WHITE PAPER

Artificial Intelligence
Intel® AI Builders

# Amazing Inference Performance with Intel® Xeon® Scalable Processors

**Over the past year, Intel has focused on optimizing popular deep learning frameworks and primitives for Intel® Xeon® processors. Now, in addition to being a common platform for inference workloads, Intel Xeon Scalable processor (formerly codename Skylake-SP) is a competitive platform for both training and inference.**


**Authors**

**Andres Rodriguez**
Senior Principal Engineer, Data Center Group

**Wei Li**
Vice President, Core and Visual Computing Group, and General Manager, Machine Learning and Translation

**Jason Ye**
Engineering Manager, Machine and Translation

**Rinat Rappoport**
Software Engineering Manager, Machine Learning and Translation

**Eden Segal**
Software Engineer, Machine Learning and Translation

**Koichi Yamada**
Senior Principal Engineer, Core and Visual Computing Group

**Niharika Maheshwari**
Deep Learning Software Engineer, Machine and Translation

**Rong Zhang**
Software Engineer, Core and Visual Computing Group/Developer Products Division/Machine Learning and Translation

**Etay Meiri**
Software Engineer, Software and Services Group

**Amit Bleiweiss**
Director of AI Workloads, Artificial Intelligence Products Group

**Zhiyuan Huang**
Deep Learning Software Engineer, Machine Learning and Translation


Previously, deep learning training and inference on CPUs took an unnecessarily long time because the software was not written to take full advantage of the hardware features and functionality. That is no longer the case. The Intel Xeon Scalable processor with optimized software has demonstrated enormous performance gains for deep learning compared to the previous generation without optimized software – up to 198x for inference and 127x for training.[1] This applies to various types of models including multi-layer perceptron (MLP), convolutional neural networks (CNNs), and the various types of recurrent neural networks (RNNs). The performance gap between GPUs and CPUs for deep learning training and inference has narrowed, and for some workloads, CPUs now have an advantage over GPUs.

For machine translation which uses RNNs, the Intel Xeon Scalable processor outperforms NVidia* V100* by 4x on the AWS Sockeye Neural Machine Translation (NMT) model with Apache* MXNet* when Intel® Math Kernel Library (Intel MKL) is used. Details to replicate this result are shown below.

The experiments were conducted using the servers at Amazon Web Services (AWS*) with the publicly available Apache MXNet framework (a neutral framework maintained by neither Intel nor Nvidia*). The benchmark is the AWS Sockeye, an open source project for NMT. For price parity, the c5.18xlarge and p3.2xlarge instances were used–both have the exact on-demand price of $3.06 in the U.S. regions (as of the date of this publication) and similar spot-pricing. The c5.18xlarge instance offers a 2-socket Intel Xeon Platinum processor with 72 vCPUs. The p3.2xlarge instance offers a V100* GPU and 8 vCPUs.

The experiments on the p3.2xlarge instance used the Amazon Deep Learning AMI (Ubuntu) [tested on Version 8.0 (ami-dff741a0)] and the following set up steps:

The experiments on the p3.2xlarge instance used the Amazon Deep Learning AMI (Ubuntu) [tested on Version 8.0 (ami-dff741a0)] and the following set up steps:

Step 1: Activate mxnet_p36 environment and python3 *(Note that Sockeye requires python3)*

```
$ source activate mxnet_p36
$ easy_install pip
```

Step 2: Install sockeye *(Note: the sockeye version must match the pretrained model)*

```
$ pip3 install sockeye==1.16.2
```

Step 3: Install mxnet with CUDA-9.0

```
$ pip3 install mxnet-cu90==1.1.0.post0
```

Step 4: Get pretrained model for inference

Download pretrained model `wmt_model.bz2` from https://drive.google.com/drive/folders/1kLGvYT_fGabliSolkxcCzjDIrN4Hf-6e

```
$ mkdir nmt && tar -jxvf wmt_model.bz2 -C nmt
```

Step 5: Inference (Translate from German to English)

```
$ cd nmt
$ python3 -m sockeye.translate -m wmt_model -i newstest2016.tc.BPE.de -o newstest2016.tc.BPE.en --batch-size 64
--output-type benchmark
```

The experiments on the c5.18xlarge instance used the Amazon Deep Learning AMI (Ubuntu) and the following set up steps.

Step 1: Install Sockeye

```
$ sudo apt install virtualenv
$ virtualenv -p python3 ~/virtu_nmt
$ source ~/virtu_nmt/bin/activate
$ pip3 install sockeye==1.16.2
```

*(Note: the sockeye version must match the pretrained model)*

```
$ pip3 uninstall mxnet
```

*(Note: will use mxnet-mkl instead of MXNet installed by sockeye)*

Step 2: Install MXNet and MKL

```
$ pip3 install mxnet-mkl==1.2.0b20180507
```

Step 3: Get pretrained model for inference

Download pretrained model `wmt_model.bz2` from: https://drive.google.com/drive/folders/1kLGvYT_fGabliSolkxcCzjDIrN4Hf-6e

```
$ mkdir nmt && tar -jxvf wmt_model.bz2 -C nmt
$ cd nmt/
```

