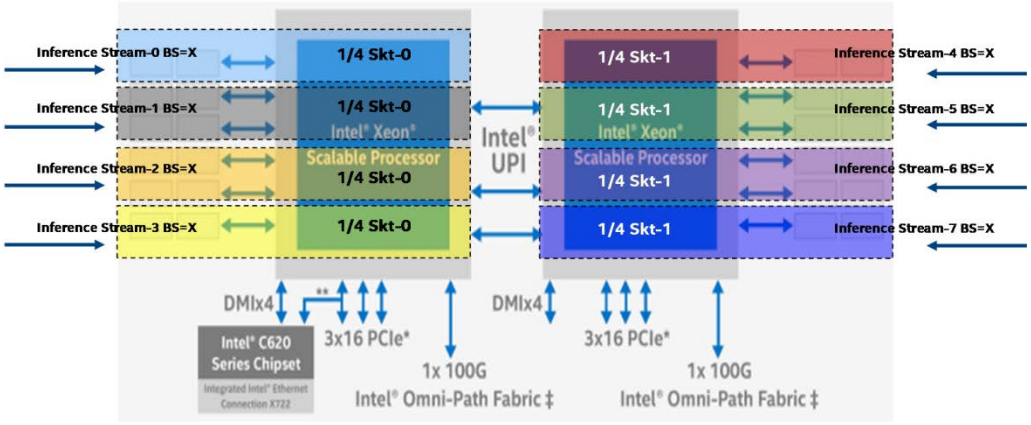

図 2. デュアルソケットのインテル® Xeon® プラットフォームで複数の推論ストリームの処理を分割

ディープラーニング推論にも同様の手法を適用できます。シングルソケットまたはマルチソケット・システムの分割されたコアのセットとメモリーの局所性に対して、複数の独立したディープラーニング推論フレームワーク・インスタンスを作成し、各インスタンスにアフィニティーを設定します。図 2 は、スレッドとメモリーの局所性にアフィニティーが設定された、入力データの個別のストリームを同時に処理する 8 つのフレームワーク・インスタンスの例を示しています。推論バッチサイズとシステムのメモリー容量に応じて、異なるコアにマップされた、多くのフレームとストリームを処理することができます。

A. 最適化された推論パフォーマンス

このシナリオでは、ノードあたり *K* ワーカーです。最適化されたパフォーマンスとして、指定されたバッチサイズ *BSize* で *K* ワーカーにより処理された、*K* 入力ストリームのノードあたりの合計スループット (画像/秒) を測定します。推論の *K* ワーカーのノードごとの合計バッチ数は *K*BSize* になります。

6. TensorFlow* の訓練パフォーマンス

図 3 は、6 つのディープラーニング・ベンチマーク・トポロジーで TensorFlow* 1.4 を使用したディープラーニング訓練パフォーマンス (画像/秒) の向上率です。グラフの 3 つのバーは、デュアルソケットのインテル® Xeon® Platinum 8168 プロセッサーと 10Gb イーサネット・ファブリックで構成された 1、2、4 ノードのクラスターのパフォーマンスの向上を示しています。この図から、コア/スレッド・アフィニティーとメモリーの局所性の最適化を適用すると、シングルノードでも 4 ワーカー/ノードのパフォーマンスが最大 2.1 倍向上することが分かります。

インテル® Xeon® Platinum 8168 プロセッサー: TensorFlow* マルチノード、マルチワーカー/ソケット
訓練: TensorFlow* 1.4、BSize=64、画像データセット、grpc/10Gb イーサネット、各ノードにパラメーター・サーバーを用意

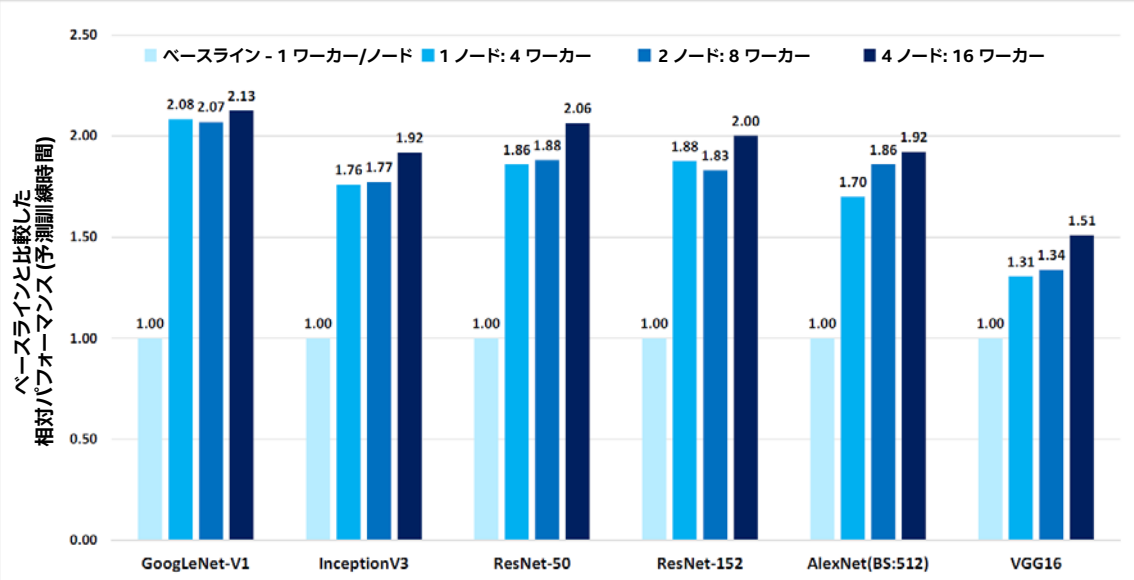

図 3. TensorFlow* 1.4 の訓練パフォーマンス (予測訓練時間) の向上率、コアのアフィニティーとメモリーの局所性を最適化して 4 ワーカー/ノードとベースライン (1 ワーカー/ノード) を比較。プラットフォーム構成: 2x インテル® Xeon® Platinum 8168 プロセッサー、2.70GHz、24 コア、インテル® ハイパースレッディング・テクノロジー有効、インテル® ターボ・ブースト・テクノロジー無効、intel_pstate ドライバーによりスケーリング・ガバナーは "performance" に設定、192GB DDR4-2666 ECC RAM。CentOS* 7.3.1611 (Core)、Linux* カーネル 3.10.0-514.10.2.el7.x86_64。SSD: インテル® SSD DC S3700 シリーズ。ノード接続: 10Gb イーサネット。TensorFlow* 1.4.0、GCC 6.2.0、インテル® MKL-DNN。訓練は SSD に格納された画像データで測定。出典: 2017 年 12 月に実施したインテル社内での測定値。

