Intel DCAI Built-In Accelerators Compete Methodology

Methodology

This methodology summarizes the deployment and configuration steps taken by the Prowess Consulting engineers to test the following applications:

- OpenSSL®
- NAMD®
- ResNet®-50

For this study, conducted August 29, 2022, Intel sponsored Prowess Consulting to conduct testing and apply benchmarking best practices. Each of these applications was tested on 3rd Gen Intel® Xeon® Scalable processor–based Amazon Web Services® (AWS®) Elastic Compute Cloud™ (Amazon EC2®) and AMD EPYC™ 7003 Series processor–based Amazon EC2 instances:

- M6i.4xlarge instance with Intel Xeon Platinum 8375C processor (16 vCPUs)
- M6a.4xlarge instance with AMD EPYC 7R13 processor (16 vCPUs)
- M6a.8xlarge instance with AMD EPYC 7R13 processor (32 vCPUs)

Deployment Methodology

OpenSSL® on Intel® Processors

We took the following steps to configure OpenSSL for testing on the Intel® processor–based Amazon EC2 instance:

1. Log on to the AWS portal.
2. Launch an Amazon EC2 instance with the following details:
    a. **Name:** ***[Your Amazon EC2 instance name]***
    b. **Application and OS Images: ubuntu**
    c. **Amazon Machine Image: Ubuntu Server 20.04**
    d. **Instance Type: M6i.4xlarge** or **M6a.4xlarge**
    e. **Key pair:** Create a new key pair or use an existing pair.
    f. **Security group:** Select an existing security group with Secure Shell (SSH) enabled or create a new security group.
    g. **Configure storage:** Set size to a **100 GiB gp2** root volume.
3. Click **Launch Instance**.
4. Log on to the instance.
5. Run the following command to install updates:

```
sudo apt-get update && sudo apt-get upgrade
```

6. Run the following command to install the prerequisites for OpenSSL testing:

```
sudo apt install make cmake libssl-dev nasm autoconf libtool
```

7. Run the following remaining commands from the **/home/ubuntu** location on the Amazon EC2 instance:

```
mkdir –p openssl_benchmark

cd openssl_benchmark/

git clone https://github.com/openssl/openssl.git

git clone https://github.com/intel/ipp-crypto.git

git clone https://github.com/intel/intel-ipsec-mb.git

git clone https://github.com/intel/QAT_Engine.git

mkdir -p openssl_install

mkdir -p mb_build

export OPENSSL_SOURCE=/home/ubuntu/openssl_benchmark/openssl

export OPENSSL_LIB=/home/ubuntu/openssl_benchmark/openssl_install

export MB_LOCATION=/home/ubuntu/openssl_benchmark/mb_build

export IPPC_SOURCE=/home/ubuntu/openssl_benchmark/ipp-crypto

export IPSEC_SOURCE=/home/ubuntu/openssl_benchmark/intel-ipsec-mb

export QAT_ENGINE=/home/ubuntu/openssl_benchmark/QAT_Engine

export LD_LIBRARY_PATH=$OPENSSL_SOURCE:ROOT_LOC/:$OPENSSL_LIB/:/lib64
```

8. Run the following command to change directory into the OpenSSL Installation directory:

```
cd $OPENSSL_SOURCE
```

9. Run the following commands to install OpenSSL:

```
git checkout OpenSSL_1_1_1l

./config –prefix=$OPENSSL_LIB -Wl,-rpath,$OPENSSL_SOURCE

make update

make depend

make -j

make install -j

cd –
```

10. Run the following commands to install Intel® Integrated Performance Primitives (Intel® IPP) Cryptography:

```
cd $IPPC_SOURCE/sources/ippcp/crypto_mb

git checkout ipp-crypto_2021_5

cmake . -B"../build" -DOPENSSL_ROOT_DIR=$OPENSSL_ROOT -DCMAKE_INSTALL_PREFIX=$MB_
LOCATION -DOPENSSL_LIBRARIES=$OPENSSL_LIB

cd ../build

make -j

make install –j

cd /home/ubuntu/openssl_benchmark/
```

