Amazon Web Services* とインテルが協力して Amazon* EC2 CPU インスタンス上で最適化されたディープラーニング・フレームワークを実現

この記事は、インテル® AI Blog に公開されている「Amazon Web Services Works with Intel to Enable Optimized Deep Learning Frameworks on Amazon* EC2 CPU Instances」の日本語参考訳です。

幅広い計算、メモリー、通信構成を備えた Amazon Web Services* (AWS*) は、ディープラーニング (DL) システムを構築するリッチなプラットフォームを提供します[1]。さらに、事前ビルド済みのディープラーニング環境を含む Amazon* ディープラーニング AMI (Amazon* Machine Image) は、一般的なフレームワークのセットアップに必要な時間と労力を軽減し、同社の DL 製品を強化します[2]。

以前は、これらの事前ビルド済み環境は CPU 向けに最適化されていなかったため、ディープラーニング向けの CPU インスタンスを希望するユーザーは、利用可能なすべての CPU 最適化を活用するため、選択したフレームワークをソースからリビルドするか、カスタム・インストールする必要がありました。これに対処するため、AWS* は DL AMI を v6.0 にアップデートしました。このバージョンには、事前ビルド済みでインテル® マス・カーネル・ライブラリー (インテル® MKL) プリミティブとディープ・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) プリミティブを使用できるインテル® Optimizations for TensorFlow* が含まれています。これにより、データ・サイエンティストやディープラーニングの実践者は、実行するハードウェアに関係なくフレームワークが最適化されるため、直ちに作業に取り掛かることができます[3]。

この記事では、インテル® Optimization for TensorFlow* でスループットの向上を可能にするいくつかの最適化を概説し、ディープラーニング環境に特化した一般的な手法 (BKM) を確認します。

インテル® アーキテクチャー向けの最適化

インテルによる TensorFlow* 向けの CPU 最適化では、大きく分けて次の 3 つの手法が取られます。

1. プリミティブ操作を最新の SIMD 命令 (インテル® Xeon® プロセッサー向けのインテル® アドバンスト・ベクトル・エクステンション 2 (インテル® AVX2) およびインテル® アドバンスト・ベクトル・エクステンション 512 (インテル® AVX-512)) を使用してベクトル化し、CPU の並列計算能力を最大限に引き出します。畳み込み、内積、プーリング、正規化、活性化などの操作は、可能な限りインテル® MKL-DNN カーネルを利用してビルドされます。
2. グラフ最適化は、デフォルトの TensorFlow* 操作をインテルにより最適化された操作にシームレスに置き換えて、ネットワーク・アーキテクチャーを変更することなく、パフォーマンス・ゲインをもたらします。また、コストのかかるデータレイアウト変換を排除し、高速なバックプロパゲーションを可能にするため中間状態を制御します。
3. メモリーの最適化は、計算リソースを奪い合うことなく CPU 上で共生関係を維持しつつ、TensorFlow* とインテル® MKL カーネルが同じメモリープールを共有できるようにします。さらに、プリフェッチ、キャッシュ・ブロッキング手法、データ形式のバランスの良い使用により空間と時間の局所性を促進します。

これらの最適化は、プログラマーが意識することなく、パフォーマンスの向上をもたらします。CPU で TensorFlow* を高速化するインテルの取り組みの詳細は、Elmoustapha ほかによる記事[4] を参照してください。

データ・サイエンティストは、インテル® Optimization for TensorFlow* を使用することで、ディープラーニングの設計全般において大幅なスピードアップが得られるでしょう。図 1 は、デフォルトの TensorFlow* (AWS* の DL AMI v5.0 で利用可能) とインテル® Optimization for TensorFlow* (AWS* の DL AMI v6.0 で利用可能) のトレーニング・スループットを 4 つのベンチマーク・トポロジー (Inception-v3、ResNet-50、ResNet-152、VGG-16) で比較した結果です。