7. Caffe の訓練パフォーマンス

図 4 は、インテル® Distribution of Caffe の最も一般的な手法を使用して、デュアルソケットのインテル® Xeon® Platinum 8170 プロセッサー・ベースの 1、2、4 ノードのクラスターシステムにおける GoogLeNet-v1 のパフォーマンスをベースラインから最大 1.2 倍向上できることを示しています。GitHub* から入手可能な最新の Caffe は、すでにインテル® CPU 向けに高度に最適化されておりコアを効率的に利用しているため、パフォーマンスの向上は TensorFlow* よりも低くなります。

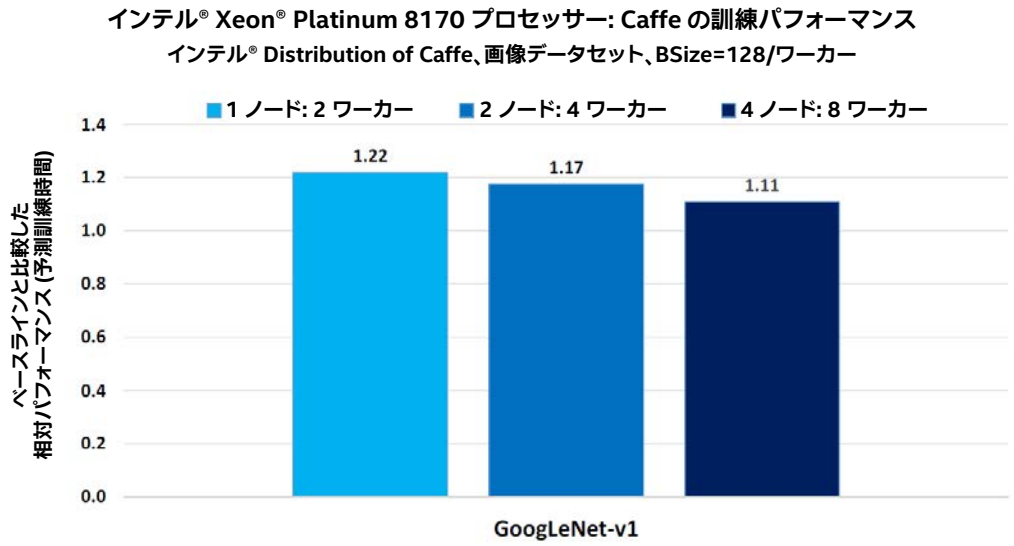

図 4. インテル® Distribution of Caffe の訓練パフォーマンス (予測訓練時間) の向上率、コアのアフィニティーとメモリーの局所性を最適化して 2 Caffe インスタンス/ノードとベースライン (1 インスタンス/ノード) を比較。出典: 2017 年 12 月に実施したインテル社内での測定値。

8. TensorFlow* の推論パフォーマンス

図 5 は、TensorFlow* 1.4 を使用したディープラーニング推論パフォーマンス (画像/秒) の向上率です。グラフの 3 つのバーは、シングルノード、デュアルソケットのインテル® Xeon® Platinum 8168 プロセッサー・ベースのシステムにおけるグローバル・バッチ・サイズ 512 (2 ストリーム、各バッチサイズ 256)、1024、および 2048 のパフォーマンスの向上を示しています。最適化されたテストでは、コアのアフィニティーを 2、4、8 ワーカーに設定し、適切なメモリーの局所性をマップしました。複数のストリームの入力データでは、各ワーカーにより各ストリームが同時に処理されます。例えば、グローバル・バッチ・サイズ 2048 では、それぞれ 256 のバッチサイズを処理する 8 つのストリームを使用します。測定されたパフォーマンス・データから、システムレベルの最適化により、推論のパフォーマンスを最大 2.7 倍に向上できることが分かります。

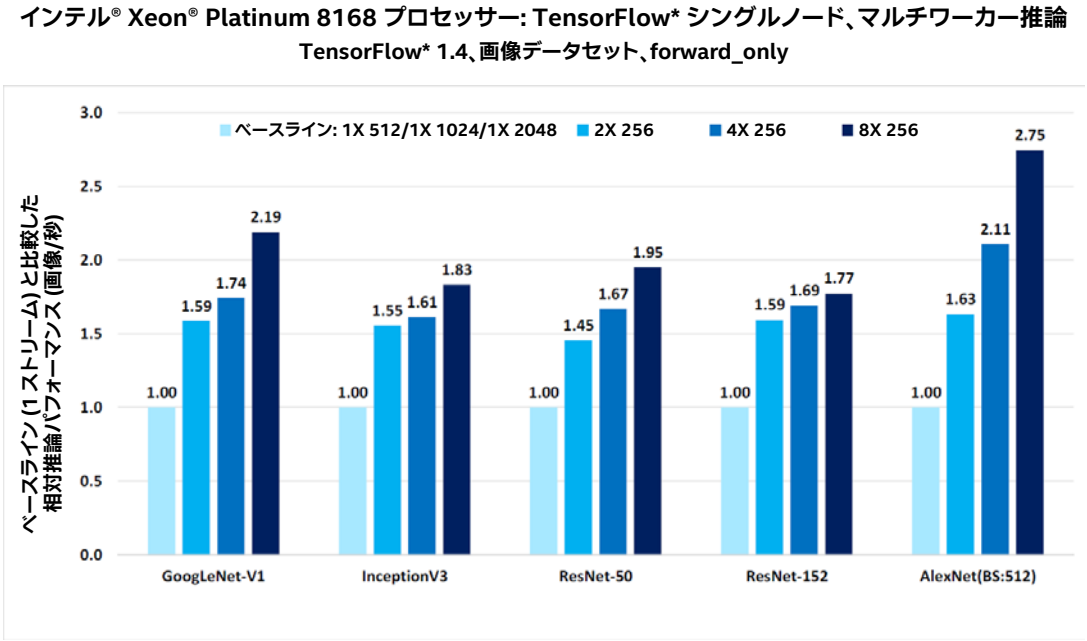

図 5. TensorFlow* の推論パフォーマンス (画像/秒) の向上率、コアのアフィニティーとメモリーの局所性を最適化して 2、4、8 ストリームとベースライン (バッチサイズ等価、1 ストリーム) を比較。プラットフォーム構成: 2x インテル® Xeon® Platinum 8168 プロセッサー、2.70GHz、24 コア、インテル® ハイパースレッディング・テクノロジー有効、インテル® ターボ・ブースト・テクノロジー無効、intel_pstate ドライバーによりスケーリング・ガバナーは "performance" に設定、192GB DDR4-2666 ECC RAM。CentOS* 7.3.1611 (Core)、Linux* カーネル 3.10.0-514.10.2.el7.x86_64。SSD: インテル® SSD DC S3700 シリーズ。TensorFlow* 1.4.0、GCC 6.2.0、インテル® MKL-DNN。推論は --forward_only で測定。出典: 2017 年 12 月に実施したインテル社内での測定値。