11. Run the following commands to install the Intel® Multi-Buffer Crypto for IPsec Library:

```
cd $IPSEC_SOURCE

git checkout v1.1

make –j

make install PREFIX=$MB_LOCATION

cd –
```

12. Run the following commands to install the Intel® QuickAssist Technology (Intel® QAT) Engine for a software-optimized experience:

```
cd $QAT_ENGINE

git checkout v0.6.11

./autogen.sh

./configure –enable-qat_sw  --disable-qat_hw –with-qat_sw_install_dir=$MB_LOCATION
–with-openssl_install_dir=$OPENSSL_LIB

make –j

make install –j

cd –
```

13. Run the following command to test the Intel QAT Engine:

```
$OPENSSL_SOURCE/apps/openssl engine –t –c qatengine
```

OpenSSL on AMD® Processors

We took the following steps to configure OpenSSL for testing on the AMD processor–based Amazon EC2 instance:

1. Log on to the AWS portal.
2. Launch an Amazon EC2 instance with the following details:
    - **Name:** ***[Your Amazon EC2 instance name]***
    - **Application and OS Images: ubuntu**
    - **Amazon Machine Image: Ubuntu Server 20.04**
    - **Instance Type: M6i.4xlarge** or **M6a.4xlarge**
    - **Key pair:** Create a new key pair or use an existing pair.
    - **Security group:** Select an existing security group with SSH enabled or create a new security group.
    - **Configure Storage:** Set size to a **100 GiB gp2** root volume.
3. Click **Launch Instance**.
4. Log on to the instance.
5. Run the following command to install updates:

```
sudo apt-get update && sudo apt-get upgrade
```

ResNet®-50 on Intel Processors

1. Log on to the AWS portal.
2. Launch an Amazon EC2 instance with the following details:
    - **Name:** ***[Your Amazon EC2 instance name]***
    - **Application and OS Images: ubuntu**
    - **Amazon Machine Image: Ubuntu Server 20.04**
    - **Instance Type: M6i.4xlarge** or **M6a.4xlarge**
    - **Key pair:** Create a new key pair or use an existing pair.
    - **Security group:** Select an existing security group with SSH enabled or create a new security group.
    - **Configure Storage:** Set size to a **100 GiB gp2** root volume.
3. Click **Launch Instance.**
4. Log on to the instance.
5. Run the following command to install updates:

```
sudo apt update -y && sudo apt upgrade -y
```

6. Run the following command to get Anaconda® for Linux®:

```
sudo wget https://repo.anaconda.com/archive/Anaconda3-2021.05-Linux-x86_64.sh
```

7. Run the following commands to start the Anaconda for Linux installer:

```
bash ./Anaconda3-2021.05-Linux-x86_64.sh
```

    - Press **Enter** to confirm the location.

```
[/home/ubuntu/anaconda3] >>> /home/ubuntu/anaconda3
```

8. Run the following command to source the Anaconda environment:

```
source /home/ubuntu/anaconda3/bin/activate
```

9. Run the following commands to install the Intel® Optimization for TensorFlow™ using aikit-tensorflow:

```
conda create -n aikit-tf -c intel intel-aikit-tensorflow
conda activate aikit-tf
pip install flatbuffers
```

10. Run the following command to verify the Intel Optimization for TensorFlow version:

```
python
>>> import tensorflow as tf
>>> print (tf.__version__)
2.8.0 <- This is correct version
exit()
```

11. Run the following command to install prerequisites for ResNet-50:

```
apt-get install numactl -y
```

12. Run the following commands to install and source the Intel artificial intelligence (AI) models:

```
mkdir AI; cd AI
git clone https://github.com/IntelAI/models.git
```

13. Run the following command to retrieve the ResNet-50 models:

```
wget https://storage.googleapis.com/intel-optimized-tensorflow/models/v1_8/
resnet50_fp32_pretrained_model.pb

export PRETRAINED_MODEL=$(pwd)/resnet50_fp32_pretrained_model.pb

cd models/benchmarks
```

