A performance comparison of Deep Learning frameworks on KNL

R. Zanella, G. Fiameni, M. Rorro
Middleware, Data Management - SCAI - CINECA

IXPUG Bologna, March 5, 2018
1. Problem description

2. Tested Neural Networks

3. Architectures and Software

4. Single node benchmark

5. Multinode benchmark

6. Conclusions
Given a single input, a trained neural network is able to predict a distribution of probability:

$$\hat{y} = f(x; \theta)$$
Given a single input, a trained neural network is able to predict a distribution of probability:

\[ \hat{y} = f(x; \theta) \]

NN parameters are chosen in order to minimize the average error on a given training set \( \{x^{(i)}, y^{(i)}\}_{i=1,\ldots,N} \):

\[
J(\theta) = \frac{1}{N} \sum_{1}^{N} L \left( f(x^{(i)}; \theta), y^{(i)} \right)
\]
Given a single input, a trained neural network is able to predict a distribution of probability:

\[ \hat{y} = f(x; \theta) \]

NN parameters are chosen in order to minimize the average error on a given training set \( \{ x^{(i)}, y^{(i)} \}_{i=1,...,N} \):

\[ J(\theta) = \frac{1}{N} \sum_{1}^{N} L \left( f(x^{(i)}; \theta), y^{(i)} \right) \]

A Stochastic Gradient Descent (SGD) algorithm step is:

\[ \theta^{(k+1)} = \theta^{(k)} - \epsilon_k \hat{g}^{(k)} \]

where \( \hat{g}^{(k)} = \frac{1}{n} \nabla_{\theta} \sum_{l \in \text{batch}} L(f(x^{(l)}; \theta^{(k)}), y^{(l)}) \), \( n = \# \text{batch} \).
Convolutional Neural Networks

ImageNet Large Scale Visual Recognition Competition (ILSVRC):
- annual software contest (since 2010);
- tasks: image classification, object localization/detection, scene detection;

Tested Networks:
- AlexNet: winner of 2012 classification task,
- Overfeat: winner of 2013 localization task, remarkable results also in classification and detection
- VGG: winner of 2014 localization task, second place on classification

Russakovsky et al., *ImageNet Large Scale Visual Recognition Challenge, 2015*
AlexNet (Krizhevsky, Sutskever, Hinton, 2012)

Structure:

- five convolutional layers:
  - kernel sizes $11 \times 11$, $5 \times 5$, three $3 \times 3$;
  - output channels 64, 192, 384, 256, 256;
- three max-pool layers (kernel $3 \times 3$, stride $2 \times 2$);
- three fully-connected layers;
- Rectified Linear Unit (ReLU) as nonlinear activation function.
AlexNet (Krizhevsky, Sutskever, Hinton, 2012)

Structure:
- **five** convolutional layers:
  - kernel sizes 11 × 11, 5 × 5, three 3 × 3;
  - output channels 64, 192, 384, 256, 256;
- **three** max-pool layers (kernel 3 × 3, stride 2 × 2);
- **three** fully-connected layers;
- Rectified Linear Unit (ReLU) as nonlinear activation function.

A note on benchmarked network:
- original version exploited multiple GPUs, to overcome memory limits;
- benchmarked version: AlexNet_v2 (one device),

Structure (fast model):

- **five** convolutional layers
  - kernel sizes $11 \times 11$, $5 \times 5$, three $3 \times 3$;
  - output channels 96, 256, 512, 1024, 1024;
- **three** max-pool layers (kernel $2 \times 2$, stride $2 \times 2$);
- **three** fully-connected layers;
- Rectified Linear Unit (ReLU) as nonlinear activation function.

---

Structure (fast model):

- five convolutional layers
  - kernel sizes $11 \times 11$, $5 \times 5$, three $3 \times 3$;
  - output channels 96, 256, 512, 1024, 1024;
- three max-pool layers (kernel $2 \times 2$, stride $2 \times 2$);
- three fully-connected layers;
- Rectified Linear Unit (ReLU) as nonlinear activation function

Differences w.r.t. AlexNet:

- remarkable size increase of output channels of convolutional layers (larger number of filters);
- no overlap on max-pool layers;

---

Structure (vgg_a/vgg11 network):

- **eight** convolutional layers
  - fixed kernel size $3 \times 3$;
  - output channels $64, 128, 2 \times 256, 4 \times 512$;
- **five** max-pool layers (kernel $2 \times 2$, stride $2 \times 2$);
- **three** fully-connected layers;
- Rectified Linear Unit (**ReLU**) as nonlinear activation function

