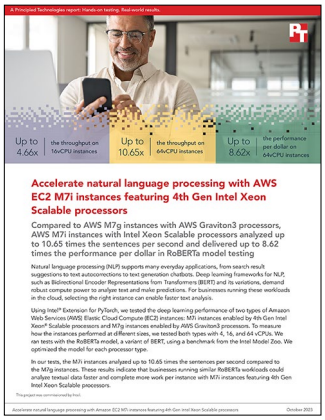

The science behind the report:

Accelerate natural language processing with Amazon EC2 M7i instances featuring 4th Gen Intel Xeon Scalable processors

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 Accelerate natural language processing with Amazon EC2 M7i instances featuring 4th Gen Intel Xeon Scalable processors.

We concluded our hands-on testing on August 10, 2023. 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 August 4, 2023 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 sentences per second.

	4 vCPUs	16 vCPUs	64 vCPUs
BF16, batch size 1			
M7g	4.635	12.083	9.885
M7i	16.439	56.385	105.316
BF16, batch size 32			
M7g	5.699	20.245	52.473
M7i	20.656	92.914	271.718
FP32, batch size 1			
M7g	2.101	6.969	8.713
M7i	3.834	15.076	37.115
FP32, batch size 32			
M7g	2.249	8.701	29.052
M7i	4.349	17.555	56.144

Table 2: Cost, in USD per hour, of each instance type as of August 10, 2023.

	4 vCPUs	16 vCPUs	64 vCPUs
M7g	0.1632	0.6528	2.6112
M7i	0.2016	0.8064	3.2256

Table 3: Performance per cost, in throughput per USD/hour.

	4 vCPUs	16 vCPUs	64 vCPUs
BF16, batch size 1			
M7g	28.401	18.509	3.786
M7i	81.543	69.922	32.650
BF16, batch size 32			
M7g	34.920	31.013	20.095
M7i	102.460	115.221	84.238
FP32, batch size 1			
M7g	12.874	10.676	3.337
M7i	19.018	18.695	11.506
FP32, batch size 32			
M7g	13.781	13.329	11.126
M7i	21.572	21.770	17.406

System configuration information

Table 4: Detailed information on the systems we tested.

System configuration information	m7i.xlarge	m7i.4xlarge	m7i.16xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	08/10/2023	08/10/2023	08/10/2023
CSP / Region	us-east-1f	us-east-1f	us-east-1f
Workload	RoBERTa (Hugging Face)	RoBERTa (Hugging Face)	RoBERTa (Hugging Face)
Workload-specific parameters	max_seq_length: 384 doc_stride: 128	max_seq_length: 384 doc_stride: 128	max_seq_length: 384 doc_stride: 128
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	m7i.xlarge	m7i.4xlarge	m7i.16xlarge
BIOS name and version	Amazon EC2 1.0 10/16/2017	Amazon EC2 1.0 10/16/2017	Amazon EC2 1.0 10/16/2017
Operating system name and version/build number	Amazon Linux 2023 cpe 2.3 6.1.38-59.109.amzn2023	Amazon Linux 2023 cpe 2.3 6.1.38-59.109.amzn2023	Amazon Linux 2023 cpe 2.3 6.1.38-59.109.amzn2023
Date of last OS updates/ patches applied	08/04/2023	08/04/2023	08/04/2023
Processor			
Vendor and model	Intel® Xeon® Platinum 8488C	Intel Xeon Platinum 8488C	Intel Xeon Platinum 8488C
Architecture	x86_64	x86_64	x86_64
Number of processors	1	1	1
Core count per VM	2	8	32
Core frequency (GHz)	3.2	3.2	3.2
Stepping	8	8	8
Hyper-Threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Number of vCPU per VM	4	16	64
Memory			
Total memory in system (GB)	16	64	256
NVMe memory present?	No	No	No
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)	30	30	30
Drive information (speed, interface, type)	gp3, EBS, 3000 IOPS	gp3, EBS, 3000 IOPS	gp3, EBS, 3000 IOPS
Network adapter			
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1x (Up to 12.5 Gbps)	1x (Up to 12.5 Gbps)	1x (25 Gbps)

Table 5: Detailed information on the systems we tested.