14. Run the following command to launch into a virtual environment:

```
conda activate aikit-tf
```

ResNet-50 on AMD Processors

1. Log on to the AWS portal.
2. Launch an Amazon EC2 instance with the following details:
    - **Name:** ***[Your Amazon EC2 instance name]***
    - **Application and OS Images: ubuntu**
    - **Amazon Machine Image: Ubuntu Server 20.04**
    - **Instance Type: M6i.4xlarge** or **M6a.4xlarge**
    - **Key pair:** Create a new key pair or use an existing pair.
    - **Security group:** Select an existing security group with SSH enabled or create a new security group.
    - **Configure Storage:** Set size to a **100 GiB gp2** root volume.
3. Click **Launch Instance**.
4. Log on to the instance.
5. Run the following command to install updates:

```
sudo apt update -y && sudo apt upgrade -y
```

6. Run the following command to install dependencies:

```
sudo apt install unzip wget numactl hwloc hwloc-nox -y
```

7. Run the following commands to install and launch the Anaconda environment:

```
wget -P /tmp https://repo.anaconda.com/archive/Anaconda3-2020.02-Linux-x86_64.sh

bash /tmp/Anaconda3-2020.02-Linux-x86_64.sh

source ~/.bashrc
```

8. Browse to https://developer.amd.com/zendnn/ and download the **TF_v2.7_ZenDNN_v3.2_Python_v3.7.zip** from the AMD® Zen Deep Neural Network (ZenDNN) page.
9. Upload **TF_v2.7_ZenDNN_v3.2_Python_v3.7.zip** to the **/home/ubuntu** directory on the Amazon EC2 instance.
10. Browse to the **/home/ubuntu** directory:

```
cd /home/ubuntu
```

11. Run the following command to unzip the TensorFlow Zip file:

```
unzip TF_v2.7_ZenDNN_v3.2_Python_v3.7.zip
```

12. Browse to the **/home/ubuntu/TF_v2.7_ZenDNN_v3.2_Python_v3.7_2021-12-10T16** directory:

```
cd /home/ubuntu/ TF_v2.7_ZenDNN_v3.2_Python_v3.7_2021-12-10T16
```

13. Run the following command to install AMD ZenDNN:

```
source scripts/TF_ZenDNN_setup_release.sh
```

14. Run the following command to set up the AMD ZenDNN environment:

```
source scripts/zendnn_TF_env_setup.sh
```

15. Run the following command to download the Intel AI models:

```
git clone https://github.com/IntelAI/models.git
```

16. Run the following command to create an ImageNet_data directory:

```
mkdir /home/ubuntu/models/imagenet_data
```

17. Run the following command to browse to the ImageNet_data directory:

```
cd /home/ubuntu/models/imagenet_data
```

18. Upload the **ILSVRC2012_img_val.tar** files to the ImageNet_data directory:

19. Run the following commands to create an Anaconda environment with ImageNet®:

```
conda create -n imagenet python=3.8

conda activate imagenet

pip install intel-tensorflow==2.5.0

pip install -I urllib3

pip install wget
```

20. Run the following command to download the **ImageNet_to_tfrecords.sh** script:

```
wget https://raw.githubusercontent.com/IntelAI/models/master/datasets/imagenet/
imagenet_to_tfrecords.sh
```

21. Comment the following lines in the **ImageNet_to_tfrecords.sh** script:

```
22, 35, 36, 37, 49, 54, 57, 58, 59, 60, 61, 54, 65, 81, and 83
```

22. Run the following command to extract the ImageNet TAR files and run the **ImageNet_to_gcs.py** script to convert the files to TensorFlow records:

```
bash imagenet_to_tfrecords.sh $(pwd)
```

