AI SOFTWARE OPTIMIZATIONS ON INTEL
BREAKING BARRIERS BETWEEN THEORY AND REALITY

Partner with Intel to accelerate your AI journey

Tame your data deluge with our data layer expertise

Choose any approach from analytics to deep learning

Simplify AI via our robust community

Speed up development with open AI software

Deploy AI anywhere with unprecedented HW choice

Scale with confidence on the platform for IT & cloud

Choose any approach from analytics to deep learning

Simplify AI via our robust community

Tame your data deluge with our data layer expertise

Speed up development with open AI software

Deploy AI anywhere with unprecedented HW choice

Scale with confidence on the platform for IT & cloud

www.intel.ai
SPEED UP DEVELOPMENT
with open AI software

### LIBRARIES
- **Data scientists**
  - **Python**
    - Scikit-learn
    - Pandas
    - NumPy
  - **R**
    - Cart
    - Random Forest
    - Mahout
  - **Distributed**
    - MLLib (on Spark)

### DEEP LEARNING FRAMEWORKS
- Optimized for CPU & more
- Status & installation guides
- More framework optimizations underway (e.g. PaddlePaddle*, CNTK* & more)

### DEEP LEARNING GRAPH COMPILER
- **Intel® nGraph™ Compiler (Beta)**
  - Open source compiler for deep learning model computations optimized for multiple devices (CPU, GPU, NNP) from multiple frameworks (TF, MXNet, ONNX)

### KERNELS
- **Library developers**
  - **Intel® Distribution for Python**
    - Intel distribution optimized for machine learning
  - **Intel® Data Analytics Library**
    - Intel® Data Analytics Acceleration Library (incl machine learning)
  - **Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN)**
    - Open source DNN functions for CPU / integrated graphics

---

1. An open source version is available at: 01.org/openvinotoolkit
2. Other names and brands may be claimed as the property of others.
3. All products, computer systems, dates, and figures are preliminary based on current expectations, and are subject to change without notice.

© 2019 Intel Corporation  Optimization Notice
INTEL DISTRIBUTION FOR PYTHON
Introduction to Python* Performance

General Python behavior (Cpython)

- Cpython provides an interpreter to run commands from Python Bytecode (.pyc)
- Compiling doesn't go down to x86 instructions, but instead
- Python interpreter → Compiled Bytecode → Python Virtual Machine
- Allows for very flexible bytecode, and the Python interpreter is the main ingredient
- Cpython and PyPy have a Global Interpreter Lock (GIL)
Why does this matter? (Python layers)

- Example with array loops
- GIL will force loops to run in a single threaded fashion
- NumPy* dispatch helps get around single-threaded by using C functions
- C functions can then call processor vectorization

Getting out of Python layer is key for performance
Introduction to Python* Performance, cont.

The layers of quantitative Python

- The Python language is interpreted and has many type checks to make it flexible
- Each level has various tradeoffs; NumPy* value proposition is immediately seen
- For best performance, escaping the Python layer early is best method

Enforces Global Interpreter Lock (GIL) and is single-threaded, abstraction overhead. No advanced types.

Gets around the GIL (multi-thread and multi-core) BLAS API can be the bottleneck
*Basic Linear Algebra Subprograms (BLAS) [CBLAS]

Gets around BLAS API bottleneck
Much stricter typing
Fastest performance level
Dispatches to hardware vectorization

Intel® MKL included with Anaconda* standard bundle; is Free for Python
Performance of Python

Python + Numba*
http://numba.pydata.org/

Small %% performance gap

C

LLVM-based compiler
Multiple threading runtimes

Optimizing compiler
OpenMP*/TBB/pthreads

```
@numba.jit(nopython=True, parallel=True)
def logistic_regression(Y, X, w0, step, iterations):
    """SGD solver for binary logistic regression."""
    w = w0.copy()
    for i in range(iterations):
        w += step * np.dot((1.0/(1.0 + np.exp(Y * np.dot(X, w)))) * Y, X)
    return w
```