System configuration information	m7g.xlarge	m7g.4xlarge	m7g.16xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	08/10/2023	08/10/2023	08/10/2023
CSP / Region	us-east-1f	us-east-1f	us-east-1f
Workload	RoBERTa (Hugging Face)	RoBERTa (Hugging Face)	RoBERTa (Hugging Face)
Workload-specific parameters	max_seq_length: 384 doc_stride: 128	max_seq_length: 384 doc_stride: 128	max_seq_length: 384 doc_stride: 128
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	m7g.xlarge	m7g.4xlarge	m7g.16xlarge
BIOS name and version	Amazon EC2 1.0 11/1/2018	Amazon EC2 1.0 11/1/2018	Amazon EC2 1.0 11/1/2018
Operating system name and version/build number	Amazon Linux 2023 cpe 2.3 6.1.38-59.109.amzn2023	Amazon Linux 2023 cpe 2.3 6.1.38-59.109.amzn2023	Amazon Linux 2023 cpe 2.3 6.1.38-59.109.amzn2023
Date of last OS updates/ patches applied	08/04/2023	08/04/2023	08/04/2023
Processor			
Vendor and model	AWS Graviton3	AWS Graviton3	AWS Graviton3
Architecture	aarch64	aarch64	aarch64
Number of processors	1	1	1
Core count per VM	4	16	64
Core frequency (GHz)	2.6	2.6	2.6
Stepping	r1p1	r1p1	r1p1
Hyper-Threading	No	No	No
Turbo	No	No	No
Number of vCPU per VM	4	16	64
Memory			
Total memory in system (GB)	16	64	256
NVMe memory present?	No	No	No
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)	30	30	30
Drive information (speed, interface, type)	gp3, EBS, 3000 IOPS	gp3, EBS, 3000 IOPS	gp3, EBS, 3000 IOPS
Network adapter			
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1x (Up to 12.5 Gbps)	1x (Up to 12.5 Gbps)	1x (30 Gbps)

How we tested

Testing overview

We tested two types of AWS instances: M7i instances with 4th Gen Intel Xeon Scalable processors, and M7g instances with AWS Graviton3 processors. We ran a RoBERTa workload on the AWS instances to show the performance difference between these instance types in terms of sentences per second.

Creating the AWS instance

1. Log into AWS, and navigate to the AWS Management Console.
2. Click EC2.
3. Click Launch instance, and from the drop-down menu, click Launch instance to open the Launch Instance wizard.
4. On the Name and tags section, enter a name for the instance.
5. On the Application and OS Images section, select the Amazon Linux Quick Start, and then Amazon Linux 2023 AMI.
6. Select the correct Architecture for the OS image to match the instance type, either x86 for m7i or ARM for m7g.
7. On the Instance Type section, select {m7i,m7g}.{xlarge, 4xlarge, 16xlarge}, depending the type of instance you are setting up.
8. Select the appropriate key pair and network settings.
9. On the Storage section, enter 30GB for the size, and gp3 as the type for the Root volume.
10. Verify the settings, and click Launch instance when you are ready.

Configuring Amazon Linux 2023 for RoBERTa benchmark

1. Log into the AWS instance via SSH.
2. Install updates and reboot instance (if necessary):

```
sudo dnf update -y
sudo reboot
```

3. Install new tools:

```
sudo dnf install -y pip git numactl gperftools-libs
```

4. Install python virtual environment tools:

```
pip install virtualenv
```

5. Create and activate python virtual environment for RoBERTa:

```
virtualenv roberta_env
source roberta_env/bin/activate
```

6. Install PyTorch:

```
pip install torch==2.0.1
```

7. Install the PyTorch datasets library:

```
pip install datasets
```

8. Install Intel Extension for PyTorch (x86 architecture only):

```
pip install intel-extension-for-pytorch==2.0.100
```