Simonyan, Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2014
Structure (vgg_a/vgg11 network):

- eight convolutional layers
  - fixed kernel size $3 \times 3$;
  - output channels 64, 128, $2 \times 256$, $4 \times 512$;
- five max-pool layers (kernel $2 \times 2$, stride $2 \times 2$);
- three fully-connected layers;
- Rectified Linear Unit (ReLU) as nonlinear activation function

Differences w.r.t. AlexNet:

- slight size increase of output channels of convolutional layers;
- remarkable depth increase (conv layers: five $\rightarrow$ eight);
- smaller kernels;
- no overlap on max-pool layers;

Simonyan, Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014
CPU architectures:

- **2x Intel Broadwell**, 2x Intel Xeon E5-2697 v4@2.3GHz, 36 cores (total), 128 GB RAM
- **Intel Knights Landing**, Intel Xeon Phi7250 @1.4GHz, 68 cores, 96 GB RAM (+16 GB MCDRAM)
- **2x Intel Skylake**, 2x Intel Xeon 8160 @2.1GHz, 2x Intel Xeon 8160 @2.1GHz, 48 cores (total), 192 GB RAM
Systems description

CPU architectures:
- **2x Intel Broadwell**, 2x Intel Xeon E5-2697 v4@2.3GHz, 36 cores (total), 128 GB RAM
- **Intel Knights Landing**, Intel Xeon Phi7250 @1.4GHz, 68 cores, 96 GB RAM (+16 GB MCDRAM)
- **2x Intel Skylake**, 2x Intel Xeon 8160 @2.1GHz, 2x Intel Xeon 8160 @2.1GHz, 48 cores (total), 192 GB RAM

GPU architectures¹:
- **Nvidia K80**: 5.6 Tflops peak performance (sp), 24 GB RAM
- **Nvidia P100 (PCIe)**: 10.6 Tflops peak performance (sp), 16 GB RAM

¹ hosted on: Intel Haswell, 2x Intel Xeon 2630 v3 @2.4GHz, 16 cores (total), 128 GB RAM
Systems description

CPU architectures:
- 2x Intel Broadwell, 2x Intel Xeon E5-2697 v4@2.3GHz, 36 cores (total), 128 GB RAM
- Intel Knights Landing, Intel Xeon Phi7250 @1.4GHz, 68 cores, 96 GB RAM (+16 GB MCDRAM)
- 2x Intel Skylake, 2x Intel Xeon 8160 @2.1GHz, 2x Intel Xeon 8160 @2.1GHz, 48 cores (total), 192 GB RAM

GPU architectures¹:
- Nvidia K80: 5.6 Tflops peak performance (sp), 24 GB RAM
- Nvidia P100 (PCIe): 10.6 Tflops peak performance (sp), 16 GB RAM

Knights Landing settings:
- Hyper-Threading is enabled;
- Memory Mode is cache;
- Cluster Mode is quadrant;
- Network type: Intel Omnipath, 100 Gb/s

¹ hosted on: Intel Haswell, 2x Intel Xeon 2630 v3 @2.4GHz, 16 cores (total), 128 GB RAM
Caffe

theano

TensorFlow

Tested software
Caffe

- Berkeley Vision and Learning Center (BVLC)
- C++, Matlab, Python APIs
- prototxt text file for network definition
- GPU: CUDA, cuDNN, (+nccl for single-node multi-gpu)
- CPU: BLAS implementation (ATLAS, MKL, or OpenBlas)
- tested configuration: BVLC/Caffe 1.0.0, CUDA 8.0, cuDNN 6.0
- Berkeley Vision and Learning Center (BVLC)
- C++, Matlab, Python APIs
- prototxt text file for network definition
- GPU: CUDA, cuDNN, (+nccl for single-node multi-gpu)
- CPU: BLAS implementation (ATLAS, MKL, or OpenBlas)
- tested configuration: BVLC/Caffe 1.0.0, CUDA 8.0, cuDNN 6.0

- Intel branch of original project
- prerequisites: mkl-dnn (based on mklml)
- tested configuration: intel/Caffe 1.0.0, mkl-dnn 0.9
- Berkeley Vision and Learning Center (BVLC)
- C++, Matlab, Python APIs
- prototxt text file for network definition
- GPU: CUDA, cuDNN, (+nccl for single-node multi-gpu)
- CPU: BLAS implementation (ATLAS, MKL, or OpenBlas)
- tested configuration: BVLC/Caffe 1.0.0, CUDA 8.0, cuDNN 6.0