# Accelerate libraries with Intel® Distribution for Python*

**High Performance Python* for Scientific Computing, Data Analytics, Machine Learning**

## FASTER PERFORMANCE

<table>
<thead>
<tr>
<th>Performance Libraries, Parallelism, Multithreading, Language Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerated NumPy/SciPy/scikit-learn with Intel® MKL(^1) &amp; Intel® DAAL(^2)</td>
</tr>
<tr>
<td>Data analytics, machine learning with scikit-learn, pyDAAL</td>
</tr>
<tr>
<td>Optimized run-times Intel MPI*, Intel® TBB</td>
</tr>
<tr>
<td>Scale with Numba* &amp; Cython*</td>
</tr>
<tr>
<td>Includes optimized mpi4py, works with Dask* &amp; PySpark*</td>
</tr>
<tr>
<td>Optimized for latest Intel® architecture</td>
</tr>
</tbody>
</table>

## GREATER PRODUCTIVITY

<table>
<thead>
<tr>
<th>Prebuilt &amp; Accelerated Packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prebuilt &amp; optimized packages for numerical computing, machine/deep learning, HPC &amp; data analytics</td>
</tr>
<tr>
<td>Conda build recipes included in packages</td>
</tr>
<tr>
<td>Free download &amp; free for all uses including commercial deployment</td>
</tr>
</tbody>
</table>

## ECOSYSTEM COMPATIBILITY

<table>
<thead>
<tr>
<th>Supports Python 2.7 &amp; 3.6, conda, pip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compatible &amp; powered by Anaconda*, supports conda &amp; pip</td>
</tr>
<tr>
<td>Distribution &amp; individual optimized packages also available through Anaconda*</td>
</tr>
<tr>
<td>Intel MKL accelerated Numpy, and scipy now in Anaconda!</td>
</tr>
<tr>
<td>Optimizations upstreamed to main Python trunk</td>
</tr>
<tr>
<td>Commercial support through Intel® Parallel Studio XE 2018</td>
</tr>
</tbody>
</table>

**Intel® Architecture Platforms**

**Operating System**: Windows*, Linux*, MacOS\(^1\)*

---

\(^1\)Intel® Math Kernel Library  
\(^2\)Intel® Data Analytics Acceleration Library
Productivity with Performance via Intel® Python*

Intel® Distribution for Python*

Data acquisition & preprocessing

Numerical/Scientific computing & machine learning

Composable multi-threading

Distributed parallelism

Learn More: software.intel.com/distribution-for-python

Intel® Distribution Python* Distribution Channels

- **Standalone Installer**

- **Open-source Channels**
  - Anaconda Cloud
  - apt-get
  - Docker

- **Intel Software Tools suite**
  - Parallel Studio XE
  - System Studio

Available on Google Cloud Platform: Deep Learning Images

https://blog.kovalevskyi.com/deeplearning-images-revision-m9-intel-optimized-images-273164612e93
Installing Intel® Distribution for Python* 2019

**Standalone Installer**

**Anaconda.org**
Anaconda.org/intel channel
> conda config --add channels intel
> conda install intelpython3_full
> conda install intelpython3_core

**PyPI**

**Docker Hub**
docker pull intelpython/intelpython3_full

**YUM/APT**
Access for yum/apt:
Close to native code Umath Performance with Intel Python 2019

Compared to Stock Python packages on Intel® Xeon processors

87%

native efficiency on

Black-Scholes Formula code with Intel numpy + numba.

Configuration: Stock Python: python 3.6.6 hc3d631a_0 installed from conda, numpy 1.15, numba 0.39.0, llvmlite 0.24.0, scipy 1.1.0, scikit-learn 0.19.2 installed from pip;Intel Python: Intel Distribution for Python 2019 Gold: python 3.6.5 intel_11, numpy 1.14.3 intel_py36_5, mkl 2019.0 intel_101, mkl_fft 1.0.2 intel_mp114py36_5, mkl_random 1.0.1 intel_mp114py36_43; OS: CentOS Linux 7.3.1611, kernel 3.10.0-514.el7.x86_64; Hardware: Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz (2 sockets, 18 cores/socket, HT:off), 256 GB of DDR4 RAM, 16 DIMMs of 16 GB@2666MHz.