9. Install additional build tools (needed specifically for transformers v4.10 or older running on aarch64):

```
sudo dnf install -y rust cargo gcc g++
```

Download and build RoBERTa benchmark

1. Clone the Intel Model Zoo git repo and set the MODEL_DIR environment variable:

```
git clone https://github.com/IntelAI/models.git
cd models
git checkout v2.11.1
export MODEL_DIR=$(pwd)
```

2. Clone the Hugging Face Transformers repo in the RoBERTa Base inference directory:

```
cd quickstart/language_modeling/pytorch/roberta_base/inference/cpu
git clone https://github.com/huggingface/transformers.git
cd transformers
git checkout v4.10.0
```

3. Patch the Transformers repo to include IPEX and BF16 support (using patch file included with the Mode Zoo repo and the "fix_mkldnn_for_roberta-base.diff" file included in the Appendix):

```
git apply ../enable_ipex_for_roberta-base.diff
git apply ../fix_mkldnn_for_roberta-base.diff
```

4. Build and install the modified Transformers library:

```
pip install -e ./
cd ../
```

Running the RoBERTa benchmark

1. Log into the AWS instance and run the follow commands to prepare the environment:

```
source roberta_env/bin/activate
cd models
export MODEL_DIR=$(pwd)
export OUTPUT_DIR=${HOME}/output
cd quickstart/language_modeling/pytorch/roberta_base/inference/cpu
```

2. Use run_throughput.sh script to run the benchmark, replacing <BATCHSIZE> with 1 or 32, and <PRECISION> with either fp32 or bf16:

```
BATCH_SIZE=<BATCHSIZE> bash run_multi_instance_throughput.sh <PRECISION>
```

3. Use the following example command to loop through each batch size and precision, and then power off the instance:

```
time for p in fp32 bf16; do for bs in 1 32; do BATCH_SIZE=${bs} bash run_multi_instance_throughput.
sh ${p}; sleep 5 ; done ; done ; sudo poweroff
```

4. To quickly see the throughput in all the output files, you can use a command such as the following:

```
grep Throughput: ~/${OUTPUT_DIR}/throughput_log_*
```

run_throughput.sh:

```
#!/bin/bash
#!/bin/bash
ARGS=""

TIMESTAMP=$(date '+%Y%m%d%H%M%S')
INSTANCE_TYPE=$(TOKEN='curl -s -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-
token-ttl-seconds: 21600"' \
&& curl -H "X-aws-ec2-metadata-token: $TOKEN" -s http://169.254.169.254/latest/meta-data/instance-
type 2> /dev/null)

export DNNL_PRIMITIVE_CACHE_CAPACITY=1024
unset DNNL_DEFAULT_FPMATH_MODE

#export DNNL_VERBOSE=1
export LRU_CACHE_CAPACITY=256
export LD_PRELOAD="/usr/lib64/libtcmalloc.so.4"

export THP_MEM_ALLOC_ENABLE=1

export OMP_DISPLAY_ENV=VERBOSE

if [[ "$(uname -p)" != "aarch64" ]]; then
path="ipex"
ARGS="$ARGS --use_ipex"
echo "### running with intel extension for pytorch"
fi

precision="fp32"
if [[ "$1" == "bf16" ]]
then
    precision="bf16"
    echo "### running bf16 mode"
    if [[ "$(uname -p)" == "aarch64" ]]; then
      export DNNL_DEFAULT_FPMATH_MODE=BF16
    else
      ARGS="$ARGS --bf16"
    fi
elif [[ "$1" == "fp32" ]]
then
    echo "### running fp32 mode"
else
    echo "The specified precision '$1' is unsupported."
    echo "Supported precisions are: fp32, bf16"
    exit 1
fi

mode="jit"
ARGS="$ARGS --jit_mode"
echo "### running with jit mode"

CORES='lscpu | grep Core | awk '{print $4}''
BATCH_SIZE=${BATCH_SIZE:-'expr 4 \* $CORES'}
FINETUNED_MODEL=${FINETUNED_MODEL:-"deepset/roberta-base-squad2"}
if [ -z "${OUTPUT_DIR}" ]; then
  echo "The required environment variable OUTPUT_DIR has not been set, please create the output path and
set it to OUTPUT_DIR"
  exit 1
fi
mkdir -p ${OUTPUT_DIR}

EVAL_SCRIPT=${EVAL_SCRIPT:-"./transformers/examples/pytorch/question-answering/run_qa.py"}

python3 \
  ${EVAL_SCRIPT} $ARGS \
  --model_name_or_path   ${FINETUNED_MODEL} \
  --dataset_name squad_v2 \
  --version_2_with_negative \
  --do_eval \
  --max_seq_length 384 \
  --doc_stride 128 \
  --output_dir ./tmp \
  --per_device_eval_batch_size $BATCH_SIZE \
  --max_eval_samples=$(($(nproc)*128)) \
  | tee ${OUTPUT_DIR}/throughput_log_${path}_${precision}_bs${BATCH_SIZE}_${mode}_${INSTANCE_
TYPE}_${TIMESTAMP}
```

