AI JOURNEY WITH INTEL WORKSHOP

Tuesday, September 17, 12:00 - 2:00 pm
399 Julis Romo Rabinowitz Building
(Free event and lunch is included.)

Why attend:

This workshop will teach you how you can take advantage of your Intel® hardware platforms using with Intel® Optimized frameworks and software offerings without much of a change in your workload.

What you will learn:

This workshop is aimed at providing the attendees a full overview of Intel® AI Software offerings. You will learn how to take advantage of Intel® optimized Tensorflow, and learn the steps of deploying on different hardware with Intel® OpenVino™ Toolkit. Lastly, Intel® DAAL, HPAT, and many other libraries will be introduced and their performance will be shown on a given workload.

Agenda:

- ✓ Intel® AI Portfolio
- ✓ AI Journey with Intel®
- ✓ Intel® Optimized Tensorflow
- ✓ Intel® AI Libraries
- ✓ Introduction to Intel® OpenVino™ Toolkit
- ✓ Optimizations and Performance Comparisons

CPU TENSORFLOW

conda create --name tf-cpu --channel anaconda tensorflow

```
python                    3.7.4                h265db76_1    anaconda
tensorflow                1.14.0          mkl_py37h45c423b_0    anaconda
```

conda create --name tf-cpu --channel intel tensorflow

```
intelpython               2019.5                        0    intel
intel-tensorflow          1.14.0                    pypi_0    pypi
```

conda create --name IDP --channel intel intelpython3_full

```
daal                      2019.5                intel_281    intel
daal4py                   2019.5           py36ha68da19_2    intel
intelpython               2019.5                        0    intel
intel-tensorflow          1.14.0                    pypi_0    pypi
```

Anaconda vs. Intel Python

```
(base) [jdh4@della5 ~]$ conda activate tf-cpu-intel
(tf-cpu-intel) [jdh4@della5 ~]$ python
Python 3.6.9 |Intel Corporation| (default, Sep 11 2019, 16:40:08)
[GCC 4.8.2 20140120 (Red Hat 4.8.2-15)] on linux
Type "help", "copyright", "credits" or "license" for more information.
Intel(R) Distribution for Python is brought to you by Intel Corporation.
Please check out: https://software.intel.com/en-us/python-distribution
```

```
(base) [jdh4@della5 ~]$ module load anaconda3
(base) [jdh4@della5 ~]$ python
Python 3.7.3 (default, Mar 27 2019, 22:11:17)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
```

Intel® Data Analytics Acceleration Library (DAAL)

https://software.intel.com/en-us/get-started-with-daal-for-linux

This library helps reduce the time it takes to develop high-performance data science applications. Enable applications to make better predictions faster and analyze larger data sets with available compute resources.

- **Includes highly optimized machine learning and analytics functions**
- Simultaneously ingests data and computes results for highest throughput performance
- Supports batch, streaming, and distribution use models to meet a range of application needs
- Use the same API for application development on multiple operating systems
- C++, Java and Python APIs
- Multi-threaded