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 purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks. Source: Intel Corporation - performance measured in Intel labs by Intel employees. Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804.
Intel® MKL: Python* Integration

Python usage

Intel® MKL included in Intel® Distribution of Python*
Numpy accelerated out of the box
No code changes

What MKL brings to Python

Single-Core: vectorization, prefetching, cache utilization
→ SIMD support for AVX-512 ISA

Multi-Many Core (processor/socket) level parallelization
→ OpenMP and TBB support

Multi-Socket (node) level parallelization & Clusters scaling

---

### Requires No Python Code Changes

```python
# Calculate with Numpy
import numpy as np
result = np.cov(fullArray, rowvar=False, bias=True)

# Calculate with Scikit-learn
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(npa)
result = pca.get_covariance()
```
Accelerating K-Means

Performance speedups for Intel® Distribution for Python* scikit-learn on Google Cloud Platform's 96 vCPU instance Intel® Xeon™ Processors

- PCA-based: 23X faster
- Random: 21X faster
- K-means clustering algorithms: 22X faster

**System Configuration:**
GCP VM, zone us-central1-c; 96 vCPU, Intel Skylake; 360 GB memory. Ubuntu 16.04.3 LTS; Linux instance-1 4.10.0-38-generic #42~16.04.1-Ubuntu SMP Tue Oct 10 16:32:20 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux Intel® Distribution for Python* from Docker image intelpython/intelpython3_full:latest (created 2017-09-12T20:10:42:86269635592); Stock Python*: pip install scikit-learn

Performance Against Native* Code

Intel optimizations improve NumPy & SciPy FFT efficiency closer to native code speeds on Intel® Core™ processors

Performance results are based on testing as of July 9, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

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 purchases, including the performance of that product when combined with other products. For more complete information, see Performance Benchmark Test Disclosure.

Testing by Intel as of July 9, 2018. Configuration: Stock Python: python 3.6.6 nf.c3d331a_0 installed from conda, numpy 1.15, numba 0.39.0, lvmrte 0.24.0, scipy 1.1.0, scikit-learn 0.19.2 installed from pip; Intel Python: Intel® Distribution for Python* 2019 Gold: python 3.6.5 intel_1.1, numpy 1.1.3 intel_py36_5, mkl 2019.0 intel_101, mkl_intel_t01, mkl_intel_t02, intel_py36_3, 64M RAM, Intel® Core™ i7-7567U CPU @ 3.50GHz, 32GB of 2084 SDRAM, 2 DMMS at 15.66GHz.

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #201710034.

For more complete information about compiler optimizations, see our Optimization Notice.
Performance Against Native* Code, cont.

Intel optimizations improve Python Linear Algebra efficiency closer to native code speeds on Intel® Xeon™ processors

Performance results are based on testing as of July 9, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

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 these factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information, see Performance Benchmark Test Disclosure.

Testing by Intel as of July 9, 2018. Configuration: Stock Python: python 3.6.6 h-c3d031a: 0 installed from conda, numpy 1.15, numba 0.39,0, Bmin: 0.24.0, scipy 1.1.0, scikit-learn 0.19.2 installed from pips; Intel Python: Intel® Distribution for Python* 2019 Gold; python 3.6.5 intel_11, numpy 1.14.3 intel_py36 3.5, mkl 2019.0 intel 102, mkl 1.0.2 intel_mp134py36 6, numba 0.39.0, intel_mp114py36 6, Bmin: 0.24.0 intel_py36 0, scikit-1.1.0 intel_mp114py36 6, scikit-learn 0.19.1 intel_mp14py36 3; OS: CentOS Linux 7.3.1611, kernel 3.10.0-317.el7.x86_64; Intel Xeon® Gold 6140 CPU @ 2.3GHz (2x sockets, 18 core/socket, HT off), 256 GB of DDR4 RAM, 16 DMEM of 16 Gb GDDR5 5500 MHz. Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20119624.

