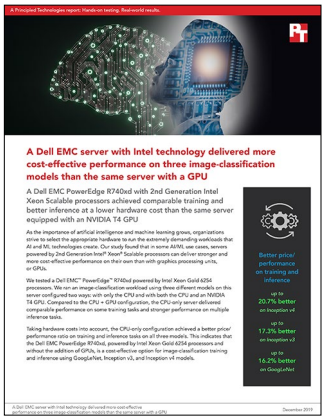

The science behind the report:

A Dell EMC server with Intel technology delivered more cost-effective performance on three image-classification models than the same server with a GPU

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 A Dell EMC server with Intel technology delivered more cost-effective performance on three image-classification models than the same server with a GPU.

We concluded our hands-on testing on November 18, 2019. 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 November 12, 2019 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

The tables below presents our findings in detail.

Training performance (average images per second)

Model	CPU-only configuration	CPU + GPU configuration	Percentage difference
GoogLeNet	1,126.2	1,011.8	11.30%
Inception v4	160.9	136.9	17.56%
Inception v3	333.7	296.1	12.70%

Inference performance (average images per second)

Model	CPU-only configuration	CPU + GPU configuration	Percentage difference
GoogLeNet	300.4	321.2	-6.47%
Inception v4	38.6	37.4	6.51%
Inception v3	84.1	79.0	3.37%

Publicly available pricing for the server configurations we tested (with extra RAM)

		CPU-only configuration		CPU + GPU configuration	
	Price	**Quantity**	**Extended price**	**Quantity**	**Extended price**
Basic server components	$3,695.00	1	$3,695.00	1	$3,695.00
Intel® Xeon® Gold 6254 processor	$3,367.82	2	$6,735.64	2	$6,735.64
64GB RAM stick	$1,438.23	24	$34,517.52	24	$34,517.52
NVIDIA™ Tesla™ T4 GPU	$3,298.07	0	N/A	1	$3,298.07
Total hardware cost			$44,948.16		$48,246.23
Percentage difference in total cost					7.34

Publicly available pricing for hypothetical server configurations (with adequate RAM)

		CPU-only configuration		CPU + GPU configuration	
	Price	**Quantity**	**Extended price**	**Quantity**	**Extended price**
Basic server components	$3,695.00	1	$3,695.00	1	$3,695.00
Intel Xeon Gold 6254 Processor	$3,367.82	2	$6,735.64	2	$6,735.64
8GB RAM stick	$238.13	24	$5,715.12	24	$5,715.12
NVIDIA Tesla T4 GPU	$3,298.07	0	N/A	1	$3,298.07
Total hardware cost			$16,145.76		$19,443.83
Percentage difference in total cost					20.43

Price relative to training performance in the server configurations we tested (with extra RAM)

Model	**CPU-only configuration**	**CPU + GPU configuration**	**Percentage difference**
GoogLeNet	$39.91	$47.68	16.29%
Inception v4	$279.30	$352.45	20.75%
Inception v3	$134.70	$162.94	17.34%

Price relative to training performance in hypothetical server configurations (with adequate RAM)

Model	**CPU-only configuration**	**CPU + GPU configuration**	**Percentage difference**
GoogLeNet	$14.34	$19.22	25.39%
Inception v4	$100.33	$142.04	29.37%
Inception v3	$48.38	$65.67	26.32%

Price relative to inference performance in the server configurations we tested (with extra RAM)

Model	CPU-only configuration	CPU + GPU configuration	Percentage difference
GoogLeNet	$150.22	$149.63	0.39%
Inception v4	$1,164.16	$1,291.73	9.88%
Inception v3	$534.33	$610.87	12.53%

Price relative to inference performance in hypothetical server configurations (with adequate RAM)

Model	CPU-only configuration	CPU + GPU configuration	Percentage difference
GoogLeNet	$53.75	$60.54	11.22%
Inception v4	$418.18	$520.58	19.67%
Inception v3	$191.94	$246.19	22.04%

System configuration information

The table below presents detailed information on the systems we tested.