23. Run the following commands to configure the TensorFlow Model Zoo folder:

```
cd /home/ubuntu/models

git checkout 092e3c62fbf26548577c3f2a3886b270a15751cd

mkdir DATASET_DIR

mv /home/ubuntu/imagenet_data/tf_records /home/ubuntu/DATASET_DIR/

mkdir PRETRAINED_MODEL_DIR
```

24. Run the following commands to download the pretrained models:

```
cd PRETRAINED_MODEL_DIR

wget https://zenodo.org/record/2535873/files/resnet50_v1.pb

wget https://storage.googleapis.com/intel-optimized-tensorflow/models/v1_8/
resnet50v1_5_int8_pretrained_model.pb
```

25. Run the following commands to copy the FP32 batch-inference script:

```
cd quickstart/image_recognition/tensorflow/resnet50v1_5/inference/cpu/fp32

cp fp32_batch_inference.sh fp32_batch_inference_param.sh
```

26. Update the **fp32_batch_inference_param.sh** script:

    Line 58:
    ```
    --batch-size=${1}
    ```
    a. Line 60–61: Remove lines

    Line 64:
    ```
    --steps=${2}
    ```

27. Run the following commands to copy the INT8 batch-inference script:

```
cd quickstart/image_recognition/tensorflow/resnet50v1_5/inference/cpu/int8

cp int8_batch_inference.sh int8_batch_inference_param.sh
```

28. Update the int8_batch_inference_param.sh script:

    Line 58:
    ```
    --batch-size=${1}
    ```
    a. Line 62: Remove line

    Line 64:
    ```
    --steps=${2}
    ```

29. Run the following commands to modify the inference script to enable timing of the instance runs:

```
cd models/image_recognition/tensorflow/resnet50v1_5/inference/
```

    Make the following changes in **eval_image_classifier_inference.py**:

    Line 20 update:
    ```
    from datetime import datetime
    ```
    Line 182 update:
    ```
    from total_run = self.args.steps to total_run = self.args.steps
    + warm_up_iteration
    ```

    Between line 185 and line 186, insert the following:

```
if iteration == warm_up_iteration:

print('Instance Start: ', datetime.now().strftime('%a %d %b %Y %H:%M:%S.%f')[:-3])
```

    Above line 217, insert the following two lines:

```
print('Instance End:', datetime.now().strftime('%a %d %b %Y %H:%M:%S.%f')[:-3])

print('Num Iters:', iteration - warm_up_iteration)
```

30. Upload the **performance_test_zendnn.sh** script to the **/home/ubuntu/models** directory.
31. Modify the **performance_test_zendnn.sh BATCH_SIZE** to **1** and **16**.

NAMD® on Intel Processors

1. Log on to the AWS portal.
2. Launch an Amazon EC2 instance with the following details:
    - **Name:** ***[Your Amazon EC2 instance name]***
    - **Application and OS Images: ubuntu**
    - **Amazon Machine Image: Ubuntu Server 20.04**
    - **Instance Type: M6i.4xlarge** or **M6a.4xlarge**
    - **Key pair:** Create a new key pair or use an existing pair.
    - **Security group:** Select an existing security group with SSH enabled or create a new security group.
    - **Configure Storage:** Set size to a **100 GiB gp2** root volume.
3. Click **Launch Instance**.

4. Log on to the instance.
5. Run the following command to install updates:

```
sudo apt update -y && sudo apt upgrade -y
```

6. Request GitLab® access to https://www.ks.uiuc.edu/Research/namd/gitlabrequest.html.
7. Run the following commands to download the Intel® oneAPI Base Toolkit and the Intel® oneAPI HPC Toolkit:

```
wget https://registrationcenter-download.intel.com/akdlm/irc_nas/18673/l_
BaseKit_p_2022.2.0.262_offline.sh

wget https://registrationcenter-download.intel.com/akdlm/irc_nas/18679/
l_HPCKit_p_2022.2.0.191_offline.sh
```