- Intel branch of original project
- prerequisites: mkl-dnn (based on mklml)
- tested configuration: intel/Caffe 1.0.0, mkl-dnn 0.9

Projects status:
- ✗ BVLC/Caffe: project closed with 1.0 (18 Apr 2017), development efforts moved to Caffe2;
- ✔ Intel/Caffe: latest release is 1.1.0 (13 Jan 2018).
• Montreal Institute for Learning Algorithms (MILA)
• Python API (dynamic C code generation)
• prerequisites: CUDA, cuDNN, libgpuarray
• tested configuration: MILA/Theano (git: 12/07/2017), CUDA 8.0, cuDNN 6.0, libgpuarray 0.6.8
Montreal Institute for Learning Algorithms (MILA)

Python API (dynamic C code generation)

prerequisites: CUDA, cuDNN, libgpuarray

tested configuration: MILA/Theano (git: 12/07/2017), CUDA 8.0, cuDNN 6.0, libgpuarray 0.6.8

Intel branch of original project

prerequisites: MKL

tested configuration: intel/Theano 1.1, MKL 2017
Montreal Institute for Learning Algorithms (MILA)

- Python API (dynamic C code generation)
- prerequisites: CUDA, cuDNN, libgpuarray
- tested configuration: MILA/Theano (git: 12/07/2017), CUDA 8.0, cuDNN 6.0, libgpuarray 0.6.8

Intel branch of original project

- prerequisites: MKL
- tested configuration: intel/Theano 1.1, MKL 2017

Projects status:

- Mila/Theano: development closed, latest bug fix is 1.0.1 (7 Dec 2017);
- Intel/Theano: latest release is 1.1.0 (1 Apr 2017).
- Intel (previously Nervana Systems)
- Python API
- prerequisites: CUDA
- tested configuration: Neon 2.1.0, CUDA 8.0, (mklml_lnx_2018)
• Intel (previously Nervana Systems)
• Python API
• prerequisites: CUDA
• tested configuration: Neon 2.1.0, CUDA 8.0, (mklml_lnx_2018)

Projects status:
✓ Nervana/Neon: latest release is 2.6.0 (5 Jan 2018).
• Google
• Python API
• prerequisites: CUDA,
• tested configuration:
  ▶ CPU: TensorFlow (git: 18/07/2017), (mklml_lnx_2018)
  ▶ GPU: Tensorflow 1.2.1, CUDA 8.0

- Google
- Python API
- prerequisites: CUDA,
- tested configuration:
  - CPU: TensorFlow (git: 18/07/2017), (mklml_lnx_2018)
  - GPU: Tensorflow 1.2.1, CUDA 8.0

Project status:
✓ tensorflow/tensorflow: latest release 1.6.0 (28 Feb 2018)

## Software summary

<table>
<thead>
<tr>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel/Caffe</td>
<td>BVLC/Caffe</td>
</tr>
<tr>
<td>Intel/Theano</td>
<td>MILA/Theano</td>
</tr>
<tr>
<td>TensorFlow</td>
<td></td>
</tr>
<tr>
<td>Neon</td>
<td></td>
</tr>
</tbody>
</table>
Network structures are simplified:
- no preprocessing phase is implemented;
- input data is in-memory generated;
- no dropout is considered;
- classification task: 1000 classes.
Network structures are simplified:
- no preprocessing phase is implemented;
- input data is in-memory generated;
- no dropout is considered;
- classification task: 1000 classes.

Network source codes:
- Caffe/Theano/Tensorflow: convnet-benchmarks source code is used;
- Neon: code is in Neon sources.
Network structures are simplified:
- no preprocessing phase is implemented;
- input data is in-memory generated;
- no dropout is considered;
- classification task: 1000 classes.

Network source codes:
- Caffe/Theano/Tensorflow: convnet-benchmarks source code is used;
- Neon: code is in Neon sources.

Chosen performance measure: number of images per second, considering:
- forward step: the evaluation of the network on a batch (inference);
- forward-backward step: fwd step + the backpropagation of the errors on a batch (training).
OverFeat (forward) vs OverFeat (forward-backward)

- **OverFeat (forward)**: The training time for forward pass is shown.
- **OverFeat (forward-backward)**: The training time for both forward and backward passes is shown.

