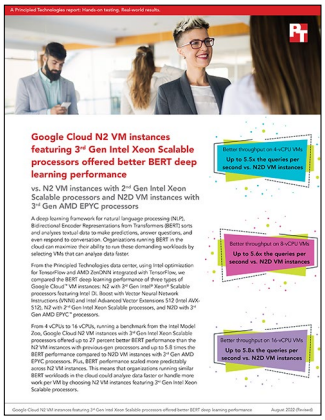

The science behind the report:

Google Cloud N2 VM instances featuring 3rd Gen Intel Xeon Scalable processors offered better BERT deep learning performance

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Google Cloud N2 VMs featuring 3rd Gen Intel Xeon Scalable processors offered better BERT deep learning performance.

We concluded our hands-on testing on May 13, 2022. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on May 5, 2022 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

To learn more about how we have calculated the wins in this report, go to http://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

Table 1: Throughput results of our testing, in examples per second.

	standard-4	standard-8	standard-16	standard-32	standard-64
Batch size 1					
N2 3rd Gen (FP32)	0.75	1.41	2.55	4.05	4.87
N2 2nd Gen (FP32)	0.65	1.24	2.2	3.28	3.91
N2D 3rd Gen AMD (FP32)	0.44	0.80	1.40	2.07	2.22
N2 3rd Gen (INT8)	2.44	4.51	8.15	12.60	12.90
N2 2nd Gen (INT8)	2.07	3.99	6.39	8.90	9.40
Batch size 32					
N2 3rd Gen (FP32)	0.93	1.81	3.35	5.81	8.64
N2 2nd Gen (FP32)	0.80	1.59	2.92	4.42	6.71
N2D 3rd Gen AMD (FP32)	0.56	1.04	1.83	2.78	3.39
N2 3rd Gen (INT8)	2.58	4.9	8.85	14.99	24.12
N2 2nd Gen (INT8)	2.27	4.47	8.06	11.77	18.55

System configuration information

Table 2: Detailed information about the N2 standard VM instances (2nd Gen Intel Xeon Scalable processors) we tested.

System configuration information	n2-standard-4 (2nd Gen)	n2-standard-8 (2nd Gen)	n2-standard-16 (2nd Gen)
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	05/12/2022	05/12/2022	05/12/2022
Google Cloud region	us-central1-a	us-central1-a	us-central1-a
Workload	BERT-Large, Uncased (Whole Word Masking)	BERT-Large, Uncased (Whole Word Masking)	BERT-Large, Uncased (Whole Word Masking)
Workload-specific parameters	Sequence Length: 384	Sequence Length: 384	Sequence Length: 384
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	n2-standard-4	n2-standard-8	n2-standard-16)
BIOS name and version	Google 1.0 01/01/2021	Google 1.0 01/01/2021	Google 1.0 01/01/2021
Operating system name and version/build number	Ubuntu 20.04 LTS 5.13.0-1025-gcp	Ubuntu 20.04 LTS 5.13.0-1025-gcp	Ubuntu 20.04 LTS 5.13.0-1025-gcp
Date of last OS updates/ patches applied	05/11/2022	05/11/2022	05/11/2022
Processor			
Number of processors	1	1	1
Vendor and model	Intel® Xeon® 6268CL	Intel Xeon 6268CL	Intel Xeon 6268CL
VM core count (per processor)	2	4	8
Core frequency (GHz)	2.80	2.80	2.80
Stepping	7	7	7
Hyper-threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Number of vCPU per VM	4	8	16
Memory module(s)			
Total memory in system (GB)	16	32	64
General HW			
Storage: NW or Direct Att / Instance	Direct Att / Instance	Direct Att / Instance	Direct Att / Instance
Local storage			
Number of drives	1	1	1
Drive size (GB)	50	50	50
Drive information	SSD persistent disk	SSD persistent disk	SSD persistent disk
Network adapter			
Vendor and model	Virtio network device	Virtio network device	Virtio network device
Number and type of ports	1x (Up to 10 Gbps)	1x (Up to 16 Gbps)	1x (Up to 32 Gbps)

Table 3: Detailed information about the N2D standard VM instances (3rd Gen AMD EPYC processors) we tested.