9. Caffe の推論パフォーマンス

図 6 は、インテル® Distribution of Caffe を使用したディープラーニング推論パフォーマンス (画像/秒) の向上率です。グラフの 4 つのバーは、シングルノード、デュアルソケットのインテル® Xeon® Platinum 8170 プロセッサー・ベースのシステムにおけるグローバル・バッチ・サイズ 256、512、1024、および 2048 のパフォーマンスの向上を示しています。Caffe は適切に最適化されているにもかかわらず、大きなバッチサイズでは推論パフォーマンスが最大 1.8 倍向上していることが分かります。

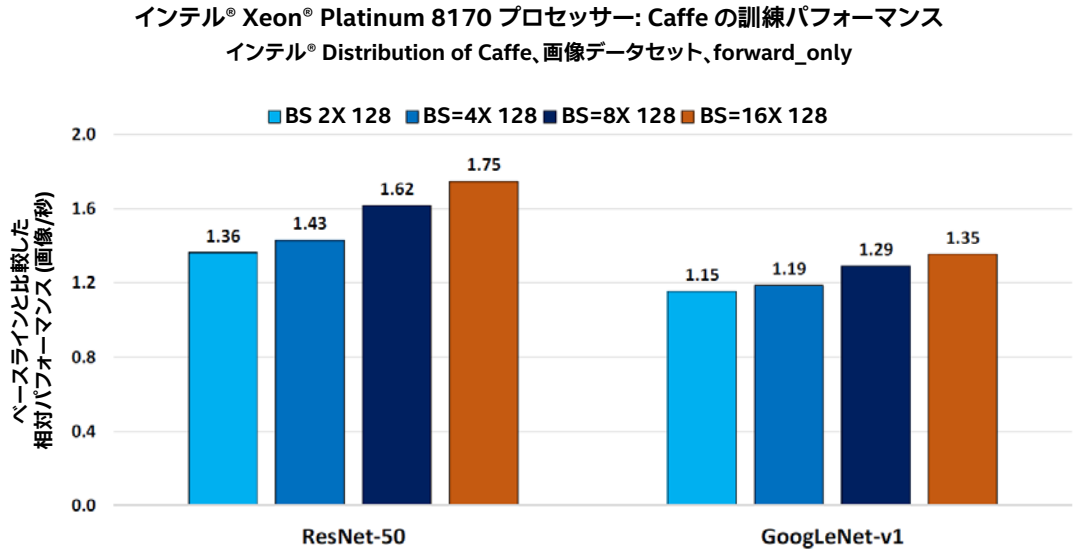

図 6. インテル® Distribution of Caffe の推論パフォーマンス (画像/秒) の向上率、コアのアフィニティーとメモリーの局所性を最適化して 2、4、8 ストリーム/ノードとベースライン (バッチサイズ等価、1 ストリーム/ノード) を比較。出典: 2017 年 12 月に実施したインテル社内での測定値。

10. ディープラーニング訓練: シングルソケットのインテル® Xeon Phi™ プロセッサー・ベースのシステムで処理を分割

インテル® Xeon® プロセッサーの訓練に使用した最適化をシングルソケットのインテル® Xeon Phi™ プロセッサー・ベースのシステムに適用しました。インテル® Xeon Phi™ プロセッサー 7250 には 68 のコアがあり、各コアで 4 つのスレッドを実行できるため、合計で 272 のスレッドを利用できます。図 7 は、フレームワークの 4 つのインスタンスにソケットを分割して、コアのアフィニティーを各インスタンスに設定した場合のシンボリック・ビューを示しています。分散訓練形式で 64 コアで 4 つのインスタンスを実行し (16 コア/インスタンス)、残りの 4 コアは I/O、パラメーター・サーバー (必要な場合) に割り当てます。

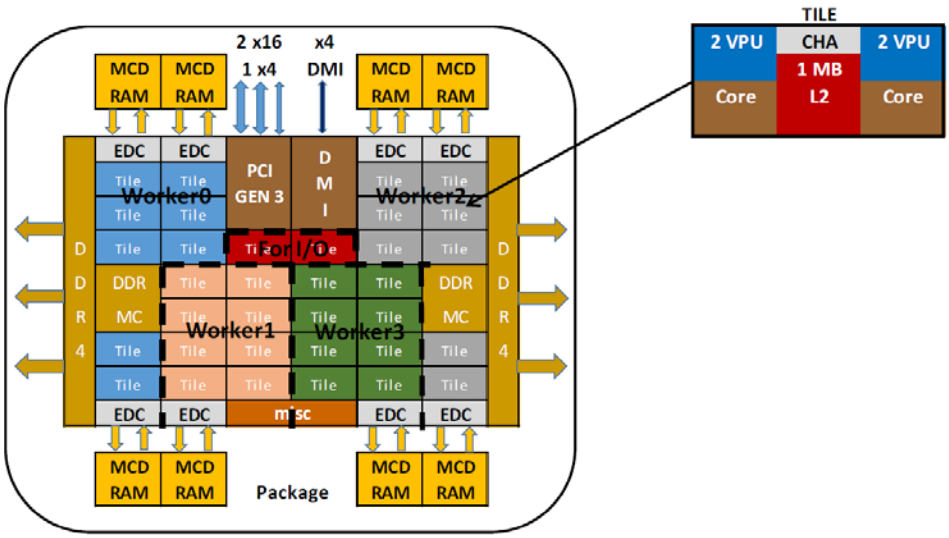

図 7. シングルソケットのインテル® Xeon Phi™ プロセッサー 7250 のシンボリック・サブソケット分割

A. インテル® Xeon Phi™ プロセッサー・ベースのプラットフォームにおける TensorFlow* の訓練パフォーマンス

シングルソケットのインテル® Xeon Phi™ プロセッサー・ベースのシステムでマルチワーカーをサポートするため、システムの MCDRAM をキャッシュモードでブートします。図 8 から、シングルソケットのインテル® Xeon Phi™ プロセッサー 7250 に最適化を適用すると、TensorFlow* 1.3 を使用した ResNet-50 ニューラル・ネットワーク・ベンチマークで 4 ワーカー/ノードのパフォーマンスが最大 1.4 倍向上することが分かります。最適化は、インテル® Omni-Path アーキテクチャー (インテル® OPA) を使用したマルチワーカーおよびマルチノード (1、2、4) の分散訓練にも有効です。