Optimization Notice.

Copyright © 2018, Intel Corporation. All rights reserved.

Other names and brands may be claimed as the property of others.
INTEL® DATA ANALYTICS ACCELERATION LIBRARY (INTEL® DAAL)

Included in Intel® Parallel Studio and Intel® Distribution for Python*

Also available as Standalone Version (includes priority support)
Faster Machine Learning & Analytics with Intel® DAAL

- Features highly tuned functions for classical machine learning and analytics performance across spectrum of Intel® architecture devices
- Optimizes data ingestion together with algorithmic computation for highest analytics throughput
- Includes Python*, C++, and Java* APIs and connectors to popular data sources including Spark* and Hadoop*
  - New High-level Python API, daal4py introduced
  - Out-of-box acceleration for key scikit-learn* algorithms
- Free and open source community-supported versions are available, as well as paid versions that include premium support.

What's New in 2018 and 2019

- New Algorithms:
  - Logistic regression
  - Classification & Regression GBT
  - Classification & Regression Decision Forest
  - Spark* MLlib-compatible API wrappers for easy substitution of faster Intel DAAL functions
- Improved APIs for ease of use
- Repository distribution via PIP, Conda YUM and APT

Pre-processing
- Decompression, Filtering, Normalization

Transformation
- Aggregation, Dimension Reduction

Analysis
- Summary Statistics Clustering, etc.

Modeling
- Machine Learning (Training) Parameter Estimation Simulation

Validation
- Hypothesis testing Model errors

Decision Making
- Forecasting Decision Trees, etc.
Processing Modes

### Batch Processing

\[ R = F(D_1, \ldots, D_k) \]

### Distributed Processing

\[ R_{i+1} = F(S_{i+1}) \]

\[ S_{i+1} = T(S_i, D_i) \]

### Online Processing

\[ R_1 \]

\[ R_2 \]

\[ R_k \]

\[ R = F(R_1, \ldots, R_k) \]

\[ R_{i+1} = F(S_{i+1}) \]
Use directly for
• Scaling to multiple nodes
• Streaming data
• Non-homogeneous dataframes

Scikit-Learn Equivalents
- PCA
- KMeans
- LinearRegression
- Ridge
- SVC
- pairwise_distances
- logistic_regression_path

Scikit-Learn API Compatible
- KNeighborsClassifier
- RandomForestClassifier
- RandomForestRegressor

daal4py

Intel® DAAL
Accelerating scikit-learn through daal4py

> python -m daal4py <your-scikit-learn-script>

import daal4py.sklearn
daal4py.sklearn.patch_sklearn()

Monkey-patch any scikit-learn on the command-line

Monkey-patch any scikit-learn programmatically

Scikit-learn with daal4py patches applied passes scikit-learn test-suite
Get a fly with daal4py

Fast & Scalable
- Close to native performance through Intel® DAAL
- Efficient MPI scale-out
- Streaming

Easy to use
- Intuitive usage model
- Picklable

Flexible
- Plugs into scikit-learn
- Plugs into HPAT/Numba

Open
- Open source: https://github.com/IntelPython/daal4py
Scaling Machine Learning Beyond a Single Node

- Simple Python API
  Powers scikit-learn

- Powered by DAAL

- Scalable to multiple nodes

Try it out! conda install -c intel daal4py

- scikit-learn
- daal4py

- Intel® Data Analytics Acceleration Library (DAAL)
- Intel® Math Kernel Library (MKL)
- Intel® Threading Building Blocks (TBB)

Intel® Data Analytics Acceleration Library (DAAL)
Intel® Math Kernel Library (MKL)
Intel® Threading Building Blocks (TBB)
Intel* Distribution for Python* Scikit-learn Optimizations, cont.

Intel optimizations improve scikit-learn efficiency closer to native code speeds on Intel® Xeon™ processors

Performance results are based on testing as of July 9, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

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 purchases, including the performance of that product when combined with other products. For more complete information, see Performance Benchmark Test Disclosure.