System configuration information	n2d-standard-4 (3rd Gen AMD)	n2d-standard-8 (3rd Gen AMD)	n2d-standard-16 (3rd Gen AMD)
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	05/12/2022	05/12/2022	05/12/2022
Google Cloud region	us-central1-a	us-central1-a	us-central1-a
Workload	BERT-Large, Uncased (Whole Word Masking)	BERT-Large, Uncased (Whole Word Masking)	BERT-Large, Uncased (Whole Word Masking)
Workload-specific parameters	Sequence Length: 384	Sequence Length: 384	Sequence Length: 384
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	n2d-standard-4	n2d-standard-8	n2d-standard-16
BIOS name and version	Google 1.0 01/01/2021	Google 1.0 01/01/2021	Google 1.0 01/01/2021
Operating system name and version/build number	Ubuntu 20.04 LTS 5.13.0-1025-gcp	Ubuntu 20.04 LTS 5.13.0-1025-gcp	Ubuntu 20.04 LTS 5.13.0-1025-gcp
Date of last OS updates/ patches applied	05/11/2022	05/11/2022	05/11/2022
Processor			
Number of processors	1	1	1
Vendor and model	AMD EPYC™ 7B13 processor	AMD EPYC 7B13 processor	AMD EPYC 7B13 processor
VM core count (per processor)	2	4	8
Core frequency (GHz)	2.45	2.45	2.45
Stepping	0	0	0
Hyper-threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Number of vCPU per VM	4	8	16
Memory module(s)			
Total memory in system (GB)	16	32	64
General HW			
Storage: NW or Direct Att / Instance	Direct Att / Instance	Direct Att / Instance	Direct Att / Instance
Local storage			
Number of drives	1	1	1
Drive size (GB)	50	50	50
Drive information	SSD persistent disk	SSD persistent disk	SSD persistent disk
Network adapter			
Vendor and model	Virtio network device	Virtio network device	Virtio network device
Number and type of ports	1x (Up to 10 Gbps)	1x (Up to 16 Gbps)	1x (Up to 32 Gbps)

Table 4: Detailed information about the N2 standard VM instances (3rd Gen Intel Xeon Scalable processors) we tested.

System configuration information	n2-standard-4 (3rd Gen)	n2-standard-8 (3rd Gen)	n2-standard-16 (3rd Gen)
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	05/12/2022	05/12/2022	05/12/2022
Google Cloud region	us-central1-a	us-central1-a	us-central1-a
Workload	BERT-Large, Uncased (Whole Word Masking)	BERT-Large, Uncased (Whole Word Masking)	BERT-Large, Uncased (Whole Word Masking)
Workload-specific parameters	Sequence Length: 384	Sequence Length: 384	Sequence Length: 384
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	n2-standard-4	n2-standard-8	n2-standard-16
BIOS name and version	Google 1.0 01/01/2021	Google 1.0 01/01/2021	Google 1.0 01/01/2021
Operating system name and version/build number	Ubuntu 20.04 LTS 5.13.0-1025-gcp	Ubuntu 20.04 LTS 5.13.0-1025-gcp	Ubuntu 20.04 LTS 5.13.0-1025-gcp
Date of last OS updates/ patches applied	05/11/2022	05/11/2022	05/11/2022
Processor			
Number of processors	1	1	1
Vendor and model	Intel Xeon 8373C	Intel Xeon 8373C	Intel Xeon 8373C
VM core count (per processor)	2	4	8
Core frequency (GHz)	2.80	2.80	2.80
Stepping	6	6	6
Hyper-threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Number of vCPU per VM	4	8	16
Memory module(s)			
Total memory in system (GB)	16	32	64
General HW			
Storage: NW or Direct Att / Instance	Direct Att / Instance	Direct Att / Instance	Direct Att / Instance
Local storage			
Number of drives	1	1	1
Drive size (GB)	50	50	50
Drive information	SSD persistent disk	SSD persistent disk	SSD persistent disk
Network adapter			
Vendor and model	Virtio network device	Virtio network device	Virtio network device
Number and type of ports	1x (Up to 10 Gbps)	1x (Up to 16 Gbps)	1x (Up to 32 Gbps)

How we tested

Testing overview

We tested three types of Google Cloud VM instances: N2 standard (Ice Lake) with 3rd Gen Intel Xeon Scalable processors, N2 standard (Cascade Lake) with 2nd Gen Intel Xeon Scalable processors, and N2D standard (Milan) with 3rd Gen AMD EPYC processors. We ran a BERT workload on the GCP instances to show the performance difference between these instance types in terms of examples per second.

