Deep Learning with Intel DAAL on Knights Landing Processor

David Ojika
dave.n.ojika@cern.ch

March 22, 2017
Outline

• Introduction and Motivation
• Intel Knights Landing Processor
• Intel Data Analytics and Acceleration Library (DAAL)
• Experiment and current progress

GitHub: https://github.com/davenso/DKN
Object Identification

- Object identification is the task of **training** a computer to **recognize** patterns in data
  - **Object**: could be car, person, a piece of flower, higgs, muons, etc.
  - **Learning algorithm** (e.g. decision tree) is used to for training
Machine Learning

- Involves two steps:
Deep Learning

• Deep learning means using **neural networks** (a class of machine learning) with **multiple hidden layers**
  • Neural networks are modelled based on the **dynamics of the neurons** in our brain
  • *Hidden layers* represent neural computations in series of *processing stages*
  • Learning performance can generally improve with depth of network (at more cost to processing)
DNN Applications

- **Handwriting**
  - 7 Layers
  - Error rate: 15.3%
  - MNIST, 2012

- **Images**
  - GoogleNet
  - 22 layers
  - Error rate: 6.67%
  - ILSVR 2014 Winner

- **ImageNet**
  - 34 layers to > 1,000 layers
  - Error rate: 3.57%
  - ILSVR 2015 Winner

- **Tiger**
  - ILSVR 2014 Winner
  - Error rate: 6.67%
  - ILSVR 2015 Winner
DNN Training

- Back propagation and forward propagation method used training

10 Neurons
2 Hidden Layers
DNN Inferencing

A trained network with:
- 10 Neurons
- 2 Hidden Layers

Layer 1

Learned Weights (10 x 5)

Learned Biases (1 x 5)

Layer 2

Learned Weights (5 x 2)

Learned Biases (1 x 2)
Challenge with DNN Training

• Large size of dataset
  • Gigabytes, Terabytes of data

• Large number of hyper-parameters
  • # layers, # neurons, batch size, iterations, learning rate, loss function, weight decay etc.
    • Hyperparameter optimization techniques: random search, grid search, Bayesian, gradient-based

• Emerging hardware for deep learning
  • GPU
  • KNL
  • FPGA

• Software exists, but require manual fine-tuning
(Physics) Object identification: A Cross-layer Perspective

• Compose hardware, algorithm and software and components
• Derive efficient FPGA implementation to perform inference
Intel Knights Landing (KNL)

• Next generation Xeon Phi (after Knight Corner (KNC) co-processor)
  • **Self-boot:** unlike KNC
  • **Binary-compatible** with Intel Architecture (IA) and boots standard OS

• Some performance-enhancing features
  • Vector processors: AVX-512
  • High-bandwidth: MCDRAM
  • Cluster modes
  • Networking (in some models)
KNL Overview

- Chip: 36 Tiles interconnected by 2D Mesh
- Tile: 2 Cores + 2 VPU/core + 1 MB L2
- Memory: MCDRAM: 16 GB on-package; High BW
- DDR4: 6 channels @ 2400 up to 384GB
**KNL AVX-512**

- 512-bit FP/Integer Vectors
- 32 registers, & 8 mask registers
- Gather/Scatter

![SIMD Mode](http://geekswithblogs.net/akraus1/archive/2014/04/04/155858.aspx)

- 32 registers, & 8 mask registers
- Gather/Scatter

![AVX-512 Chart](http://geekswithblogs.net/akraus1/archive/2014/04/04/155858.aspx)
KNL MCDRAM

- MCDRAM as regular memory
- Managed by SW

- MCDRAM as cache
- No code change required*

- MCDRAM part cache, part memory; 25% or 75% as cache

```c
float *fv;
fv = (float *)malloc(sizeof(float) * 100);
```

```c
float *fv;
fv = (float *)hbw_malloc(sizeof(float) * 100);
```

```c
numactl -m 1 ./myProgram
```
KNL Cluster Modes

- All-to-all: uniformly distributed address
- Quadrant: four vertical quadrants
- Sub-NUMA Clustering (SNC): each quadrant as separate NUMA domain
Intel Data Analytics and Acceleration Library (DAAL)

- Optimized functions for deep learning and classical machine learning
- Language API for C++, Java and Python for Linux and Windows
- Support data ingress from Hadoop and Spark
- Free and open-source versions available
Intel DAAL