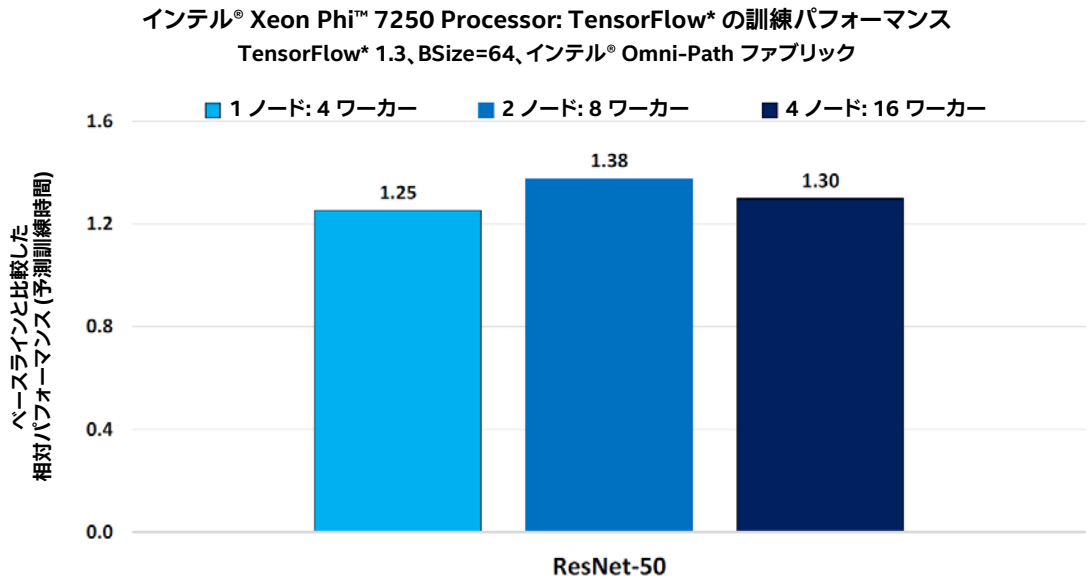

図 8. TensorFlow* の訓練パフォーマンス (予測訓練時間) の向上率、コアのアフィニティーとメモリーの局所性を最適化して 4 ワーカー/ノードとベースライン (1 ワーカー/ノード) を比較。出典: 2017 年 12 月に実施したインテル社内での測定値。

11. プラットフォーム構成

インテル® Xeon® Platinum 8168 プロセッサー
2x インテル® Xeon® Platinum 8168 プロセッサー、2.70GHz、24 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、192GB DDR4-2666 ECC RAM。CentOS* 7.3.1611 (Core)、Linux* カーネル 3.10.0-514.10.2.el7.x86_64。SSD: インテル® SSD DC S3700 シリーズ。ノード接続: 10Gb イーサネット。

インテル® Xeon® Gold 6148 プロセッサー
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、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、192GB DDR4-2666 ECC RAM。CentOS* 7.3.1611 (Core)、Linux* カーネル 3.10.0-514.10.2.el7.x86_64。ノード接続: インテル® Omni-Path ホスト・ファブリック、インテル® Omni-Path ホスト・ファブリック・インターフェイス・ドライバー 10.4.2.0.7。SSD: インテル® SSD DC S3700 シリーズ。

インテル® Xeon® Platinum 8170 プロセッサー
2x インテル® Xeon® Platinum 8170 プロセッサー、2.10GHz、26 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、384GB DDR4-2666 ECC RAM。CentOS* 7.3.1611 (Core)、Linux* カーネル 3.10.0-514.16.1.el7.x86_64。ノード接続: インテル® Omni-Path ホスト・ファブリック、インテル® Omni-Path ホスト・ファブリック・インターフェイス・ドライバー 10.4.2.0.7。SSD: インテル® SSD DC S3700 シリーズ (800GB)。

インテル® Xeon Phi™ プロセッサー 7250
1x インテル® Xeon Phi™ プロセッサー 7250、68 コア、コアあたり 4 ハードウェア・スレッド、1.40GHz、16GB 高速 MCDRAM (Quadrant キャッシュモード)、コアあたり 32KB L1 データキャッシュ、2 コアタイルあたり 1MB L2、96GB DDR4。ノード接続: インテル® Omni-Path ホスト・ファブリック、インテル® Omni-Path ホスト・ファブリック・インターフェイス・ドライバー 10.4.2.0.7。SSD: インテル® SSD DC S3500 シリーズ (480GB)。ソフトウェア: CentOS* 7.3.1611、Linux* カーネル 3.10.0-514.10.2.el7.x86_64、インテル® MPI ライブラリー 2017 Update 4。

12. ディープラーニング・フレームワーク構成

TensorFlow*:
TensorFlow* 1.4: (https://github.com/tensorflow/tensorflow)、TensorFlow* 1.4.0、GCC 6.2.0、インテル® MKL-DNN。TensorFlow* の訓練は SSD に格納されている画像データで測定、推論は -forward_only オプションで測定。

TensorFlow* 1.3: (https://github.com/tensorflow/tensorflow)、TensorFlow* 1.3.0、GCC 6.2.0、インテル® MKL 2017。TensorFlow* の訓練は SSD に格納されている画像データで測定、推論は --forward_only オプションで測定。

インテル® Distribution of Caffe:
Caffe: (http://github.com/intel/caffe/ (英語))、トポロジーの詳細は https://github.com/intel/caffe/tree/master/models/intel_optimized_models (英語) を参照、画像データは訓練および推論の前にメモリーに格納、インテル® C++ コンパイラー 17.0.2 20170213、インテル® MKL 2018.0.20170425。Caffe の訓練は -train オプションで測定、推論は -forward_only オプションで測定。

13. インテル® Xeon® プロセッサーのパフォーマンスの最も一般的な最適化手法と最適化されたディープラーニング・フレームワーク

次の 2 つのセクションでは、例として TensorFlow* および Caffe を使用した最も一般的な手法を説明します。例では、インテル® Xeon® プロセッサー・ベースのプラットフォームおよびインテル® Xeon Phi™ プロセッサー・ベースのプラットフォームを使用しました。

14. TensorFlow* の最も一般的な手法

A. TensorFlow* のビルド方法

インテルにより最適化された TensorFlow* のビルドは、最適化チームにより指定された方法または https://software.intel.com/en-us/articles/intel-optimized-tensorflow-wheel-now-available (英語) の説明に従ってください。

B. 最適化された TensorFlow* のランタイム

GitHub* の TensorFlow* tf_cnn_benchmarks を使用して、最適化されたランタイムを使用した場合のパフォーマンスの向上をテストおよび測定します。

TensorFlow* tf_cnn_benchmarks:

- tf_cnn_benchmarks のコードは https://github.com/tensorflow/benchmarks (英語) から入手できます。
- 入力パイプライン、勾配の最新の API を使用します。
- カスタム CNN トポロジーと容易に統合できます。