fix_mkldnn_for_roberta-base.diff:

```
--- a/transformers/examples/pytorch/question-answering/trainer_qa.py       2023-08-04
13:10:50.873389137 +0000
+++ b/transformers/examples/pytorch/question-answering/trainer_qa.py       2023-08-05
16:06:18.080151233 +0000
@@ -72,23 +72,15 @@
                     self.model = torch.jit.freeze(self.model)
             else:
                 if bf16:
+                    self.model = mkldnn_utils.to_mkldnn(self.model.to(memory_format=torch.channels_
last), dtype=torch.bfloat16)
                     with torch.cpu.amp.autocast(), torch.no_grad():
-                        self.model = torch.jit.trace(self.model.to(memory_format=torch.channels_last),
jit_inputs, strict=False)
+                        self.model = torch.jit.trace(self.model, jit_inputs, strict=False)
                     self.model = torch.jit.freeze(self.model)
-                    with torch.no_grad():
-                        for _,batch in enumerate(eval_dataloader):
-                            for _,label in enumerate(batch):
-                                if batch[label].dim() >=4:
-                                    batch[label]=batch[label].to(memory_format=torch.channels_last)
                 else:
+                    self.model = mkldnn_utils.to_mkldnn(self.model.to(memory_format=torch.channels_
last), dtype=torch.float32)
                     with torch.no_grad():
-                        self.model = torch.jit.trace(self.model.to(memory_format=torch.channels_last),
jit_inputs, strict=False)
+                        self.model = torch.jit.trace(self.model, jit_inputs, strict=False)
                     self.model = torch.jit.freeze(self.model)
-                    with torch.no_grad():
-                        for _,batch in enumerate(eval_dataloader):
-                            for _,label in enumerate(batch):
-                                if batch[label].dim() >=4:
-                                    batch[label]=batch[label].to(memory_format=torch.channels_last)
         else:
             if use_ipex:
                 if bf16:
@@ -100,9 +92,9 @@
                     for _,batch in enumerate(eval_dataloader):
                         for _,label in enumerate(batch):
                             batch[label]=batch[label].to(torch.bfloat16)
-                    self.model = mkldnn_utils.to_mkldnn(self.model, dtype=torch.bfloat16)
+                    self.model = mkldnn_utils.to_mkldnn(self.model.to(memory_format=torch.channels_
last), dtype=torch.bfloat16)
                 else:
-                    self.model = mkldnn_utils.to_mkldnn(self.model)
+                    self.model = mkldnn_utils.to_mkldnn(self.model.to(memory_format=torch.channels_
last), dtype=torch.float32)

         with torch.autograd.profiler.profile(
             enabled=profile,
```

Read the report at https://facts.pt/RfrK3Rr

This project was commissioned by Intel.

Principled Technologies®

Facts matter.®