[大規模科学計算システム]

サブシステム AOBA-A における TensorFlow の利用方法

サイバーサイエンスセンター スーパーコンピューティング研究部 高橋 慧智

概要 サブシステム AOBA-A に搭載されている Vector Engine (VE) はメモリ帯域幅律速の数値シミュレーションを主眼に据えていますが，深層学習をはじめとする機械学習のためのソフトウェアエコシステムも活発に整備が進んでいます．本稿では，広く用いられている深層学習のためのオープンソースフレームワークである TensorFlow を VE 向けに移植した TensorFlow-VE を AOBA-A 上で利用する手順を紹介します．TensorFlow-VE を用いることで，CPU のみを用いる場合より，大幅に訓練を高速化することができます．

1　Python 3.8 のインストール

VE 向け TensorFlow (以下 TensorFlow-VE) の公式 GitHub リポジトリ (`https://github.com/sx-aurora-dev/tensorflow`) では，ビルド済みのバイナリを含む wheel パッケージが配布されていますが，Python 3.8 向けの wheel しか提供されていません．一方，AOBA にインストールされている Python のバージョンは 3.6 であるため，ホームディレクトリに Python 3.8 をインストールします．

まず，任意のバージョンの Python を容易にインストール可能にするツールである Pyenv (`https://github.com/pyenv/pyenv`) をインストールします．なお，以下ではフロントエンドノードのシェルとして bash を使用しているとします．

```
$ git clone https://github.com/pyenv/pyenv.git ~/.pyenv
$ echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bashrc
$ echo 'command -v pyenv >/dev/null || export PATH="$PYENV_ROOT/bin:$PATH"' \
>> ~/.bashrc
$ echo 'eval "$(pyenv init -)"' >> ~/.bashrc
```

次に，Python が依存するライブラリの 1 つである libffi のヘッダファイルが AOBA にはインストールされていないため，ホームディレクトリ以下にインストールします．

```
$ curl -sL https://github.com/libffi/libffi/releases/download/v3.4.2/\
libffi-3.4.2.tar.gz | tar xzvf -
$ cd libffi-3.4.2
$ ./configure --prefix=$HOME/.local
$ make
$ make install
```

libffi の共有ライブラリをインストールした`$HOME/.local/lib64` をライブラリ検索パスに追加しておきます．下記を`.bashrc` に追記します．

```
export LD_LIBRARY_PATH=$HOME/.local/lib64:$LD_LIBRARY_PATH
```

Pyenv を用いて Python 3.8 をインストールします．Python の configure スクリプトが，先ほどインストールした libffi を発見できるように環境変数によりオプションを渡します．

```
$ PYTHON_CONFIGURE_OPTS="CFLAGS=-I$HOME/.local/include \
PKG_CONFIG_PATH=$HOME/.local/lib/pkgconfig" pyenv install 3.8.13
$ pyenv global 3.8.13
```

最後に，python コマンドを実行した際に起動する Python のバージョンが 3.8 に切り替わっていることを確認します．

```
$ python3 -V
Python 3.8.13
```

2 TensorFlow-VE のインストール

GitHub リポジトリ (https://github.com/sx-aurora-dev/tensorflow) で提供されている wheel パッケージを用いて TensorFlow-VE をインストールします．ここでは，本稿を執筆時点で最新の 2.5.0 update 1 をインストールします．

```
$ curl -sL -O https://github.com/sx-aurora-dev/tensorflow/releases/download/\
tf-ve-2.5.0-u1/tensorflow_ve-2.5.0-cp38-cp38-linux_x86_64.whl
$ pip3 install tensorflow_ve-2.5.0-cp38-cp38-linux_x86_64.whl
```

3 サンプルスクリプトによる動作確認

GitHub リポジトリ (https://github.com/sx-aurora-dev/tf-samples) で公開されている TensorFlow-VE のサンプルテストを用いて動作確認を行います．ここでは，畳み込みニューラルネットワークにより手書き数字の画像データセット MNIST を分類するスクリプト (mnist_cnn/mnist_cnn.py) を使用します．

```
$ git clone https://github.com/sx-aurora-dev/tf-samples.git
```

このスクリプトでは，MNIST データセットを初回実行時に自動的にダウンロードしてホームディレクトリにキャッシュするようになっています．しかし，AOBA の計算ノードからはインターネットに疎通しないので，フロントエンドノード上で事前に MNIST データセットをダウンロードし，キャッシュしておきます．

```
$ python3 -c "import tensorflow as tf; tf.keras.datasets.mnist.load_data()"
```

以下の内容のジョブスクリプトを qsub コマンドで投入して訓練を実行します．TensorFlow-VE にはジョブスケジューラによって割り当てられていない VE を使用してしまう問題があるため，VE_NODE_NUMBER 環境変数を 0 に設定します．また，AOBA-A では，OMP_NUM_THREADS 環境変数のデフォルト値は 2 に設定されているため，VE_OMP_NUM_THREADS 環境変数を 8 に設定し，VE の全コアを使用するようにします．

```
#PBS -q sx
#PBS -l elapstim_req=00:05:00
#PBS --venode 1
#PBS -S /bin/bash

cd $PBS_O_WORKDIR

export VE_NODE_NUMBER=0
export VE_OMP_NUM_THREADS=8

python3 mnist_cnn/mnist_cnn.py -d /device:ve:0 --epoch 5 --verbose
```

ジョブの標準出力ファイルを確認すると，下記のように 1 エポックあたり 4〜5 秒で訓練を実行できていることがわかります．

```
batch_size=128 num_classes=10 epochs=5
x_train shape: (60000, 1, 28, 28)
60000 train samples
10000 test samples
Epoch 1/5
469/469 [==============================] - 5.064797s 10ms/step - loss:
   0.2160 - accuracy: 0.9346 - val_loss: 0.0528 - val_accuracy: 0.9826
(中略)
Epoch 5/5
469/469 [==============================] - 4.515069s 10ms/step - loss:
   0.0492 - accuracy: 0.9863 - val_loss: 0.0356 - val_accuracy: 0.9894
Test loss: 0.035571131855249405
Test accuracy: 0.9893999695777893
```

性能比較の対象として，同一の内容の訓練を下記のジョブスクリプトで AOBA-A の VH (AMD EPYC 7402P) のみを使用して実行してみます．なお，AMD EPYC 7402P は 24 コアを搭載していますが，AOBA-A ではシステム側で 8 コアを確保しているため，利用者が要求できるコア数の上限は 16 コアとなっています．

```
#PBS -q sx
#PBS -l elapstim_req=00:05:00
#PBS --venode 2
#PBS -S /bin/bash
#PBS --cpunum-lhost=16

cd $PBS_O_WORKDIR

export OMP_NUM_THREADS=16

python3 mnist_cnn/mnist_cnn.py -d /cpu:0 --epoch 5 --verbose --nhwc
```

VH 上で訓練を実行すると，1 エポックあたり 15〜16 秒かかります．したがって，このモデルとデータセットの場合には，VE を使用することにより VH の約 3 倍高速に訓練を実行することができます．TensorFlow-VE は複数の VE を用いた訓練にも対応しているため，VH に搭載されている 8 基全ての VE を使用すれば，更なる高速化が期待できます．