C. シングルノード、マルチソケット分散訓練の最も一般的な手法

例 1: 2 ソケットのインテル® Xeon® Gold 6148 プロセッサー・ベースのシステム、マルチソケット (サブソケット)、20 コア/ソケット、ノードごとに 4 つの TensorFlow* ワーカー・インスタンスおよび 1 つのパラメーター・サーバーを使用して 1 ノードの分散訓練を実行。次のように指定して実行します。

PS_HOST: "hostname1"

ps_list: "hostname1:2218"

WK_HOST= hostname2"

workers_list : "hostname2:2223,hostname2:2224,hostname2:2225,hostname2:2226"

worker_env:"export OMP_NUM_THREADS=9; export TF_ADJUST_HUE_FUSED=1; export TF_ADJUST_SATURATION_FUSED=1;"

common_args: "--model resnet50 --batch_size 64 --data_format NCHW --num_batches 100 --distortions=True --mkl=True --local_parameter_device cpu --num_warmup_batches 10 --device cpu --data_dir '/path-to/TF_Records' --data_name imagenet --server_protocol grpc --optimizer rmsprop --ps_hosts $ ps_list --worker_hosts $workers_list --display_every 10"

ps_args: "$common_args --num_intra_threads 4 --num_inter_threads 2"

worker_args: "$common_args --num_intra_threads 9 --num_inter_threads 4"

パラメーター・サーバーの開始方法

ssh $PS_HOST; numactl -l python tf_cnn_benchmarks.py $ps_args --job_name ps --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &

ワーカーの開始方法

ssh $WK_HOST; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[0-9,40-49],explicit,verbose" --job_name worker --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[10-19,50-59],explicit,verbose" --job_name worker --task_index 1 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST; $worker_env;nohup numactl -m 1 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[20-29,60-69],explicit,verbose" --job_name worker --task_index 2 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST; $worker_env;nohup numactl -m 1 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[30-39,70-79],explicit,verbose" --job_name worker --task_index 3 --ps_hosts $ps_list --worker_hosts $workers_list &

$ps_list および $workers_list は、パラメーター・サーバーとワーカーホストのペア (hostname:port) のカンマ区切りのリストです。$ps_args は、パラメーター・サーバーの引数 (--num_inter_threads および --num_intra_threads など) です。$worker_args は、ワーカーの引数 (モデル名、batch_size、data_format、data_dir、server_protocol、num_inter_threads および num_intra_threads など) です。

D. マルチノード、マルチソケット訓練の最も一般的な手法

例 2: 2 ソケットのインテル® Xeon® Gold 6148 プロセッサー・ベースのシステム、マルチソケット (サブソケット)、20 コア/ソケット、ノードごとに 4 つの TensorFlow* ワーカー・インスタンスおよび 1 つのパラメーター・サーバーを使用して 2 ノードの分散訓練を実行。次のように指定して実行します。

PS_HOST_0: "hostname1"

ps_list: "hostname1:2218"

WK_HOST_0=hostname2, WK_HOST_1=hostname3

workers_list:
"hostname2:2223,hostname2:2224,hostname2:2225,hostname2:2226,hostname3:2227,hostname3:2228,hostname3:2229,hostname3:2230"

worker_env:"export OMP_NUM_THREADS=9; export TF_ADJUST_HUE_FUSED=1; export TF_ADJUST_SATURATION_FUSED=1;"

common_args: "--model resnet50 --batch_size 64 --data_format NCHW --num_batches 100 --distortions=True --mkl=True --local_parameter_device cpu --num_warmup_batches 10 --device cpu --data_dir '/path-to/TF_Records' --data_name imagenet --server_protocol grpc --optimizer rmsprop --ps_hosts $ ps_list --worker_hosts $workers_list --display_every 10"

ps_args: "$common_args --num_intra_threads 4 --num_inter_threads 2"

worker_args: "$common_args --num_intra_threads 9 --num_inter_threads 4"

パラメーター・サーバーの開始方法

ssh $PS_HOST_0; numactl -l python tf_cnn_benchmarks.py $ps_args --job_name ps --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &

ノード 0 のワーカーの開始方法

ssh $WK_HOST_0; $worker_env; nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[0-9,40-49],explicit,verbose" --job_name worker --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST_0; $worker_env; nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[10-19,50-59],explicit,verbose" --job_name worker --task_index 1 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST_0; $worker_env; nohup numactl -m 1 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[20-29,60-69],explicit,verbose" --job_name worker --task_index 2 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST_0; $worker_env; nohup numactl -m 1 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[30-39,70-79],explicit,verbose" --job_name worker --task_index 3 --ps_hosts $ps_list --worker_hosts $workers_list &

ノード 1 のワーカーの開始方法

ssh $WK_HOST_1; $worker_env; nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[0-9,40-49],explicit,verbose" --job_name worker --task_index 4 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST_1; $worker_env; nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[10-19,50-59],explicit,verbose" --job_name worker --task_index 5 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST_1; $worker_env; nohup numactl -m 1 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[20-29,60-69],explicit,verbose" --job_name worker --task_index 6 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST_1; $worker_env; nohup numactl -m 1 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[30-39,70-79],explicit,verbose" --job_name worker --task_index 7 --ps_hosts $ps_list --worker_hosts $workers_list &

$ps_list および $workers_list は、パラメーター・サーバーとワーカーホストのペア (hostname:port) のカンマ区切りのリストです。$ps_args は、パラメーター・サーバーの引数 (--num_inter_threads および --num_intra_threads など) です。$worker_args は、ワーカーの引数 (モデル名、batch_size、data_format、data_dir、server_protocol、num_inter_threads および num_intra_threads など) です。

E. インテル® Xeon® プロセッサー・ベースのシステムでのマルチソケット・ディープラーニング推論

例 3: 2 ソケットのインテル® Xeon® Platinum 8170 プロセッサー・ベースのシステム、マルチソケット (サブソケット)、26 コア/ソケット、ノードごとに 8 つの TensorFlow* ワーカー・インスタンスを使用して推論を実行。次のように指定して実行します。

common_args: "--model resnet50 --batch_size 256 --data_format NCHW --num_batches 100 --distortions=True --mkl=True --num_warmup_batches 10 --device cpu --data_dir ~/tensorflow/TF_Records --data_name imagenet --display_every 10"

WK_HOST= hostname"

worker_env:"export OMP_NUM_THREADS=6; export TF_ADJUST_HUE_FUSED=1; export TF_ADJUST_SATURATION_FUSED=1;"

inf_args: "$common_args --num_intra_threads 6 --num_inter_threads 4"

ソケット 0 の 4 つの推論ストリームの開始方法

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 0 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[0-5,52-57],explicit,verbose" &

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 0 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[6-12,58-64],explicit,verbose" &