### Hardware Configurations
- **2x Broadwell**: Knights Landing
- **2x Skylake**: K80, P100

### Software Environments
- **BVLC/caffe**: Theano, neon, intel/caffe, tensorflow
- **MILA/theano**: neon, intel/caffe, tensorflow

### Data Points
- **128**, **256**, **512**, **1024**, **2048**

### Performance Metrics
- **P100**: Performance metric for each configuration.
VGG (I)

VGG (forward)

VGG (forward-backward)
VGG (II)

VGG (forward)

VGG (forward-backward)
Based on official TensorFlow benchmark suite: https://github.com/tensorflow/benchmarks

- input data can be in-memory generated, or from dataset (+preprocessing);
- contains definitions of fully functional neural networks (dropout is present);
- support for multinode (and multinode, multi-GPU runs);
- support for larger (and growing) number of models;

but the support is for TensorFlow only.

Goyal et al., Accurate, Large Minibatch SGD Training ImageNet in 1 Hour, 2017
Based on official TensorFlow benchmark suite: https://github.com/tensorflow/benchmarks

- input data can be in-memory generated, or from dataset (+preprocessing);
- contains definitions of fully functional neural networks (dropout is present);
- support for multinode (and multinode, multi-GPU runs);
- support for larger (and growing) number of models;

but the support is for TensorFlow only.

Multi node parallelization approach: given $n = \#\text{batch}, M = \#\text{nodes}$

$$\theta^{(k+1)} = \theta^{(k)} - \epsilon_k \hat{g}^{(k)}$$

where

$$\hat{g}^{(k)} = \frac{1}{Mn} \nabla_{\theta} \sum_{m \in \text{nodes}} \sum_{l \in \text{batch}_m} L(f(x^{(l)}; \theta^{(k)}), y^{(l)})$$

---

Goyal et al., Accurate, Large Minibatch SGD Training ImageNet in 1 Hour, 2017
Supported communication protocols

TensorFlow supported protocols:
- **gRPC**: google Remote Procedure Call
- **gRPC+VERBS**:
  - gRPC for administrative tasks (set up RDMA path),
  - RDMA (Remote Direct Memory Access) for actual tensors (weights, gradients, etc) exchange.
- **gRPC+MPI**: [not tested yet]

---

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/verbs/README.md
Supported communication protocols

TensorFlow supported protocols:
- gRPC: google Remote Procedure Call
- gRPC+VERBS:
  - gRPC for administrative tasks (set up RDMA path),
  - RDMA (Remote Direct Memory Access) for actual tensors (weights, gradients, etc) exchange.
- gRPC+MPI: [not tested yet]

Supported variable management procedures:
- parameter server: variables are stored on a parameter server that holds the master copy of the variable. For each step, each wn gets a copy of the variables from the ps, and sends its gradients to the ps;
- distributed replicated: wn has a copy of the variables, and updates its copy after the ps are all updated with the gradients from all wn [not tested yet].

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/verbs/README.md
Supported communication protocols

TensorFlow supported protocols:

- **gRPC**: google Remote Procedure Call
- **gRPC+VERBS**:
  - gRPC for administrative tasks (set up RDMA path),
  - RDMA (Remote Direct Memory Access) for actual tensors (weights, gradients, etc) exchange.
- **gRPC+MPI**: [not tested yet]

Supported variable management procedures:

- **parameter server**: variables are stored on a parameter server that holds the master copy of the variable. For each step, each wn gets a copy of the variables from the ps, and sends its gradients to the ps;
- **distributed replicated**: wn has a copy of the variables, and updates its copy after the ps are all updated with the gradients from all wn [not tested yet].

Chosen performance measure: average number of images per second per node, considering:

- fwd step + the backpropagation of the errors on a batch (training);
- fixed local batch size comparison: overall batch size grows with nodes.

---

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/verbs/README.md
gRPC (forward-backward, VGG)

VGG (fwd-back), protocol: grpc, dedicated ps

![Graph](image_url)
gRPC+VERBS (forward-backward, VGG)

VGG (fwd-back), protocol: grpc+verbs, dedicated ps

VGG (fwd-back), protocol: grpc+verbs, ps+wn
Conclusions

Single node benchmarks:

- Best combination is Nvidia P100 and Neon.
- Nodes composed by a single Intel Knights Landing or 2x intel Skylake can achieve comparable, or slightly superior performances to a single Nvidia K80.
- Concerning CPU systems, no software outperforms the others.
- TensorFlow exhibit good performances in both architectures.
Conclusions

Single node benchmarks:

- Best combination is Nvidia P100 and Neon.
- Nodes composed by a single Intel Knights Landing or 2x intel Skylake can achieve comparable, or slightly superior performances to a single Nvidia K80.
- Concerning CPU systems, no software outperforms the others.
- TensorFlow exhibit good performances in both architectures.