Creating the Google Cloud VM instance

1. Log into Google Cloud, and click Go to console.
2. Click Compute engine, and click VM instances.
3. Click Create.
4. In the left window, select New VM instance.
5. Add the following information:
    a. Name: Name your VM instance.
    b. Labels: Use any appropriate labels.
    c. Region: Select your desired region. We used us-central1.
    d. Zone: Select your desired zone. We used us-central1-a.
    e. Machine Configuration:
    f. Machine family: General-purpose
    g. Series: {N2,N2D}
    h. Machine type: {n2,n2d} standard-4,8,16
    i. CPU platform: {Intel Cascade Lake, Intel Ice Lake, AMD Milan} or later
    j. Keep Enable display device unchecked.
    k. Keep Confidential VM Service unchecked.
    l. Boot Disk, click Change:
        i. Operating System: Ubuntu
        ii. Version: Ubuntu 20.04 LTS
        iii. Boot disk type: SSD persistent disk
        iv. Size: 50GB
        v. Click Select
    m. Identity and API access: Compute Engine default service account.
6. Delete Disk
7. Click Save.
8. Click Create.

Configuring Ubuntu 20.04 LTS for BERT benchmark

1. Log into the GCP instance via SSH.
2. Install updates, and reboot the instance:

```
sudo apt-get update
sudo apt-get upgrade -y
sudo reboot
```

3. Install new tools:

```
sudo apt-get autoremove -y
sudo apt-get install -y build-essential numactl hwloc zip unzip sysstat
```

4. Install Anaconda. Accept the license agreement when prompted, choose the default installation location, and answer Yes when prompted to initialize Anaconda using conda init:

```
wget https://repo.anaconda.com/archive/Anaconda3-2021.11-Linux-x86_64.sh
bash Anaconda3-2021.11-Linux-x86_64.sh
```

5. Install and configure libtcmalloc:

```
sudo apt-get install -y libgoogle-perftools4
sudo ln -s /lib/x86_64-linux-gnu/libtcmalloc.so.4 /usr/lib/libtcmalloc.so
```

6. Install and configure docker-ce:

```
sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common
gnupg lsb-release
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
sudo apt-get install -y docker-ce
sudo usermod -aG docker ${USER}
```

7. Log out of the VM instance, then log back in via SSH to enable Conda and Docker environment variables.

Installing ZenDNN v3.2 TensorFlow v2.7

1. Download the TF_v2.7_ZenDNN_v3.2_Python_v3.8.zip bundle from https://developer.amd.com/zendnn/.
2. Extract the downloaded archive:

```
unzip TF_v2.7_ZenDNN_v3.2_Python_v3.8.zip
```

3. Install ZenDNN v3.2 TensorFlow v2.7:

```
cd TF_v2.7_ZenDNN_v3.2_Python_v*/
source scripts/TF_ZenDNN_setup_release.sh
```

This will automatically create a new Conda environment named tf-2.7-zendnn-v3.2-rel-env.

Installing Intel TensorFlow v2.7

1. Create a new conda environment with Python v3.8 to use with Intel TensorFlow v2.7, and activate it:

```
conda create -n tf-2.7-intel python=3.8 -y
conda activate tf-2.7-intel
```

2. Install intel-tensorflow-2.7 in the conda environment by running the following command:

```
pip install intel-tensorflow==2.7
```

Downloading the BERT benchmark and models

1. Make a directory to hold the benchmark and all model files:

```
mkdir /bert
cd /bert
```

2. Download the BERT benchmark from Intel Model Zoo git repo:

```
git clone https://github.com/IntelAI/models.git
git -C models checkout 01839eaa
```

3. Patch the benchmark to include INT8 support (using patch file included in the Appendix):

```
mkdir models/benchmarks/language_modeling/tensorflow/bert_large/inference/int8
cd models/benchmarks/language_modeling/tensorflow/bert_large/inference/int8
wget https://raw.githubusercontent.com/IntelAI/models/icx-launch-public/benchmarks/language_
modeling/tensorflow/bert_large/inference/int8/config.json
wget https://raw.githubusercontent.com/IntelAI/models/icx-launch-public/benchmarks/language_
modeling/tensorflow/bert_large/inference/int8/model_init.py
patch model_init.py int8_model_init.py.patch
cd /bert
```