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 0 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[13-18,65-70],explicit,verbose" &

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 0 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[19-25,71-77],explicit,verbose" &

ソケット 1 の 4 つの推論ストリームの開始方法

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 1 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[26-31,78-83],explicit,verbose" &

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 1 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[32-38,84-90],explicit,verbose" &

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 1 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[39-44,91-96],explicit,verbose" &

ssh $WK_HOST; $worker_env; nohup unbuffer numactl -m 1 python tf_cnn_benchmarks.py --forward_only True $inf_args --kmp_affinity="granularity=thread,proclist=[45-51,96-102],explicit,verbose" &

$inf_args は、推論を実行する TF インスタンスの引数 (モデル名、batch_size、data_format、data_dir、num_inter_threads および num_intra_threads など) です。

F. インテル® Xeon Phi™ プロセッサー 7250 での訓練向けに最適化された TensorFlow* のランタイム

例 4: 1 ソケットのインテル® Xeon Phi™ プロセッサー 7250 ベースのシステム、マルチソケット (サブソケット)、68 コア/ソケット、ノードごとに 4 つの TensorFlow* ワーカー・インスタンスおよび 1 つのパラメーター・サーバーを使用して 1 ノードの分散訓練を実行。次のように指定して実行します。計算に 64 コアを、I/O に残りの 4 コアを使用します。インテル® Xeon Phi™ プロセッサー・ベースのシステムの MCDRAM はキャッシュモードでブートされると仮定しています。

PS_HOST: "hostname1"

ps_list: "hostname1:2218"

WK_HOST= hostname2"

workers_list : "hostname2:2223,hostname2:2224,hostname2:2225,hostname2:2226"

worker_env:"export OMP_NUM_THREADS=15; export TF_ADJUST_HUE_FUSED=1; export TF_ADJUST_SATURATION_FUSED=1;"

common_args: "--model resnet50 --batch_size 64 --data_format NCHW --num_batches 100 --distortions=True --mkl=True --local_parameter_device cpu --num_warmup_batches 10 --device cpu --data_dir '/path-to/TF_Records' --data_name imagenet --server_protocol grpc --optimizer rmsprop --ps_hosts $ ps_list --worker_hosts $workers_list --display_every 10"

ps_args: "$common_args --num_intra_threads 4 --num_inter_threads 2"

worker_args: "$common_args --num_intra_threads 15 --num_inter_threads 4"

パラメーター・サーバーの開始方法

ssh $PS_HOST; numactl -l python tf_cnn_benchmarks.py $ps_args --job_name ps --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &

ワーカーの開始方法

ssh $WK_HOST; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[0-15,68-115],explicit,verbose" --job_name worker --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[16-31,116-163],explicit,verbose" --job_name worker --task_index 1 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[32-47,164-211],explicit,verbose" --job_name worker --task_index 2 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[48-63,212-259],explicit,verbose" --job_name worker --task_index 3 --ps_hosts $ps_list --worker_hosts $workers_list &

$ps_list および $workers_list は、パラメーター・サーバーとワーカーホストのペア (hostname:port) のカンマ区切りのリストです。$ps_args は、パラメーター・サーバーの引数 (--num_inter_threads および --num_intra_threads など) です。$worker_args は、ワーカーの引数 (モデル名、batch_size、data_format、data_dir、server_protocol、num_inter_threads および num_intra_threads など) です。

G. マルチノード、マルチソケット分散訓練の最も一般的な手法

例 5: 2 ソケットのインテル® Xeon Phi™ プロセッサー 7250 ベースのシステム、マルチソケット (サブソケット)、68 コア/ソケット、ノードごとに 4 つの TensorFlow* ワーカー・インスタンスおよび 1 つのパラメーター・サーバーを使用して 2 ノードの分散訓練を実行。次のように指定して実行します。

PS_HOST_0: "hostname1"

ps_list: "hostname1:2218"

WK_HOST_0=hostname2, WK_HOST_1=hostname3

workers_list: "hostname2:2223,hostname2:2224,hostname2:2225,hostname2:2226,hostname3:2227,hostname3:2228,hostname3:2229,hostname3:2230"

worker_env:"export OMP_NUM_THREADS=15; export TF_ADJUST_HUE_FUSED=1; export TF_ADJUST_SATURATION_FUSED=1;"

common_args: "--model resnet50 --batch_size 64 --data_format NCHW --num_batches 100 --distortions=True --mkl=True --local_parameter_device cpu --num_warmup_batches 10 --device cpu --data_dir '/path-to/TF_Records' --data_name imagenet --server_protocol grpc --optimizer rmsprop --ps_hosts $ ps_list --worker_hosts $workers_list --display_every 10"

ps_args: "$common_args --num_intra_threads 4 --num_inter_threads 2"

worker_args: "$common_args --num_intra_threads 15 --num_inter_threads 4"

パラメーター・サーバーの開始方法

```
ssh $PS_HOST; numactl -l python tf_cnn_benchmarks.py $ps_args --job_name ps --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &
```

ノード 0 のワーカーの開始方法

```
ssh $WK_HOST_0; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[0-15,68-115],explicit,verbose" --job_name worker --task_index 0 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST_0; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[16-31,116-163],explicit,verbose" --job_name worker --task_index 1 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST_0; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[32-47,164-211],explicit,verbose" --job_name worker --task_index 2 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST_0; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[48-63,212-259],explicit,verbose" --job_name worker --task_index 3 --ps_hosts $ps_list --worker_hosts $workers_list &
```

ノード 1 のワーカーの開始方法

```
ssh $WK_HOST_1; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[0-15,68-115],explicit,verbose" --job_name worker --task_index 4 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST_1; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[16-31,116-163],explicit,verbose" --job_name worker --task_index 5 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST_1; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[32-47,164-211],explicit,verbose" --job_name worker --task_index 6 --ps_hosts $ps_list --worker_hosts $workers_list &

ssh $WK_HOST_1; $worker_env;nohup numactl -m 0 python tf_cnn_benchmarks.py $worker_args --kmp_affinity="granularity=thread,proclist=[48-63,212-259],explicit,verbose" --job_name worker --task_index 7 --ps_hosts $ps_list --worker_hosts $workers_list &
```

$ps_list および $workers_list は、パラメーター・サーバーとワーカーホストのペア (hostname:port) のカンマ区切りのリストです。$ps_args は、パラメーター・サーバーの引数 (--num_inter_threads および --num_intra_threads など) です。$worker_args は、ワーカーの引数 (モデル名、batch_size、data_format、data_dir、server_protocol、num_inter_threads および num_intra_threads など) です。