Multinode benchmarks (TensorFlow only):

- RDMA (gRPC+VERBS) is fundamental for acceptable efficiency.
Conclusions

Single node benchmarks:
- Best combination is Nvidia P100 and Neon.
- Nodes composed by a single Intel Knights Landing or 2x intel Skylake can achieve comparable, or slightly superior performances to a single Nvidia K80.
- Concerning CPU systems, no software outperforms the others.
- TensorFlow exhibit good performances in both architectures.

Multinode benchmarks (TensorFlow only):
- RDMA (gRPC+VERBS) is fundamental for acceptable efficiency.

Ongoing work:
- Caffe and Theano can exploit Intel Machine Learning Scaling Library (MLSL);
- TensorFlow can exploit also MPI.
References:

- Russakovsky et al., *ImageNet Large Scale Visual Recognition Challenge*, 2015
- Simonyan, Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2014
- Goyal et al., *Accurate, Large Minibatch SGD Training ImageNet in 1 Hour*, 2017

We would like to thank:

- Andrea Luiselli (Intel), for support on new Intel software branches;
- Walter Riviera (Intel), for multinode TensorFlow support.
### Net structures recap

<table>
<thead>
<tr>
<th>AlexNet</th>
<th>OverFeat</th>
<th>VGG_a (vgg11)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input RGB image</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv: k 11×11, ch 64, stride 4</td>
<td>conv: k 11×11, ch 96, stride 4</td>
<td>conv: k 3×3, ch 64, stride 1</td>
</tr>
<tr>
<td>maxpool: k 3×3, stride 2</td>
<td>maxpool: k 2×2, stride 2</td>
<td>maxpool: k 2×2, stride 2</td>
</tr>
<tr>
<td>conv: k 5×5, ch 192, stride 1</td>
<td>conv: k 5×5, ch 256, stride 1</td>
<td>conv: k 3×3, ch 128, stride 1</td>
</tr>
<tr>
<td>maxpool: k 3×3, stride 2</td>
<td>maxpool: k 2×2, stride 2</td>
<td>maxpool: k 2×2, stride 2</td>
</tr>
<tr>
<td>conv: k 3×3, ch 384, stride 1</td>
<td>conv: k 3×3, ch 512, stride 1</td>
<td>conv: k 3×3, ch 256, stride 1</td>
</tr>
<tr>
<td>conv: k 3×3, ch 256, stride 1</td>
<td>conv: k 3×3, ch 1024, stride 1</td>
<td>conv: k 3×3, ch 256, stride 1</td>
</tr>
<tr>
<td>maxpool: k 3×3, stride 2</td>
<td>maxpool: k 2×2, stride 2</td>
<td>maxpool: k 2×2, stride 2</td>
</tr>
<tr>
<td>FC: output 4096</td>
<td>FC: output 3072</td>
<td>FC: output 4096</td>
</tr>
<tr>
<td>FC: output 4096</td>
<td>FC: output 4096</td>
<td>FC: output 4096</td>
</tr>
<tr>
<td>FC: output 1000</td>
<td>FC: output 1000</td>
<td>FC: output 1000</td>
</tr>
</tbody>
</table>

**Table:** Convolutional and fully-connected layers are followed by ReLU nonlinear function.
Not familiar with applications, so the legend on the graphs is unclear. Unless you want to compare across FWD and FWD-BWD, I would put the graphs on separate slides so that you can increase their sizes. I hope the talk includes a brief discussion about how the applications are different. It isn’t clear how the work is being distributed and there are many gaps in the results.

This seems a nice benchmark for a relevant case study. I suggest to improve the quality of the slide 6: what is on the y axis? Too many columns make the comparison difficult: I suggest to split the plots. On slide 7: capitalize Intel;
You state that only TensorFlow supports MPI. Does that mean that all of the applications are running OpenMP-only for this work? Single-node benchmark results are obtained with applications running OpenMP-only. Multinode TensorFlow resides on gRPC or gRPC+VERBS.

Why aren’t there results for 4096 NEON? Why are the GPU results missing 4096? Why are half of the GPU results missing 2048? K80 has a total of 24 GB of memory: for large batch sizes, all SDKs run out of memory.

Were any of the results surprising? Intel efforts on CPU based Deep Learning allows TensorFlow to request comparable cpu time either on KNL or on K80.

Are there remaining challenges or questions to be answered?

Comment on the parallelization of Neon: multithreading? All tested SDKs are exploiting multithreading parallelization.