4. Download and unzip the BERT large uncased (whole word masking) model from the Google BERT repo:

```
wget https://storage.googleapis.com/bert_models/2019_05_30/wwm_uncased_L-24_H-1024_A-16.zip
unzip wwm_uncased_L-24_H-1024_A-16.zip
```

5. Download the dev-v1.1.json file from the Google BERT repo into the wwm_uncased_L-24_H-1024_A-16 directory that you just unzipped:

```
wget https://rajpurkar.github.io/SQuAD-explorer/dataset/dev-v1.1.json -P wwm_
uncased_L-24_H-1024_A-16
```

6. Download and unzip the pretrained model checkpoint files:

```
wget https://storage.googleapis.com/intel-optimized-tensorflow/models/v1_8/bert_large_
checkpoints.zip
unzip bert_large_checkpoints.zip
```

7. Download the FP32 frozen graph:

```
wget https://storage.googleapis.com/intel-optimized-tensorflow/models/v2_7_0/fp32_bert_squad.pb
```

8. Download the INT8 frozen graph:

```
wget https://storage.googleapis.com/intel-optimized-tensorflow/models/r2.5-icx-b631821f/
asymmetric_per_channel_bert_int8.pb
```

Running the BERT benchmark

1. If using FP32, activate the conda environment for either ZenDNN or Intel TensorFlow:

   • Intel TensorFlow:

   ```
   conda activate tf-2.7-intel
   ```

   • ZenDNN TensorFlow:

   ```
   conda activate tf-2.7-zendnn-v3.2-rel-env
   ```

2. Run the conda FP32 script:

   ```
   ./run_bert_conda.sh
   ```

3. If using INT8, then simply running the INT8 script and this will automatically download and create the docker container needed:

   ```
   ./run_bert_int8.sh
   ```

4. Save the throughput results.

Appendix: Scripts

run_bert_conda.sh:

```
#!/bin/bash
cd /bert/models/benchmarks
echo ${CONDA_DEFAULT_ENV}

export OUTPUT_DIR=/bert/output_$(curl -q http://169.254.169.254/latest/meta-data/instance-type 2>/
dev/null)_${CONDA_DEFAULT_ENV}
export CHECKPOINT_DIR=/bert/bert_large_checkpoints
export DATASET_DIR=/bert/wwm_uncased_L-24_H-1024_A-16
export PRETRAINED_MODEL=/bert/fp32_bert_squad.pb

mkdir -p ${OUTPUT_DIR}

time python launch_benchmark.py \
    --model-name=bert_large \
    --precision=fp32 \
    --mode=inference \
    --framework=tensorflow \
    --batch-size=1 \
    --data-location ${DATASET_DIR} \
    --in-graph ${PRETRAINED_MODEL} \
    --checkpoint ${CHECKPOINT_DIR} \
    --output-dir ${OUTPUT_DIR} \
    --benchmark-only \
    --warmup-steps=30 \
    --steps=90 \
    --infer_option=SQuAD

time python launch_benchmark.py \
    --model-name=bert_large \
    --precision=fp32 \
    --mode=inference \
    --framework=tensorflow \
    --batch-size=32 \
    --data-location ${DATASET_DIR} \
```

```
    --in-graph ${PRETRAINED_MODEL} \
    --checkpoint ${CHECKPOINT_DIR} \
    --output-dir ${OUTPUT_DIR} \
    --benchmark-only \
    --warmup-steps=3 \
    --steps=9 \
    --infer_option=SQuAD

grep -B1 -A2 "Throughput: " ${OUTPUT_DIR}/*.log
```