FASTER MACHINE LEARNING

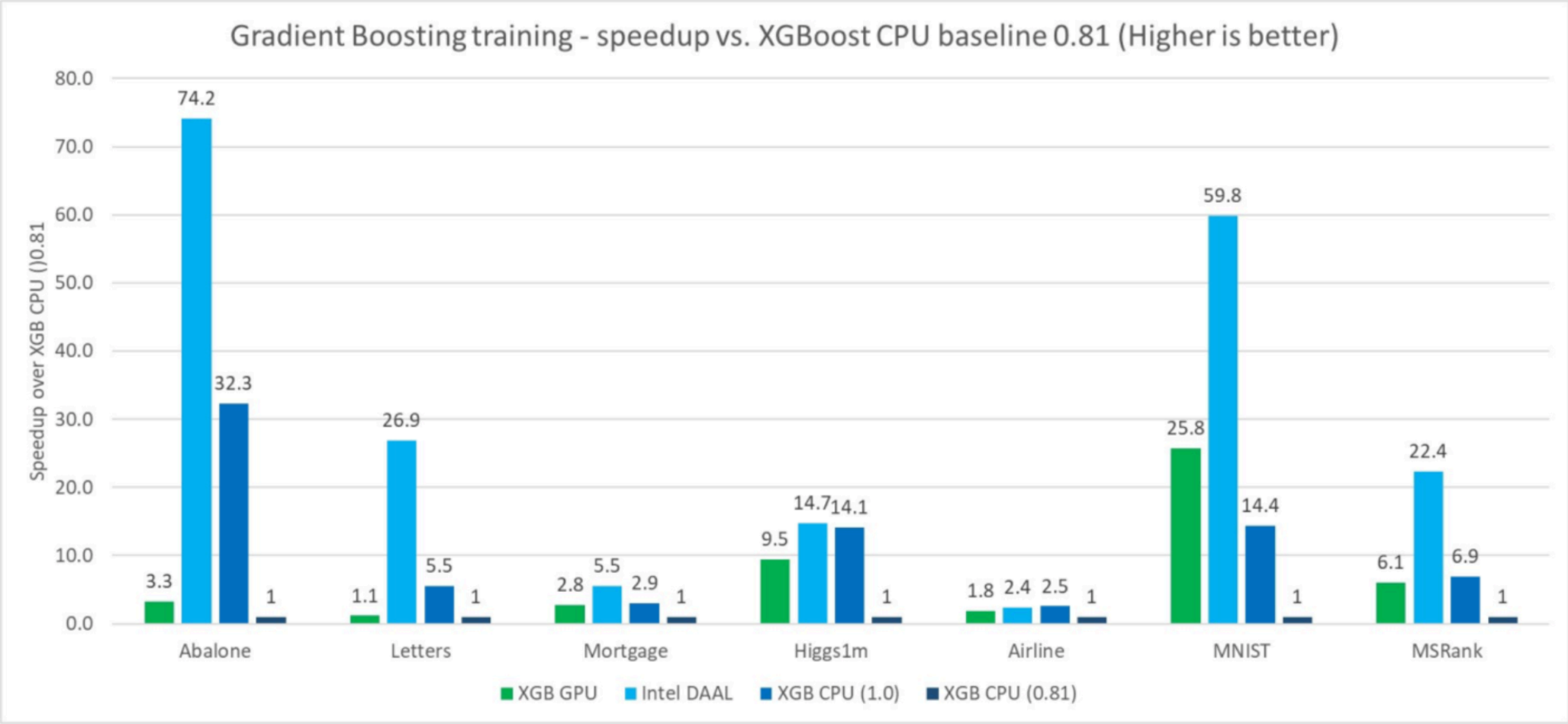
WITH SCIKIT-LEARN AND INTEL® DAAL

SCIKIT-LEARN BENEFITS AND FEATURES

- Scikit-Learn is a mature python package with hundreds of algorithms with different configuration parameters each.
- Intel® DAAL has a different set of algorithms and sometimes implementations use slightly different variants of the algorithm.

```
import daal4py as d4p
import numpy as np
import pandas as pd
nrows_train = 1000000
nrows_test = 1000000
from time import time

start = time()
print("Reading data set...")
data = pd.read_csv("HIGGS.csv.gz", delimiter=",", header=None, compression=
end = time()
print("Reading data set takes {0} seconds.".format(end-start))

print("Pre-processing data set...")
start = time()
data = data[list(data.columns[1:])+list(data.columns[0:1])]
n_features = data.shape[1]
train_data = data[range(1,n_features)][:nrows_train]
train_label = data[range(0, 1)][:nrows_train]
test_data = data[range(1,n_features)][nrows_train:nrows_train+nrows_test]
test_label = data[range(0, 1)][nrows_train:nrows_train+nrows_test]
n_classes = len(np.unique(train_label))
```

```
end = time()
print("Pre-processing data set takes {0} seconds.".format(end-start))

start = time()
# Configure a training object
train_algo = d4p.gbt_classification_training(**daal_params)
print("Training data set....")
train_result = train_algo.compute(train_data, train_label)
end = time()
print("Training data set takes {0} seconds.".format(end-start))

print("Predicting test set...")
start = time()
predict_algo = d4p.gbt_classification_prediction(n_classes)
predict_result = predict_algo.compute(test_data, train_result.model)
end = time()
print("Predicting test data set takes {} seconds.".format(end-start))
```