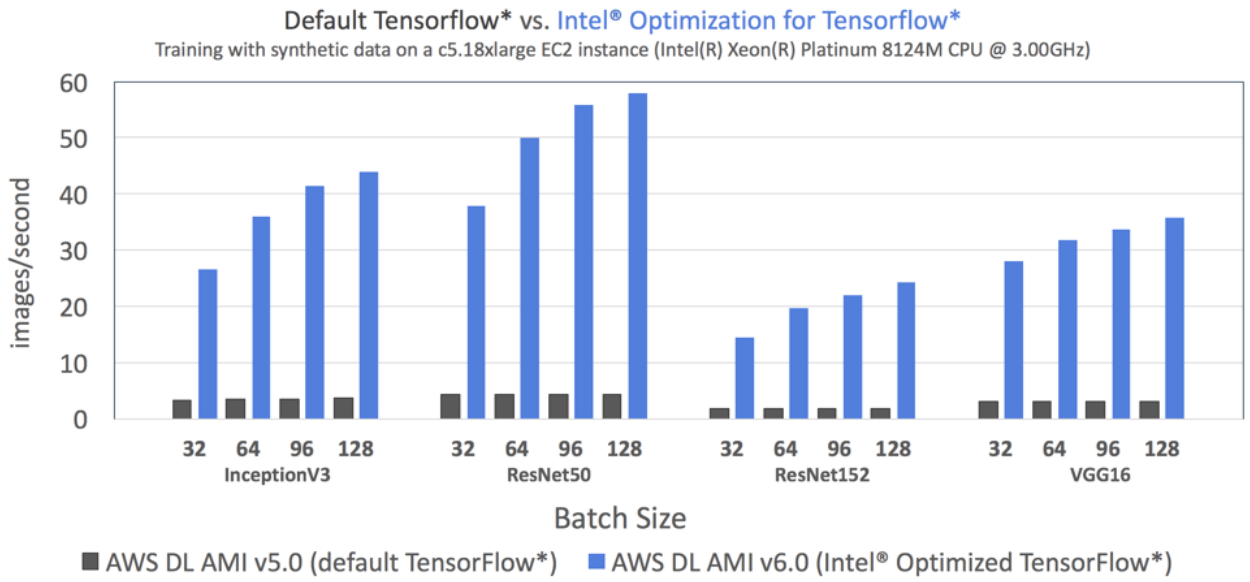

図 1: インテル® MKL とインテル® MKL-DNN による最適化を利用。すべてのベンチマーク・トポロジーにおいて (バッチサイズ 32 で) Amazon* ディープラーニング AMI v6.0 のインテル® Optimization for TensorFlow* はデフォルトの TensorFlow* よりも最大 7.4 倍高速[3]。使用したシステム構成は、次のとおりです。

プラットフォーム	Amazon EC2* c5.18xlarge インスタンス
コンピューターの種類	クラウド・コンピュート・サーバー
ソケット数	2S
プロセッサー	インテル® Xeon® Platinum 8124M CPU @ 3.00GHz (開発コード名 Skylake)
利用可能なコア数	ソケットごとに 18 コア
合計メモリー	144GB
SSD	EBS 最適化、128GB、プロビジョニングされた IOPS SSD
OS	Amazon* ディープラーニング AMI v6.0 (Ubuntu*)

環境向けの BKM

インテル® Optimization for TensorFlow* を使用すると、グラフ実行中に前述の最適化が確実に適用されますが、コア/メモリーの利用を最適にするには、追加でいくつかの環境変数と TensorFlow* パラメーターを設定する必要があります。TensorFlow* 仮想環境で次のコマンドを実行して、インテル® MKL が最適なパフォーマンスを達成するのに必要な環境変数を設定します。

```
export OMP_NUM_THREADS=<num_physical_cores>
export KMP_AFFINITY=granularity=fine,verbose,compact,1,0
export KMP_BLOCKTIME=1
export KMP_SETTINGS=1
export OMP_PROC_BIND=true
```

各設定の詳しい説明は、サポートされる環境変数に関する公式ドキュメントを参照してください[5]。さらに、マルチコア インテル® CPU の利点を引き出すには TensorFlow* で次の 2 つの変数を設定する必要があります。

1. intra-op threads: 一部の操作内でカーネルを並列化するスレッドプールのサイズ。
2. inter-op threads: 操作を並列に実行するスレッドプールのサイズ。

intra-op threads の最適な値は、計算グラフを実行するプロセスで利用可能な物理コア数に近くなりますが、トレーニング中のトポロジー、データセット、IO 要求によって異なるため、ユーザーがケースバイケースで設定する必要があります。一部のスレッドは、OS 外部プロセスや通信を処理できるように残しておく必要があります。intra-op threads 値を物理コアの最大数に設定すると、リソース不足となりプロセスがクラッシュします (特に分散計算を実行する場合)。

最新のトポロジー構造を考慮すると、通常 inter-op threads の最適な値は 1 です。カスタムトポロジーを使用する場合は、この値を計算グラフ内の独立した分岐の数に設定します。