Testing by Intel as of July 9, 2018. Configuration: Stock Python; python 3.6.6 hv36621a 0 installed fromconda, numpy 1.15, numba 0.39.0, bernlite 0.24.0, scipy 1.1.2, scikit-learn 0.19.2 installed from pip; Intel Python; Intel® Distribution for Python* 2019 Gold: python 3.6.5 intel_11, numpy 1.14.3, intel, py36.5, mkl 2018.0 intel, 101, mkl, fit 1.2 intel_, np14py36.0, nlad 1.2.1 intel, np14py36.0, mkl random 1.2.1 intel, np14py36.0, numba 0.390 intel, np14py36.0, bernlite 0.24.0 intel, py36.5, scipy 1.1.0, intel, np14py36.0, scikit-learn 0.19.1 intel, np14py36.0; OS: CentOS Linux 7.3.1611, Kernel 3.10.0-514.el7.x86_64; Hardware: Intel® Xeon® Gold 6140 CPU @ 2.30GHz (2 sockets, 18 cores/socket, HT off), 256 GB of DDR4-RAM, 16 DIMMs of 16 GB at 2666 MHz

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20170604.

For more complete information about compiler optimizations, see our Optimization Notice.
Strong & Weak Scaling via daal4py

On a 32-node cluster (1280 cores) daal4py computed linear regression of 2.15 TB of data in 1.18 seconds and 68.66 GB of data in less than 48 milliseconds.

On a 32-node cluster (1280 cores) daal4py computed K-Means (10 clusters) of 1.12 TB of data in 107.4 seconds and 35.76 GB of data in 4.8 seconds.
import daal4py as d4p

# daal4py accepts data as CSV files, numpy arrays or pandas dataframes
# here we let daal4py load process-local data from csv files
data = "kmeans_dense.csv"

# Create algob object to compute initial centers
init = d4p.kmeans_init(10, method="plusPlusDense")
# compute initial centers
ires = init.compute(data)
# results can have multiple attributes, we need centroids
centroids = ires.centroids
# compute initial centroids & kmeans clustering
result = d4p.kmeans(10).compute(data, centroids)
Distributed K-Means using daal4py

```python
import daal4py as d4p

# initialize distributed execution environment
d4p.daalinit()

# daal4py accepts data as CSV files, numpy arrays or pandas dataframes
# here we let daal4py load process-local data from csv files
data = "kmeans_dense_{}.csv".format(d4p.my_procid())

# compute initial centroids & kmeans clustering
init = d4p.kmeans_init(10, method="plusPlusDense", distributed=True)
centroids = init.compute(data).centroids
result = d4p.kmeans(10, distributed=True).compute(data, centroids)

mpirun -n 4 python ./kmeans.py
```
~400 LOC total
import daal4py as d4p

# Configure a Linear regression training object for streaming
train_algo = d4p.linear_regression_training(interceptFlag=True, streaming=True)

# assume we have a generator returning blocks of (X,y)...
rn = read_next(infile)

# on which we iterate
for chunk in rn:
    algo.compute(chunk.X, chunk.y)

# finalize computation
result = algo.finalize()
Intel® DAAL Algorithms supported by daal4py
Data Transformation and Analysis

Basic statistics for datasets
- Low order moments
- Quantiles
- Order statistics

Correlation and dependence
- Cosine distance
- Correlation distance

Matrix factorizations
- SVD
- QR
- Cholesky

Dimensionality reduction
- PCA

Outlier detection
- Univariate
- Multivariate

Math functions (exp, log,…)

Optimization solvers (SGD, AdaGrad, LBFGS)

Algorithms supporting batch processing
Algorithms supporting batch, online and/or distributed processing
Intel® DAAL Algorithms supported by daal4py

Machine Learning

- **Regression**
  - Linear Regression
  - Ridge Regression
  - Decision Tree
  - Decision Forest
  - GradientBoosting