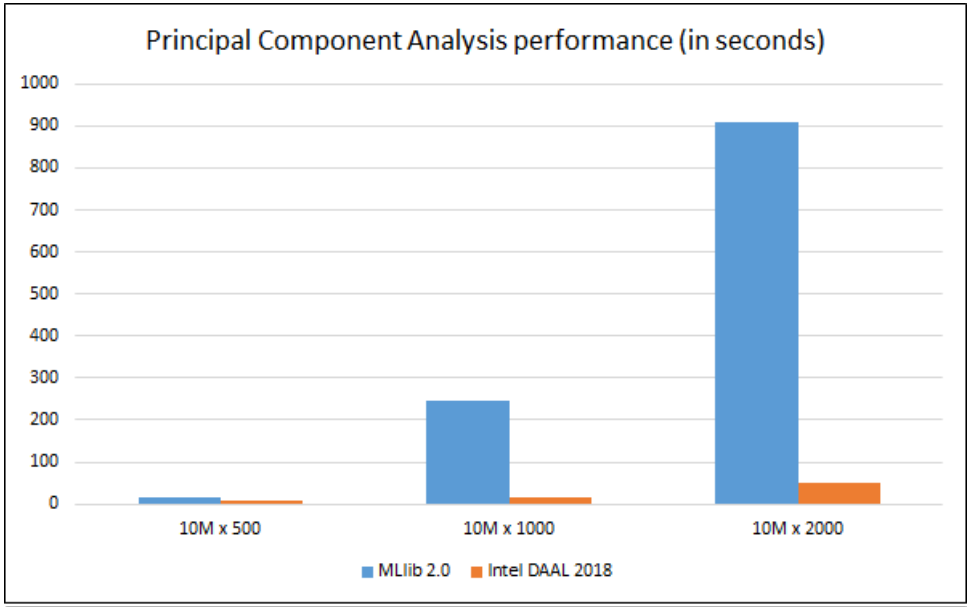

Gradient Boosting training - speedup vs. XGBoost CPU baseline 0.81 (Higher is better)
Speedup over XGB CPU ()0.81
80.0
70.0
60.0
50.0
40.0
30.0
20.0
10.0
0.0
3.3
74.2
32.3
1
1.1
26.9
5.5
1
2.8
5.5
2.9
1
9.5
14.7
14.1
1
1.8
2.4
2.5
1
25.8
59.8
14.4
1
6.1
22.4
6.9
1
Abalone
Letters
Mortgage
Higgs1m
Airline
MNIST
MSRank
XGB GPU
Intel DAAL
XGB CPU (1.0)
XGB CPU (0.81)