15. インテル® Distribution of Caffe の最も一般的な手法

Caffe のビルド方法: https://github.com/intel/caffe (英語) の説明を参照してください。

A. シングル/マルチノード、マルチソケット分散訓練の最も一般的な手法の例

例 6: 2 ソケットのインテル® Xeon® Platinum 8170 プロセッサー・ベースのシステム、マルチソケット (サブソケット)、26 コア/ソケット、ノードごとに 4 つの Caffe ワーカー・インスタンスを使用して訓練を実行。次のように指定して実行します。

```
WK_HOST="hostname"

CORES_PER_NODE=52

P=2 #Processes per node

N=2 #Total number of processes calculated as num_nodes/P

CORES_PER_MPI_PROCESS=$(($CORES_PER_NODE / $P))

OMPTHREADS=$(($CORES_PER_MPI_PROCESS - 2))

export I_MPI_DEBUG=5; mpiexec.hydra -v -l -ppn $P –n $N -f $WK_HOST -genv OMP_NUM_THREADS $OMPTHREADS -genv KMP_AFFINITY 'granularity=fine,compact,1,0' path-to-intelcaffe/build/tools/caffe train -solver $MODELDIR/solver.prototxt -engine MKL2017
```

OMP_NUM_THREADS はプロセスごとに使用される OpenMP* スレッドの数、CAFFEDIR は Caffe のインストール・パス、MODELDIR はモデルの prototxt ファイル (例: googlenet) を含むディレクトリーのパスです。

B. マルチソケット推論の例

例 7: 2 ソケットのインテル® Xeon® Platinum 8170 プロセッサー・ベースのシステム、マルチソケット (サブソケット)、26 コア/ソケット、ノードごとに 8 つの Caffe ワーカー・インスタンスを使用して推論を実行。次のように指定して実行します。

```
OMP_NUM_THREADS=6 KMP_AFFINITY="granularity=thread,proclist=[0-5,52-57],explicit,verbose" numactl -m 0 path-to-intelcaffe/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &

OMP_NUM_THREADS=7 KMP_AFFINITY="granularity=thread,proclist=[6-12,58-64],explicit,verbose" numactl -m 0 path-to-intelcaffe/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &

OMP_NUM_THREADS=6 KMP_AFFINITY="granularity=thread,proclist=[13-18,65-70],explicit,verbose" numactl -m 0 path-to-intelcaffe/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &

OMP_NUM_THREADS=7 KMP_AFFINITY="granularity=thread,proclist=[19-25,71-77],explicit,verbose" numactl -m 0 path-to-intelcaffe/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &

OMP_NUM_THREADS=6 KMP_AFFINITY="granularity=thread,proclist=[26-31,78-83],explicit,verbose" numactl -m 1 $CAFFEDIR/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &

OMP_NUM_THREADS=7 KMP_AFFINITY="granularity=thread,proclist=[32-38,84-90],explicit,verbose" numactl -m 1 $CAFFEDIR/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &

OMP_NUM_THREADS=6 KMP_AFFINITY="granularity=thread,proclist=[39-44,91-96],explicit,verbose" numactl -m 1 $CAFFEDIR/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &

OMP_NUM_THREADS=7 KMP_AFFINITY="granularity=thread,proclist=[45-51,96-102],explicit,verbose" numactl -m 1 $CAFFEDIR/build/tools/caffe time -model $MODELDIR/train_val.prototxt -iterations $iters -engine MKL2017 -forward_only &
```

プラットフォーム構成

インテル® Xeon® Platinum 8168 プロセッサー
2x インテル® Xeon® Platinum 8168 プロセッサー、2.70GHz、24 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、192GB DDR4-2666 ECC RAM。CentOS* 7.3.1611 (Core)、Linux* カーネル 3.10.0-514.10.2.el7.x86_64。SSD: インテル® SSD DC S3700 シリーズ。ノード接続: 10Gb イーサネット。

インテル® Xeon® Gold 6148 プロセッサー
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、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、192GB DDR4-2666 ECC RAM。CentOS* 7.3.1611 (Core)、Linux* カーネル 3.10.0-514.10.2.el7.x86_64。ノード接続: インテル® Omni-Path ホスト・ファブリック、インテル® Omni-Path ホスト・ファブリック・インターフェイス・ドライバー 10.4.2.0.7。SSD: インテル® SSD DC S3700 シリーズ。

インテル® Xeon® Platinum 8170 プロセッサー
2x インテル® Xeon® Platinum 8170 プロセッサー、2.10GHz、26 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、384GB DDR4-2666 ECC RAM。CentOS* 7.3.1611 (Core)、Linux* カーネル 3.10.0-514.16.1.el7.x86_64。ノード接続: インテル® Omni-Path ホスト・ファブリック、インテル® Omni-Path ホスト・ファブリック・インターフェイス・ドライバー 10.4.2.0.7。SSD: インテル® SSD DC S3700 シリーズ (800GB)。

インテル® Xeon Phi™ プロセッサー 7250
1x インテル® Xeon Phi™ プロセッサー 7250、68 コア、コアあたり 4 ハードウェア・スレッド、1.40GHz、16GB 高速 MCDRAM (Quadrant キャッシュモード)、コアあたり 32KB L1 データキャッシュ、2 コアタイルあたり 1MB L2、96GB DDR4。ノード接続: インテル® Omni-Path ホスト・ファブリック、インテル® Omni-Path ホスト・ファブリック・インターフェイス・ドライバー 10.4.2.0.7。SSD: インテル® SSD DC S3500 シリーズ (480GB)。ソフトウェア: CentOS* 7.3.1611、Linux* カーネル 3.10.0-514.10.2.el7.x86_64、インテル® MPI ライブラリー 2017 Update 4。

16. 参考文献

[1]: https://software.intel.com/en-us/articles/tensorflow-optimizations-on-modern-intel-architecture (英語)

[2]: https://github.com/intel/caffe/ (英語)

[3]: https://www.isus.jp/products/psxe/intelguide-3-5/

[4]: http://man7.org/linux/man-pages/man3/numa.3.html (英語)

[5]: https://software.intel.com/en-us/node/522691 (英語)

[6]: https://www.isus.jp/hpc/process-and-thread-affinity-for-xeon-phi-processors-x200/

[7]: https://www.open-mpi.org/doc/v2.0/man1/mpiexec.1.php (英語)