- **Classification**
  - Weak learner*
  - Boosting* (Ada, Brown, Logit)
  - Naïve Bayes
  - kNN
  - Support Vector Machine

- **Unsupervised learning**
  - K-Means Clustering
  - EM for GMM
  - Collaborative filtering
  - Alternating Least Squares

*Expected with DAAL® 2020

Algorithms supporting batch processing

Algorithms supporting batch, online and/or distributed processing
Scalable Python Solutions in Incubation

**daal4py**

*Ease-of-use of scikit-learn + Performance of DAAL*

- High-level Python API for DAAL
- 10x fewer LOC wrt DAAL for single node, 100x fewer LOC wrt DAAL for multi-node

**HPAT**

*Drop-in acceleration of Python ETL (Pandas, Numpy & select custom Python)*

- Statically compiles analytics code to binary
- Simply annotate with @hpat.jit
- Built on Anaconda Numba compiler

Automatically scales to multiple nodes with MPI

https://github.com/IntelLabs/hpat
Accelerating Pandas using HPAT

```python
import pandas as pd
import hpat

@hpat.jit
def process_times():
    df = pq.read_table('data.parquet').to_pandas();
    df['event_time'] = pd.DatetimeIndex(df['event_time'])
    df['hr'] = df.event_time.map(lambda x: x.hour)
    df['minute'] = df.event_time.map(lambda x: x.minute)
    df['second'] = df.event_time.map(lambda x: x.second)
    df['minute_day'] = df.apply(lambda row: row.hr*60 + row.minute, axis = 1)
    df['event_date'] = df.event_time.map(lambda x: x.date())
    df['indicator_cleaned'] = df.indicator.map(lambda x: -1 if x == 'na' else int(x))

$ mpirun -n 4 python ./process_times.py
```
INTEL-OPTIMIZED DL FRAMEWORKS
Deep Learning Usages & Key Topologies

**Image Recognition**
- Resnet-50, 101
- Inception V3
- MobileNet
- SqueezeNet

**Object Detection**
- R-FCN
- Faster-RCNN
- Yolo V2
- SSD-VGG16, SSD-MobileNet

**Image Segmentation**
- Mask R-CNN
- 3D-Unet

**Language Translation**
- GNMT
- Transformer LT

**Text to Speech**
- Wavenet

**Recommendation System**
- Wide & Deep NCF

There are many deep learning usages and topologies for each
Deep learning and AI ecosystem includes edge and datacenter applications.
- Open source frameworks (Tensorflow*, MXNet*, PyTorch*, PaddlePaddle*)
- Intel deep learning products (Nauta, BigDL, OpenVINO™ toolkit)
- In-house user applications

Intel® MKL and Intel® MKL-DNN optimize deep learning and machine learning applications for Intel® processors:
- Through the collaboration with framework maintainers to upstream changes (Tensorflow*, MXNet*, PyTorch, PaddlePaddle*)
- Through Intel-optimized forks (Caffe*)
- By partnering to enable proprietary solutions

Intel® MKL-DNN is an open source performance library for deep learning applications (available at https://github.com/intel/mkl-dnn)
- Fast open source implementations for wide range of DNN functions
- Early access to new and experimental functionality
- Open for community contributions

Intel® MKL is a proprietary performance library for wide range of math and science applications
Distribution: Intel Registration Center, package repositories (apt, yum, conda, pip), Intel® Parallel Studio XE, Intel® System Studio
Examples of speedups on Intel® Xeon® Scalable Processors

**INTEL-OPTIMIZED TENSORFLOW PERFORMANCE AT A GLANCE**

**TRAINING THROUGHPUT**

14X

Intel-optimized TensorFlow ResNet50 training performance compared to default TensorFlow for CPU

**INFEERENCE THROUGHPUT**

3.2X

Intel-optimized TensorFlow Inceptionv3 inference throughput compared to Default TensorFlow for CPU