conda create --name intel-sklearn scikit-learn -c intel

```
package                    |            build
---------------------------|-----------------
bzip2-1.0.6                |               18         100 KB  intel
certifi-2018.1.18          |           py36_2         143 KB  intel
daal-2019.5                |        intel_281        65.3 MB  intel
daal4py-2019.5             |   py36ha68da19_2        12.3 MB  intel
icc_rt-2019.5              |        intel_281         9.7 MB  intel
impi_rt-2019.5             |        intel_281        20.7 MB  intel
intel-openmp-2019.5        |        intel_281         888 KB  intel
intelpython-2019.5         |                0           3 KB  intel
joblib-0.13.2              |           py36_1         364 KB  intel
mkl-2019.5                 |        intel_281       205.4 MB  intel
mkl-service-2.3.0          |           py36_0         211 KB  intel
mkl_fft-1.0.14             |   py36ha68da19_1         303 KB  intel
mkl_random-1.0.4           |   py36ha68da19_2         454 KB  intel
numpy-1.17.0               |  py36ha68da19_13          64 KB  intel
numpy-base-1.17.0          |          py36_13         7.4 MB  intel
openssl-1.0.2s             |                0         2.2 MB  intel
pip-19.1.1                 |           py36_0         1.9 MB  intel
python-3.6.9               |                5        24.6 MB  intel
scikit-learn-0.21.3        |   py36ha68da19_4         8.2 MB  intel
scipy-1.3.1                |   py36ha68da19_2        22.8 MB  intel
setuptools-41.0.1          |           py36_0         611 KB  intel
six-1.12.0                 |           py36_0          27 KB  intel
sqlite-3.28.0              |                0         2.3 MB  intel
tbb-2019.8                 |        intel_281         874 KB  intel
tbb4py-2019.8              |     py36_intel_0         255 KB  intel
tcl-8.6.4                  |               24         1.3 MB  intel
tk-8.6.4                   |               29         1.1 MB  intel
wheel-0.31.0               |           py36_3          62 KB  intel
xz-5.2.4                   |                5         333 KB  intel
zlib-1.2.11                |                5          95 KB  intel
```

DAAL and Apache Spark

```
// import org.apache.spark.mllib.feature.{PCA, PCAModel}
import daal_for_mllib.{PCA, PCAModel}
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Matrix
import org.apache.spark.mllib.linalg.distributed.RowMatrix
val data    = sc.textFile("/Spark/PCA/data/PCA.txt")
val dataRDD = data.map(s => Vectors.dense(s.split(' ').map(_.toDouble))).cache()
val model = new PCA(10).fit(dataRDD)
println("Principal components:" + model.pc.toString())
```



Configuration info - Versions: Intel® Data Analytics Acceleration Library (Intel® DAAL) 2018; Hardware: Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz, Sockets 2x10, 128GB RAM, 7 nodes; OS CentOS 6.5; HT = ON; Apache Spark 2.0, Scala 2.11

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchase, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark

Intel® High Performance Analytics Toolkit (HPAT)

A compiler-based framework for big data in Python

High Performance Analytics Toolkit (HPAT) scales analytics/ML codes in Python to bare-metal cluster/cloud performance automatically. It compiles a subset of Python (Pandas/Numpy) to efficient parallel binaries with MPI, requiring only minimal code changes. HPAT is orders of magnitude faster than alternatives like Apache Spark.

https://github.com/IntelPython/hpat

HPAT knows how to handle pandas!

In [1]:
```
import hpat
import pandas as pd
import numpy as np

@hpat.jit
def calc_pi(n):
    xy = pd.DataFrame({'x': 2 * np.random.ranf(n) - 1,
                       'y': 2 * np.random.ranf(n) - 1})
    pi = 4 * xy[xy.x**2 + xy.y**2 < 1].x.count() / n
    return pi

%timeit calc_pi(2**22)
```

```
77.6 ms ± 5.29 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
import hpat
import numpy as np
import time

@hpat.jit
def calc_pi(n):
    t1 = time.time()
    x = 2 * np.random.ranf(n) - 1
    y = 2 * np.random.ranf(n) - 1
    pi = 4 * np.sum(x**2 + y**2 < 1) / n
    print("Execution time:", time.time()-t1, "\nresult:", pi)
    return pi

calc_pi(2 * 10**8)
```

```
mpiexec -n 8 python pi.py
```

SUMMARY

- Consider installing TensorFlow from the Intel channel if running on CPU-only machines
- MKL-DNN is now DNNL
- Scikit-Learn and Spark users may benefit from DAAL
- Pandas users may benefit from HPAT
- Intel XGBoost