Server configuration information	Dell EMC™ PowerEdge R740xd	
BIOS name and version	2.3.10	
Non-default BIOS settings	Virtualization enabled	
Operating system name and version/build number	CentOS 7 x86_64 18.10 (3.10.0-1062.1.2.el7.x86_64)	
Date of last OS updates/patches applied	10/04/2019	
Power management policy	Performance	
Processor		
Number of processors	2	
Vendor and model	Intel® Xeon® Gold 6254 CPU	
Core count (per processor)	18	
Core frequency (GHz)	3.10	
Graphics processing unit (when used)		
Number of GPUs	1	
Vendor and model	NVIDIA Corporation TU104GL [Tesla T4]	
Chip type and count	TU104, 1	
Memory type and size (GB)	GDDR6, 16	
Power draw (W)	70	
Memory module(s)		
Total memory in system (GB)	1,536	
Number of memory modules	24	
Vendor and model	Samsung® M386A8K40BM2-CTD	
Size (GB)	64	
Type	PC4-21300 ECC	
Speed (MHz)	2,666	
Speed running in the server (MHz)	2,666	
Storage controller	Operating system storage	Image dataset storage
Vendor and model	Dell BOSS-S1	Intel SSD DC P4510
Firmware version	2.5.13.3020	VDV10131
Driver version	sd 3.10.0-1062.1.2.el7.x86_64	nvme 1.0

Server configuration information	Dell EMC™ PowerEdge R740xd	
Local storage	Operating system storage	Image dataset storage
Number of drives	2	1
Drive vendor and model	Intel SSD DC S3520	Intel SSD DC P4510
Drive size (GB)	240	8000
Drive information (interface, type)	M2x80mm SSD	PCIe NVME
Network adapter	Data network	Public network
Vendor and model	Mellanox ConnectX-4 Lx	Broadcom NetXtreme II BCM57800
Number and type of ports	2 x 25GbE	4 x 10GbE
Firmware version	14.24.8000	FFV14.10.07
Driver version	Mlx5_core 5.0-0	Bnx2x 1.713.36-0
Cooling fans		
Vendor and model	Nidec Ultraflo 4vpx3-x30 12 VDC, 2.0 Amps 58.2CFM	
Number of cooling fans	6	
Power supplies		
Vendor and model	Dell 5RHVV	
Number of power supplies	2	
Wattage of each (W)	750	

How we tested

Installing CentOS 7 x86_64 18.10

1. Boot the server to the CentOS 7 installation media.
2. Select Install CentOS 7, and press Enter.
3. For the installation language, select English (United States), and click Continue.
4. At the Installation Summary screen, select the following options:
    a. Set the Date and Time settings to the local time zone.
    b. Set the Software Selection to Minimal Install.
    c. Set the Installation Destination as Local Disk and to configure partitioning automatically.
    d. Set the Network and Hostname to DHCP, and turn on the Ethernet device.
5. Click Begin Installation.
6. Enter a root password, and click Done two times.

Upgrading the operating system

1. Boot the server, and log in.
2. Update the operating system:

```
sudo yum -y update
```

3. Reboot the server:

```
sudo reboot
```

Installing utilities

1. Boot the server, and log in.
2. Enable the Extra Packages for Enterprise Linux (EPEL) repository:

```
sudo yum -y install epel-release
```

3. Install the prerequisites:

```
sudo yum -y install git ntp ntpdate openssh-server htop wget curl expect
```

Installing Docker

1. Boot the server, and log in.
2. Install "Development Tools":

```
sudo yum groupinstall -y "Development Tools"
```

3. If you have not yet installed the EPEL repository, enable it:

```
sudo yum -y install epel-release
```

4. Install the prerequisites:

```
sudo yum -y install python python-devel python-pip python3 python3-pip
```

5. Download the Docker installer script:

```
wget --output-file=~/get-docker.sh https://get.docker.com/
```

6. Run the docker installer script:

```
sudo ~/get-docker.sh
```

7. Enable the Docker SystemD service:

```
systemctl enable docker.service
```

8. Start the Docker SystemD service

```
systemctl start docker.service
```

Installing the NVIDIA driver

1. Boot the server, and log in.
2. Download the NVIDIA Tesla T4 driver:

```
wget --output-file=~/NVIDIA-Linux-x86_64-418.87.01.run http://us.download.nvidia.com/tesla/418.87/
NVIDIA-Linux-x86_64-418.87.01.run
```