8. Run the following commands to install the prerequisites:

```
sudo apt install build-essential libncurses-dev bison flex libssl-dev php ninja-build
environment-modules libelf-dev gcc-multilib gcc bison flex cmake libopencl-clang-12-dev clang
g++ libgtk-3-dev xorg python3-pip  alsa-tools x11-utils libgtk2.0-dev python2 -y

pip install pango
```

9. Run the following command to install the Intel oneAPI Base Toolkit:

```
bash l_BaseKit_p_2022.2.0.262_offline.sh -a -s --silent --eula accept
```

10. Run the following command to install the Intel oneAPI HPC Toolkit:

```
bash l_HPCKit_p_2022.2.0.191_offline.sh -a -s --silent --eula accept
```

11. Run the following commands to create the namd_benchmark directory and browse into the directory:

```
mkdir -p /home/ubuntu/namd_benchmark

cd /home/ubuntu/namd_benchmark
```

12. Upload the **build_namd_icc_mpi.sh, install_namd_icc.sh**, and **run_namd_Icc.sh** scripts to the **/home/ubuntu/namd_benchmark** directory.
13. Run the following command to install NAMD with Intel® C++ Compiler:

```
sudo bash install_namd_icc.sh
```

14. Run the following command to build NAMD with Intel C++ Compiler:

```
sudo bash build_namd_icc_mpi.sh
```

NAMD on AMD Processors

1. Log on to the AWS portal.
2. Launch an Amazon EC2 instance with the following details:
    - **Name:** ***[Your Amazon EC2 instance name]***
    - **Application and OS Images: ubuntu**
    - **Amazon Machine Image: Ubuntu Server 20.04**
    - **Instance Type: M6i.4xlarge** or **M6a.4xlarge**
    - **Key pair:** Create a new key pair or use an existing pair.
    - **Security group:** Select an existing security group with SSH enabled or create a new security group.
    - **Configure Storage:** Set size to a **100 GiB gp2** root volume.
3. Click **Launch Instance**.
4. Log on to the instance.

5. Run the following command to install updates:

```
sudo apt update -y && sudo apt upgrade -y
```

6. Request GitLab access to https://www.ks.uiuc.edu/Research/namd/gitlabrequest.html.
7. Download AMD Optimizing C/C++ and Fortran Compilers (AOCC) from https://developer.amd.com/amd-aocc/.
8. Upload AOCC to the AMD processor–based Amazon EC2 instance to the **/home/ubuntu** directory.
9. Run the following commands to download the Intel oneAPI Base Toolkit and the Intel oneAPI HPC Toolkit:

```
wget https://registrationcenter-download.intel.com/akdlm/irc_nas/18673/l_
BaseKit_p_2022.2.0.262_offline.sh

wget https://registrationcenter-download.intel.com/akdlm/irc_nas/18679/l_
HPCKit_p_2022.2.0.191_offline.sh
```

10. Run the following commands to install the prerequisites:

```
sudo apt install build-essential libncurses-dev bison flex libssl-dev php ninja-build
environment-modules libelf-dev gcc-multilib gcc bison flex cmake libopencl-clang-12-dev clang
g++ libgtk-3-dev xorg python3-pip  alsa-tools x11-utils libgtk2.0-dev python2 -y

pip install pango
```

11. Run the following command to install the Intel oneAPI Base Toolkit:

```
bash l_BaseKit_p_2022.2.0.262_offline.sh -a -s —silent —eula accept
```

12. Run the following command to install the Intel oneAPI HPC Toolkit:

```
bash l_HPCKit_p_2022.2.0.191_offline.sh -a -s --silent --eula accept
```

13. Run the following commands to create the namd_benchmark directory and browse into the directory:

```
mkdir -p /home/ubuntu/namd_benchmark

cd /home/ubuntu/namd_benchmark
```

14. Run the following commands to install the AMD AOCC:

```
cd /home/ubuntu/namd_benchmark

tar xf aocc-compiler-3.2.0.tar

cd aocc-compiler-3.2.0/

./install.sh

cd ..

source setenv_AOCC.sh
```