17. 著者紹介

Vikram Saletore インテル コーポレーションの AI 製品グループ、カスタマー・ソリューション、パフォーマンス・イネーブリング・チームの主席エンジニア兼マシンラーニング/ディープラーニング・パフォーマンス・アーキテクト。インテル® Xeon® 製品およびインテル® Nervana™ 製品担当。ISV (Oracle*、Informix*) 向けに並列データベース・ソフトウェアの最適化、Cloudera* 向けに Apache Spark* のマシンラーニング解析の最適化を担当しました。HP* 研究所との共同研究では責任者を、最近の SURFsara とのディープラーニングの研究では共同研究責任者を務めました。インテルに入社する前は、オレゴン州立大学 (オレゴン州コーバリス市) でコンピューター・サイエンスの教職員として勤務し、米国国立科学財団 (NSF) による支援を受けた並列プログラミングと分散コンピューティングの研究で 8 人の生徒を監督していました。AMD* および DEC では、ネットワークや CPU アーキテクチャーに携わっていました。イリノイ大学アーバナ・シャンペーン校で電気工学の博士号、カリフォルニア大学バークレー校で電気工学の修士号を取得しています。6 つの特許を所有し (ほかにも 2 つの特許を出願中)、40 を超える研究論文を発表しています。

Deepthi Karkada インテル コーポレーションの AI 製品グループ、カスタマー・ソリューション、パフォーマンス・イネーブリング・チームのマシンラーニング・エンジニア。インテル® Xeon® アーキテクチャーおよびインテル® Nervana™ 製品を対象とするディープラーニング・フレームワークとプラットフォームの最適化およびベンチマーク作業に携わっています。以前は、Cloudera* Distribution of Hadoop* でのマシンラーニングおよびデータ解析向けに、インテル® MKL と Apache Spark* のシームレスな統合に携わっていました。

Vamsi Sripathi インテルのソフトウェア・エンジニア。2010 年に入社。ノースカロライナ州立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。インテル® Xeon® アーキテクチャーおよびインテル® Xeon Phi™ アーキテクチャーの数世代にわたり、インテル® MKL の BLAS ルーチンのパフォーマンス最適化に取り組みました。最近は、インテル® アーキテクチャーおよびインテル® Nervana™ 製品向けディープラーニング・アルゴリズムとフレームワークの最適化に取り組んでいます。

Kushal Datta インテル コーポレーションの AI 製品グループ、カスタマー・ソリューション、パフォーマンス・イネーブリング・チームの研究員。マシンラーニング、ディープラーニング、システム・パフォーマンスの最適化および CPU マイクロアーキテクチャーの設計に取り組んでいます。TileDB (多次元配列向け高性能格納ライブラリー) および GenomicsDB (GATK 4.0 で使用されているゲノムデータ格納システム) の筆頭筆者の 1 人です。インテルに入社する前は、ノースカロライナ大学シャーロット校の学生として、SPARC* V9 命令セット向けのサイクル精度の CPU シミュレーターの開発に対する研究助成金を受けていました。4 つの特許を所有し、数件の研究論文を発表しています。

Ananth Sankaranarayanan インテル コーポレーションの AI 製品グループ、AI ソリューションおよび AML (Applied Machine Learning) チームのエンジニアリング・ディレクター。世界中のクラウド・サービス・プロバイダー、企業、政府、通信サービス・プロバイダーに対するインテル® Xeon® 製品およびインテル® Nervana™ 製品のプロダクト・ポートフォリオのイネーブリングとスケーリングの責任を負っています。2001 年のインテル入社以降、さまざまなエンジニアリング・チームのリーダーを務めています。ハイパフォーマンス・コンピューティング分野の貢献に対するインテル・アチーブメント・アワードをはじめ、30 を超える部門表彰を受賞しています。コンピューター・サイエンス/工学の学士号、情報システムの MBA を取得しています。2 つの特許を所有し、数件の技術文献を執筆しています。

18. 法務上の注意書きと最適化に関する注意事項

インテルは、本資料で参照しているサードパーティーのベンチマーク・データまたはウェブサイトについて管理や監査を行っていません。本資料で参照しているウェブサイトにアクセスし、本資料で参照しているデータが正確かどうかを確認してください。

ベンチマーク結果は、「Spectre」および「Meltdown」と呼ばれる脆弱性への対処を目的とした最新のソフトウェア・パッチおよびファームウェア・アップデートの適用前に取得されたものです。パッチやアップデートを適用したデバイスやシステムでは同様の結果が得られないことがあります。

性能に関するテストに使用されるソフトウェアとワークロードは、性能がインテル® マイクロプロセッサー用に最適化されていることがあります。SYSmark* や MobileMark* などの性能テストは、特定のコンピューター・システム、コンポーネント、ソフトウェア、操作、機能に基づいて行ったものです。結果はこれらの要因によって異なります。製品の購入を検討される場合は、他の製品と組み合わせた場合の本製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。詳細については、**http://www.intel.com/performance/** (英語) を参照してください。

最適化に関する注意事項: インテル® コンパイラーでは、インテル® マイクロプロセッサーに限定されない最適化に関して、他社製マイクロプロセッサー用に同等の最適化を行えないことがあります。これには、インテル® ストリーミング SIMD 拡張命令 2、インテル® ストリーミング SIMD 拡張命令 3、インテル® ストリーミング SIMD 拡張命令 3 補足命令などの最適化が該当します。インテルは、他社製マイクロプロセッサーに関して、いかなる最適化の利用、機能、または効果も保証いたしません。本製品のマイクロプロセッサー依存の最適化は、インテル® マイクロプロセッサーでの使用を前提としています。インテル® マイクロアーキテクチャーに限定されない最適化のなかにも、インテル® マイクロプロセッサー用のものがあります。この注意事項で言及した命令セットの詳細については、該当する製品のユーザー・リファレンス・ガイドを参照してください。

テストでは、特定のシステムでの個々のテストにおけるコンポーネントの性能を文書化しています。ハードウェア、ソフトウェア、システム構成などの違いにより、実際の性能は掲載された性能テストや評価とは異なる場合があります。購入を検討される場合は、ほかの情報も参考にして、パフォーマンスを総合的に評価することをお勧めします。性能やベンチマーク結果について、さらに詳しい情報をお知りになりたい場合は、**http://www.intel.com/benchmarks/** (英語) を参照してください。

インテル® テクノロジーの機能と利点はシステム構成によって異なり、対応するハードウェアやソフトウェア、またはサービスの有効化が必要となる場合があります。実際の性能はシステム構成によって異なります。絶対的なセキュリティーを提供できるコンピューター・システムはありません。詳細については、各システムメーカーまたは販売店にお問い合わせいただくか、http://www.intel.co.jp/ を参照してください。

本資料で説明されている製品には、エラッタと呼ばれる設計上の不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。現在確認済みのエラッタについては、インテルまでお問い合わせください。

本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスも許諾するものではありません。

インテルは、明示されているか否かにかかわらず、いかなる保証もいたしません。ここにいう保証には、商品適格性、特定目的への適合性、および非侵害性の黙示の保証、ならびに履行の過程、取引の過程、または取引での使用から生じるあらゆる保証を含みますが、これらに限定されるわけではありません。

Intel、インテル、Intel ロゴ、Xeon、Intel Xeon Phi、Intel Nervana は、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。

* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。