3. Disable the noveau driver:

```
echo "blacklist nouveau" >/etc/modprobe.d/blacklist-nouveau.conf
echo "options nouveau modeset=0" >> /etc/modprobe.d/blacklist-nouveau.conf
```

4. Rebuild the ramdisk:

```
dracut –force
reboot
```

5. Synchronize the distribution:

```
yum distro-sync -y
```

6. If you have not yet installed "Development Tools," install them:

```
sudo yum groupinstall -y "Development Tools"
```

7. Install the prerequisites:

```
yum -y install perl kernel-devel module-init-tools epel-release dkms
```

8. Install the driver:

```
~/NVIDIA-Linux-x86_64-418.87.01.run --silent
```

9. Ensure that you have created the NVIDIA device files:

```
nvidia-modprobe
```

10. To enable the NVIDIA persistence daemon, add the CRON job:

```
echo "@reboot /usr/bin/nvidia-persistenced" >> /tmp/nvcron
crontab /tmp/nvcron
rm /tmp/nvcron
```

Installing the NVIDIA Container Toolkit

1. Boot the server, and log in.
2. Download the NVIDIA Container Toolkit YUM repository:

```
dist=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L -O /etc/yum.repos.d/nvidia-docker.repo https://nvidia.github.io/nvidia-docker/${dist}/
nvidia-docker.repo
```

3. Update the system:

```
yum -y update
```

4. Install the NVIDIA Container Toolkit:

```
yum -y install nvidia-container-toolkit
```

5. Restart Docker:

```
systemctl restart docker
```

Installing the Docker containers

We used two different Docker containers prebuilt with TensorFlow to run the benchmark. Each container contained tuning and optimization for either a CPU-only or GPU configuration. To install the containers, perform the following steps:

1. Install the NVIDIA-optimized container:

```
docker create --gpus 1 --name=${CONTAINER_NAME} --shm-size=1g --ulimit memlock=-1 --privileged -v
${DATA_DIR}:${DATA_DIR} -i nvcr.io/nvidia/tensorflow:19.10-py3
```

2. Install the Intel MKL-DNN-optimized container:

```
docker create --gpus 1 --name=${CONTAINER_NAME} --shm-size=1g --ulimit memlock=-1 --privileged -v
${DATA_DIR}:${DATA_DIR} -i clearlinux/stacks-dlrs-mkl-vnni:1560
```

Installing the benchmarks

We pulled the benchmark tf_cnn_benchmarks from the TensorFlow Github repository. For GPU testing, we adjusted the batch size to ensure that the GPU memory utilization was near full capacity. For CPU-only testing, we increased the batch size until performance no longer improved—this always resulted in a much larger global batch size across all instances. We also adjusted the run length until the results were consistent. To make these changes, run the following commands inside the Docker container on each server:

1. Open a shell to the docker container:

```
docker exec -it tf_test_gpu /bin/bash
```

2. Download tf_cnn_benchmarks:

```
mkdir -p /tensorflow
cd /tensorflow
git clone https://github.com/tensorflow/benchmarks.git -b master --single-branch
```

3. Initialize the TensorFlow benchmark:

```
nvidia-docker exec ${CONTAINER_NAME} ${PYTHON} -u /tensorflow/benchmarks/scripts/tf_cnn_benchmarks/
tf_cnn_benchmarks.py --device=<cpu or gpu> --model=resnet50 --data_name=imagenet
```

Running the benchmarks

Training testing

CPU-only configuration

We used the horovod variable update method to spread and pin multiple instances of TensorFlow across both CPU sockets. This produced significantly better results than using a single instance or multiple instances with a parameter server.