Step 4: Inference (Translate from German to English)

a. Run without MKL:

```
$ pip uninstall mxnet-mkl
$ pip install mxnet=1.1.0.post0
$ python3 -m sockeye.translate -m wmt_model -i newstest2016.tc.BPE.de -o newstest2016.tc.BPE.en --batch-size 64 --output-type benchmark --use-cpu
```

b. Run with MKL using the default threading setting

```
$ pip3 install mxnet-mkl==1.2.0b20180507
$ python3 -m sockeye.translate -m wmt_model -i newstest2016.tc.BPE.de -o newstest2016.tc.BPE.en --batch-size 64 --output-type benchmark --use-cpu
```

c. Run with MKL controlling threads to physical cores binding:

```
$ export KMP_AFFINITY=granularity=fine,noduplicates,compact,1,0
$ export OMP_NUM_THREADS=36
$ python3 -m sockeye.translate -m wmt_model -i newstest2016.tc.BPE.de -o newstest2016.tc.BPE.en --batch-size 64 --output-type benchmark --use-cpu
```

d. Run with 6 workers running on 6 cores each

```
$ sudo apt install numactl
$ export KMP_AFFINITY=granularity=fine,noduplicates,compact,1,0
$ export OMP_NUM_THREADS=6
$ taskset -c 0-5 numactl -l python3 -m sockeye.translate -m wmt_model -i newstest2016.tc.BPE.de -o newstest2016.tc.BPE.en --batch-size 64 --output-type benchmark --use-cpu |& tee log1.64.log & taskset -c 6-11 numactl -l python3 -m sockeye.translate -m wmt_model -i newstest2016.tc.BPE.de -o newstest2016.tc.BPE.en --batch-size 64 --output-type benchmark --use-cpu |& tee log2.64.log & taskset -c 12-17 numactl -l python3 -m sockeye.translate -m wmt_model -i newstest2016.tc.BPE.de -o newstest2016.tc.BPE.en --batch-size 64 --output-type benchmark --use-cpu |& tee log3.64.log & taskset -c 18-23 numactl -l python3 -m sockeye.translate -m wmt_model -i newstest2016.tc.BPE.de -o newstest2016.tc.BPE.en --batch-size 64 --output-type benchmark --use-cpu |& tee log4.64.log & taskset -c 24-29 numactl -l python3 -m sockeye.translate -m wmt_model -i newstest2016.tc.BPE.de -o newstest2016.tc.BPE.en --batch-size 64 --output-type benchmark --use-cpu |& tee log5.64.log & taskset -c 30-35 numactl -l python3 -m sockeye.translate -m wmt_model -i newstest2016.tc.BPE.de -o newstest2016.tc.BPE.en --batch-size 64 --output-type benchmark --use-cpu |& tee log6.64.log
```

Results (sentences per second) across different batch sizes are as follows:

| Batch size | c5.18xlarge (Skylake) | | | | p3.2xlarge (V100) |
|---|---|---|---|---|---|
| | NO MKL | MKL | MKL/OMP/KMP | MKL/OMP/KMP/6 workers | pre-install |
| 1 | 2.7 | 8.1 | 7.5 | 39.8 | 9.4 |
| 2 | 5.5 | 11.7 | 11.3 | 52.5 | 13.9 |
| 8 | 7.5 | 15.2 | 16.3 | 85 | 20.1 |
| 16 | 7.8 | 18.1 | 19.7 | 93.1 | 22.2 |
| 32 | 7.9 | 20.1 | 21.4 | 92.9 | 22.8 |
| 64 | 7.8 | 21.2 | 22.5 | 83.9 | 23.2 |
| 128 | 7.5 | 20.9 | 22.3 | 75.5 | 22.9 |

These results demonstrate the gains of using Intel MKL with Intel Xeon processors. In addition, properly setting the environment variables gives additional performance and provides comparable performance to V100 (22.5 vs 23.2 sentences per second). The environmental variable `OMP_NUM_THREADS` is set to the numbers of cores available for the workload (up to 36 in the c5.18xlarge instance). The environmental variable `KMP_AFFINITY` is set to granularity=fine,noduplicates,compact,1,0 to bind threads to a specific hardware thread context. An additional 4x performance can be observed by running multiple inference instances in parallel with each instance using a subset of the cores. In this experiment, 6 cores per inference workload were used and a total of 6 inference workloads thereby using all 36 cores. Executing multiple inference workloads across CPU cores is a simple command line and does not require modifying the frameworks.

In addition to these gains, additional optimizations are coming soon that we expect will further improve CPU performance. These include 1) optimizing an LSTM kernel and adding that API to sockeye, 2) optimizing beam search, 3) reducing the batch size when a sentence is finished, 4) pre-calculating the embedding's matrix multiply, and 5) lower numerical precision inference.

The sockeye model currently does not use an Intel MKL LSTM primitive. The performance can improve by adding this primitive and optimizing the various operations of the primitive in Intel MKL.

Optimizing beam search includes concatenating various matrices across the beams to compute all the beam searches in parallel, and more efficiently computing the top beam values using the Intel® AVX-512 instructions available in the Intel Xeon Scalable processor family. Intel AVX-512 features vectors of 512 bits, which can store 16 float values. For beam-width smaller than 16, the vector can be used to store all of the top probabilities for each sentence. When stored as a sorted array, the Intel AVX-512 vector can be masked to insert new values to the array. The details of this approach will be released when these optimizations are added to the frameworks.

Dynamically reducing the batch size when a sentence has finished can reduce the computational requirements. In the current MXNet release, the amount of work done in a batch is determined by the longest sentence. In our implementation that we plan to release in a future date, when a sentence has been translated it stops going through the decoder. This reduces the amount of work needed in the following iterations.

In conclusion, Intel Xeon Scalable processors along with optimized deep learning functions in the Intel MKL and Intel MKL-DNN libraries provide competitive performance for deep learning workloads (in addition to classical machine learning and other AI algorithms). Additional optimizations are coming soon that will further improve this performance.