これらの変数は、次のように、TensorFlow* スクリプト内で宣言します。

```
config = tf.ConfigProto(
    inter_op_parallelism_threads=num_inter_op_threads,
    intra_op_parallelism_threads=num_intra_op_threads)
```

インテル® MKL は NCHW (channels_first) データ形式向けに最適化されており、NHWC を使用する際にパフォーマンス・パリティーに近い値になるように継続的な取り組みが行われています。

ベンチマーク

図 1 のベンチマークを再現するには、次のステップを実行します。

1. CPU インスタンス (C4 または C5) を起動して、ディープラーニング AMI (Ubuntu*) を選択します。CPU 向けのディープラーニング AMI の起動と接続の手順は、[6] を参照してください。
2. インスタンスに接続したら、次のコマンドを実行してベンチマーク・スクリプトをダウンロードします。

```
git clone https://gist.github.com/MattsonThieme/60e7eba13d6dc80f7d69c52bebae4d19
```

以下は、プレーンテキスト形式のスクリプトです。

1. `intel_tf_cnn_benchmarks.sh` の `num_cores` をインスタンスの物理コア数に変更してください。これは、`OMP_NUM_THREADS` 環境変数と `intra_op_threads` TensorFlow* パラメーターの値として使用されます。
2. ターミナルで `bash intel_tf_cnn_benchmarks.sh` を実行してベンチマークを開始します。

```
#!/bin/bash
# Usage: bash intel_tf_cnn_benchmarks.sh

# Activate TensorFlow virtual environment
source activate tensorflow_p36

# Assign num_cores to the number of physical cores on your machine
num_cores=36

# Set environment variables
export KMP_AFFINITY=granularity=fine,verbose,compact,1,0
export KMP_BLOCKTIME=1
export KMP_SETTINGS=1
export OMP_NUM_THREADS=$num_cores
export OMP_PROC_BIND=true

# Clone benchmark scripts
git clone -b mkl_experiment https://github.com/tensorflow/benchmarks.git
cd benchmarks/scripts/tf_cnn_benchmarks
rm *.log # remove logs from any previous benchmark runs

# Run training benchmark scripts
networks=( alexnet googlenet inception3 resnet50 resnet152 vgg16 )
batch_sizes=( 1 32 64 96 128 )
for network in "${networks[@]}" ; do
for bs in "${batch_sizes[@]}"; do
echo -e "\n\n #### Starting $network with batch size = $bs ####\n\n"
time python tf_cnn_benchmarks.py --device cpu --data_format NCHW --cpu skl --data_name
synthetic --model "$network" --learning_rate 0.001 --num_epochs_per_decay 2 --batch_size
"$bs" --optimizer rmsprop --num_intra_threads $num_cores --num_inter_threads 1 --
num_omp_threads $num_cores --num_batches 100 2>&1 | tee net_"$network"_bs_"$bs".log
done
done

# Print training benchmark throughput
echo -e "\n Network batch_size images/second \n"

for network in "${networks[@]}" ; do
for bs in "${batch_sizes[@]}"; do
fps=$(grep  "total images/sec:"  net_"$network"_bs_"$bs".log | cut -d ":" -f2 | xargs)
echo "$network $bs $fps"
done
echo -e "\n"
done

# Deactivate virtual environment
source deactivate
```

導入について

AWS* によるインテル® Optimization for TensorFlow* の採用は、ディープラーニング向けの CPU の最適化を促進し、インテル® MKL とインテル® MKL-DNN のすべての最適化を利用することで、サイクル時間が短縮され CPU 上での開発ペースが加速されます。継続的なコラボレーションの一環として、インテルと Amazon* は CPU DL AMI をより完全なものにするため、今後いくつかの CPU 向けに最適化されたディープラーニング・フレームワークを追加する予定です。独自のディープラーニング・インスタンスを起動して、インテル® Optimization for TensorFlow* を使用してビルドするには、https://aws.amazon.com/machine-learning/amis/ を参照してください。

関連情報

1. https://aws.amazon.com/deep-learning/
2. https://aws.amazon.com/machine-learning/amis/
3. https://aws.amazon.com/blogs/machine-learning/faster-training-with-optimized-tensorflow-1-6-on-amazon-ec2-c5-and-p3-instances/ (英語)
4. https://www.isus.jp/machine-learning/tensorflow-optimizations-on-modern-intel-architecture/
5. https://software.intel.com/en-us/node/522775 (英語)
6. AWS* ディープラーニング AMI とインテル CPU を使用してディープラーニングを開始: PDF (英語)

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。