1. Substitute the appropriate model `(--model=)` and batch size `(--batch_size=)` parameters according to this table:

Model	Batch size
GoogLeNet	256
Inception v4	32
Inception v3	64

2. Run the following commands:

```
COMMON_ARGS="\
--batch_size=<batch size> \
--num_batches=20 \
--num_warmup_batches=10 \
--model=<model> \
--num_inter_threads=2 \
--display_every=5 \
--data_format=NCHW \
--optimizer=momentum \
--device=cpu \
--mkl=true \
--variable_update=horovod \
--horovod_device=cpu \
--local_parameter_device=cpu \
--distortions=true \
--kmp_blocktime=0"

export OMP_NUM_THREADS=6
export CPUS=$(grep processor /proc/cpuinfo | wc -l)
export CORES=$((CPUS/2))
export WORKERS=$((CORES/OMP_NUM_THREADS))
export PPR=$((WORKERS/2))

HOROVOD_FUSION_THRESHOLD=134217728 \
mpirun -np $WORKERS \
--map-by ppr:${PPR}:socket:pe=${OMP_NUM_THREADS} \
--allow-run-as-root \
-H ${HOSTNAME} \
--report-bindings \
--oversubscribe \
-x HOROVOD_FUSION_THRESHOLD \
-x OMP_NUM_THREADS \
python ./tf_cnn_benchmarks.py $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS \
--data_dir=/data/tf_cnn/imagenet-1k-TFRecords/train \
--data_name=imagenet
```

CPU + GPU configuration

We used a single instance of TensorFlow with the default variable update method of a parameter server running on the GPU. This produced slightly better results than using other variable update or parameter server options.

1. Substitute the appropriate model `(--model)` and batch size `(--batch_size)` parameters according to this table:

Model	Batch size
GoogLeNet	512
Inception v4	64
Inception v3	96

2. Run the following command:

```
numactl --cpunodebind=0 --membind=0 python3 ./tf_cnn_benchmarks.py --device=gpu --num_gpus=1 --local_
parameter_device=gpu --data_format=NCHW --model=inception3 --batch_size=96 --distortions=true --data_
name=imagenet --data_dir=/data/tf_cnn/imagenet-1k-TFRecords/train --num_batches=20 --num_warmup_
batches=10 --display_every=5 --optimizer=momentum --summary_verbosity=1
```

Inference testing

CPU-only configuration

We used multiple instances of TensorFlow spread and pinned across both CPU socket,s each with an independent parameter server running on the CPU. This produced significantly better results than using a single instance. We then took the sum of each instance's reported images per second and used that as our result.

1. Substitute the appropriate model `(--model)` and batch size `(--batch_size)` parameters according to this table:

Model	Batch size
GoogLeNet	64
Inception v4	64
Inception v3	64

2. Run the following commands:

```
COMMON_ARGS="\
--batch_size=<batch size> \
--num_batches=100 \
--model=<model> \
--num_inter_threads=2 \
--display_every=10 \
--data_format=NCHW \
--optimizer=momentum \
--device=cpu \
--mkl=true \
--local_parameter_device=cpu \
--distortions=true \
--forward_only=true \
--kmp_blocktime=0"

export OMP_NUM_THREADS=2
export CPUS=$(grep processor /proc/cpuinfo | wc -l)
export CORES=$((CPUS/2))
export WORKERS=$((CORES/OMP_NUM_THREADS))
export PPR=$((WORKERS/2))

mpirun -np $WORKERS \
--map-by ppr:${PPR}:socket:pe=${OMP_NUM_THREADS} \
--allow-run-as-root \
-H ${HOSTNAME} \
--report-bindings \
--oversubscribe \
-x OMP_NUM_THREADS \
python ./tf_cnn_benchmarks.py $COMMON_ARGS \
--num_intra_threads=$OMP_NUM_THREADS \
--data_dir=/data/tf_cnn/imagenet-1k-TFRecords/train \
--data_name=imagenet | tee output.log

awk '/total images\/sec:/{print $3}' output.log
```

CPU + GPU configuration

We used a single instance of TensorFlow with the default variable update method of a parameter server running on the GPU. This produced slightly better results than using other variable update or parameter server options.

1. Substitute the appropriate model `(--model)` and batch size `(--batch_size)` parameters according to this table:

Model	Batch size
GoogLeNet	64
Inception v4	32
Inception v3	128

2. Run the following command:

```
numactl --cpunodebind=0 --membind=0 python3 ./tf_cnn_benchmarks.py --device=gpu --num_gpus=1 --local_
parameter_device=gpu --data_format=NCHW --model=<model> --batch_size=<batch_size> --num_batches=100
--display_every=10 --forward_only=true --distortions=true --data_name=imagenet --data_dir=/data/tf_
cnn/imagenet-1k-TFRecords/train --optimizer=momentum --summary_verbosity=
```

Read the report at http://facts.pt/d4256ck ▶

This project was commissioned by Dell Technologies.

Facts matter.®