- **Layer**: NN building block
- **Model**: Set of layers
- **Optimization**: Objective function / solver
- **Topology**: NN description
- **NN**: Topology, model & optimization algorithm
- **Tensor**: Multidimensional data structure

<table>
<thead>
<tr>
<th>Common layers</th>
<th>Activation</th>
<th>Normalization</th>
<th>Optimization / Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolutional</td>
<td>Logistic</td>
<td>Z-score</td>
<td>MSE</td>
</tr>
<tr>
<td>Pooling (max, average)</td>
<td>Hyperbolic tangent</td>
<td>Batch</td>
<td>Cross entropy</td>
</tr>
<tr>
<td>Fully connected</td>
<td>ReLU, pReLU, smooth ReLU</td>
<td>Local response</td>
<td>Mini batch SGD</td>
</tr>
<tr>
<td>Dropout</td>
<td>Softmax</td>
<td></td>
<td>Stochastic LBFGS</td>
</tr>
<tr>
<td></td>
<td>Abs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DAAL API Example

• Layer
  
  `SharedPtr<layers::fullyconnected::Batch> > fcLayer1(new fullyconnected::Batch<>(20));`

• Topology

  `SharedPtr<layers::fullyconnected::Batch> > fcLayer1(new fullyconnected::Batch<>(20));
  Collection<LayerDescriptor> configuration;
  configuration.push_back(LayerDescriptor(0, fcLayer1, NextLayers(1)));`

• Optimization Solver

  `services::SharedPtr<optimization_solver::mse::Batch<double> > mseObjectiveFunction(new optimization_solver::mse::Batch<double>(nVectors));
  optimization_solver::sgd::Batch<> sgdAlgorithm(mseObjectiveFunction);`

• Model

  `trainingNet.compute();
  ... ... ...`

  `services::SharedPtr<training::Model> tModel = trainingNet.getResult()-&gt;get(model)
  services::SharedPtr<prediction::Model> pModel = tModel-&gt;getPredictionModel();`
Higgs Classification

Data
- 11 million events \textit{(Monte Carlo simulations)}
  - 21 low-level features from particle detector
  - 7 high-level features (hand-crafted)
  - “1”: signal; “0”: background
  - A binary classification problem
- Training set: 10.5 million
- Validation set: 500 thousand

DNN
- Started with a topology with 3 layers (28-1024-2)
- Hyper-parameter: began with Random Search with minimal optimization effort

Model Development (DAAL)
- Our “simulation” environment

KLH
- MCDRAM mode = Flat
- Cluster mode = Quadrant
DNN Topology
Preliminary Results

```
[davido@knl-data dkn]$ build/neural_net_dense_batch.exe ~/dataset/higgs/8g
Training started...
Prediction started...
Neural network classification results (first 20 observations):
Ground truth    Neural network predictions: each class probability
0               0.639  0.361
1               0.278  0.722
1               0.291  0.709
0               0.468  0.532
0               0.469  0.531
1               0.626  0.374
0               0.430  0.570
1               0.452  0.548
1               0.366  0.634
0               0.513  0.487
1               0.554  0.446
0               0.664  0.336
0               0.570  0.430
0               0.453  0.547
1               0.653  0.347
1               0.224  0.776
0               0.360  0.640
0               0.379  0.621
0               0.397  0.603
0               0.573  0.427
```

code: git clone https://github.com/davenso/DKN

Use and improve!
Discussions and Conclusions

- Performance can greatly enhance with:
  - Deeper network topology
  - Better hyper-parameters

- Deep neural networks are capable of learning underlying features, and should therefore generalize well, e.g. Higgs, Muon, etc.
Current Developments

- Exploration of **more complex models** and hyper-parameter optimization techniques (beyond Random Search)
- Integration of “real” **muon data** and performance benchmarking
- **Tuning of KNL hardware** to improve runtime performance of DNN training
- Implementation of distributed DNN algorithm, utilizing **multiple KNL nodes** for training
- Exploration of alternative algorithm (likely as a ‘**hybrid model**’), e.g. Decision Forests
Thank You, UF Team

• Sergei Gleyzer
• Brendan Regnery
• Darin Acosta
• Ann Gordon-Ross
• Pranav Goswami
• Andrew Carnes
• Erik Deumens
• Jon Akers
• UF RC
Image credits

- CERN