15. Run the following command to verify the AMD AOCC installation:

```
clang -v
```

16. Upload the **build_namd_aocc_mpi.sh**, **install_namd_aocc.sh**, and **run_namd_aocc.sh** scripts to the **/home/ubuntu/namd_benchmark** directory.
17. Run the following command to install NAMD with AMD AOCC:

```
sudo bash install_namd_aocc.sh
```

18. Run the following command to build NAMD with AMD AOCC:

```
sudo bash build_namd_aocc_mpi.sh
```

Test Methodology

OpenSSL

We ran the following commands to test OpenSSL on Intel processor–based Amazon EC2 instances, both utilizing the Intel QAT Engine and without utilizing the Intel QAT Engine, and the AMD processor–based Amazon EC2 instances without the Intel QAT Engine.

Single-thread tests with the Intel QAT Engine:

```
taskset -c 1 openssl speed -engine qatengine -multi 1 -evp Bulk cipher

taskset -c 1 openssl speed -engine qatengine -multi 1 PKE or Hash cipher
```

Single-thread tests without the Intel QAT Engine and for AMD processor–based Amazon EC2 instances:

```
taskset -c 1 openssl speed -multi 1 -evp Bulk cipher

taskset -c 1 openssl speed -multi 1 PKE or Hash cipher
```

Multi-thread tests with the Intel QAT Engine:

```
taskset -c 1-15 openssl speed -async_jobs 8 -engine qatengine -multi 15 -evp Bulk cipher

taskset -c 1-15 openssl speed -async_jobs 8 -engine qatengine -multi 15 PKE or Hash cipher
```

Multi-thread tests without the Intel QAT Engine and for AMD processor–based Amazon EC2 instances:

```
taskset -c 1-15 openssl speed -async_jobs 8 -multi 15 -evp Bulk cipher

taskset -c 1-15 openssl speed -async_jobs 8 -multi 15 PKE or Hash cipher
```

ResNet-50

We ran the following commands to test ResNet-50 on the Intel and AMD processor–based Amazon EC2 instances:

Intel ResNet-50 FP32, batch size 1:

```
python launch_benchmark.py --model-name resnet50 --precision fp32 --mode inference
--framework tensorflow --in-graph /home/ubuntu/AI/resnet50_fp32_pretrained_model.pb -a 8 -e
1 --numa-cores-per-instance 8 --batch-size 1
```

Intel ResNet-50 FP32, batch size 16:

```
python launch_benchmark.py --model-name resnet50 --precision fp32 --mode inference
--framework tensorflow --in-graph     /home/ubuntu/AI/resnet50_fp32_pretrained_model.pb -a 8
-e 1 --numa-cores-per-instance 8 --batch-size 16
```

Intel ResNet-50 INT8 batch size 1:

```
python launch_benchmark.py --model-name resnet50   --precision int8   --mode inference
--framework tensorflow --in-graph     /home/ubuntu/AI/resnet50_int8_pretrained_model.pb -a 8
-e 1 --numa-cores-per-instance 8 --batch-size 1
```

Intel ResNet-50 INT8, batch size 16:

```
python launch_benchmark.py --model-name resnet50 --precision int8 --mode inference
--framework tensorflow --in-graph     /home/ubuntu/AI/resnet50_int8_pretrained_model.pb -a 8
-e 1 --numa-cores-per-instance 8 --batch-size 16
```

AMD ResNet-50 FP32, batch size 1 and 16:

```
nohup bash performance_test_zendnn.sh fp32 2400 > resnet50v1_5_fp32_log.log
```

AMD ResNet-50 INT8, batch size 1 and 16:

```
nohup bash performance_test_zendnn.sh int8 2400 > resnet50v1_5_int8_log.log
```

NAMD

We ran the following commands to test NAMD on the Intel and AMD processor–based Amazon EC2 instances:

Intel NAMD APOA1 and STMV:

```
sudo bash run_namd_icc.sh
```

AMD NAMD APOA1 and STMV:

```
sudo bash run_namd_aocc.sh
```