run_bert_int8.sh:

```
#!/bin/bash
cd /bert/models/benchmarks
DOCKER_IMAGE=tf-r2.5-icx-b631821f
echo ${DOCKER_IMAGE}

export OUTPUT_DIR=/bert/output_$(curl -q http://169.254.169.254/latest/meta-data/instance-type 2>/
dev/null)_${DOCKER_IMAGE}
export CHECKPOINT_DIR=/bert/bert_large_checkpoints
export DATASET_DIR=/bert/wwm_uncased_L-24_H-1024_A-16
export PRETRAINED_MODEL=/bert/asymmetric_per_channel_bert_int8.pb

mkdir -p ${OUTPUT_DIR}

time python launch_benchmark.py \
    --model-name=bert_large \
    --precision=int8 \
    --mode=inference \
    --framework=tensorflow \
    --batch-size=1 \
    --data-location ${DATASET_DIR} \
    --in-graph ${PRETRAINED_MODEL} \
    --checkpoint ${CHECKPOINT_DIR} \
    --output-dir ${OUTPUT_DIR} \
    --benchmark-only \
    --warmup-steps=30 \
    --steps=90 \
    --docker-image intel/intel-optimized-tensorflow:${DOCKER_IMAGE} \
    --infer_option=SQuAD

time python launch_benchmark.py \
    --model-name=bert_large \
    --precision=int8 \
    --mode=inference \
    --framework=tensorflow \
    --batch-size=32 \
    --data-location ${DATASET_DIR} \
    --in-graph ${PRETRAINED_MODEL} \
    --checkpoint ${CHECKPOINT_DIR} \
    --output-dir ${OUTPUT_DIR} \
    --benchmark-only \
    --warmup-steps=3 \
    --steps=9 \
    --docker-image intel/intel-optimized-tensorflow:${DOCKER_IMAGE} \
    --infer_option=SQuAD

grep -B1 -A2 "Throughput: " ${OUTPUT_DIR}/*.log
```

int8_model_init.py.patch:

```
--- model_init.py    2022-04-09 06:26:14.335506293 -0400
+++ int8_model_init.py      2022-03-07 16:26:12.305330529 -0500
@@ -25,6 +25,7 @@
 from common.base_model_init import set_env_var

 import os
+import sys
 from argparse import ArgumentParser


@@ -38,6 +39,24 @@
         if not platform_util:
             raise ValueError("Did not find any platform info.")

+        # multi-instance runs on bert require each instance to have separate output folders
+        # and the output folders should be empty to prevent conflicts with previous runs
+        if self.args.numa_cores_per_instance and os.path.exists(
+                self.args.output_dir):
+            if os.listdir(self.args.output_dir):
+                sys.exit(
+                    "ERROR: The output directory ({}) is not empty. BERT multi-instance "
+                    "runs require empty output directories in order to prevent conflicts "
+                    "with files generated by previous runs. Please provide an empty folder "
+                    "using the --output-dir argument.".format(self.args.output_dir))
+
+        # BERT FP32 inference requires a frozen graph
+        if not self.args.input_graph:
+            sys.exit("ERROR: BERT large FP32 inference requires a frozen graph to be passed "
+                     "to the launch_benchmark.py script using the --in-graph arg. Please download "
+                     "the frozen graph using the instructions in the README.md and
update your command "
+                     "to add the input graph.")
+
         # use default batch size of 32 if it's -1
         if self.args.batch_size == -1:
             self.args.batch_size = 32
@@ -63,6 +82,12 @@
             "--experimental-gelu", dest="experimental_gelu", default="False")
         arg_parser.add_argument(
             "--optimized-softmax", dest="optimized_softmax", default="True")
+        arg_parser.add_argument("--warmup-steps", dest='warmup_steps',
+                                type=int, default=10,
+                                help="number of warmup steps")
+        arg_parser.add_argument("--steps", dest='steps',
+                                type=int, default=30,
+                                help="number of benchmark steps")

         self.args = arg_parser.parse_args(self.custom_args, namespace=self.args)

@@ -133,6 +158,12 @@
         if self.args.num_intra_threads:
             model_args += " --intra_op_parallelism_threads=" + str(self.args.num_intra_threads)

+        if self.args.warmup_steps:
+            model_args += " --warmup_steps=" + str(self.args.warmup_steps)
+
+        if self.args.steps:
+            model_args += " --steps=" + str(self.args.steps)
+
         self.benchmark_command = self.get_command_prefix(args.socket_id) + \
             self.python_exe + " " + model_script + model_args
```

Read the report at https://facts.pt/7709ho6

This project was commissioned by Intel.

Principled Technologies®

Facts matter.®