Inference and training throughput uses FP32 instructions

System configuration:
- CPU Thread(s)/Core: 2, Core(s)/Socket: 28
- Socket(s): 2 NUMA node(s), 2 CPU family: 6
- Model: 85, Model name: Intel® Xeon® Platinum 8180 CPU @ 2.30GHz Stepping: 4
- HyperThreading: ON, ON Memory: 37GB (12 x 3GB) 24 slots, 12 occupied 2000 MHz Disk: Intel R53WCD80 x 3 (800GB, 1.5TB, 6TB) BIOS

**PERFORMANCE GAINS REPORTED BY OTHERS**

Intel TensorFlow Scalability Results Presented by Google @ TF Summit March 30, ’18

"By making use of [Intel's] open source library [MKL-DNN], we were able to achieve a 3x performance benefit and great scaling efficiency on training. This is an example of how important it is to have strong collaborations with companies like Intel."

"Other names and brands may be claimed as the property of others."

Copyright © 2018, Intel Corporation. All rights reserved. Other names and brands may be claimed as the property of others.
### What’s Happening Under The Hood?

#### Intel® MKL-DNN Functionality

#### Features:
- Training (float32) and inference (float32, int8)
- CNNs (1D, 2D and 3D), RNNs (plain, LSTM, GRU)
- Optimized for Intel processors

#### Portability:
- Compilers: Intel® C++ Compiler/Clang/GCC/MSVC*
- OSes: Linux*, Windows*, Mac*
- Threading: OpenMP*, TBB

#### Frameworks that use Intel® MKL-DNN:
- IntelCaffe, TensorFlow*, MxNet*, PaddlePaddle*, Pytorch*, ...

<table>
<thead>
<tr>
<th>Feature</th>
<th>Intel® MKL-DNN v0.16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>Direct 3D, Depthwise separable convolution, Winogrand convolution, Deconvolution</td>
</tr>
<tr>
<td>Fully Connected Layer</td>
<td>Inner Product</td>
</tr>
<tr>
<td>Pooling</td>
<td>Maximum, Average (include/exclude padding)</td>
</tr>
<tr>
<td>Normalization</td>
<td>LRN across/within channel, Batch normalization</td>
</tr>
<tr>
<td>Eltwise (Loss/activation)</td>
<td>ReLU (bounded/soft), ELU, Tanh; Softmax, Logistic, linear; square, sqrt, abs</td>
</tr>
<tr>
<td>Data manipulation</td>
<td>Reorder, sum, concat, View</td>
</tr>
<tr>
<td>RNN cell</td>
<td>RNN cell, LSTM cell, GRU cell</td>
</tr>
<tr>
<td>Fused primitive</td>
<td>Conv+ReLU+sum, BatchNorm+ReLU</td>
</tr>
<tr>
<td>Data type</td>
<td>f32, s32, s16, s8, u8</td>
</tr>
</tbody>
</table>

*Other names and brands may be claimed as the property of others.

Copyright © 2018, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.

Copyright © 2016, Intel Corporation.
Graph-level optimizations
AI Framework Software Optimizations Fusion
AI Framework Software Optimizations
Layout Conversion

• Converting to/from optimized layout can be less expensive than operating on un-optimized layout.

• All Intel® MKL-DNN operators use highly-optimized layouts for TensorFlow* tensors.
AI Framework Software Optimizations
Layout Propogation

Did you notice anything wrong with previous graph?

Problem: redundant conversions
Graph-level optimizations (contd)

- Batch Normalization Folding
- Filter Caching
- Primitive Reuse
TensorFlow* graphs offer opportunities for parallel execution.

Threading model, Tune you Intel® MKL w/

1. `inter_op_parallelism_threads` = max number of operators that can be executed in parallel

2. `intra_op_parallelism_threads` = max number of threads to use for executing an operator

3. `OMP_NUM_THREADS` = MKL-DNN equivalent of `intra_op_parallelism_threads`

More details: https://www.tensorflow.org/performance/performance_guide

```
>>> config = tf.ConfigProto()
>>> config.intra_op_parallelism_threads = 56
>>> config.inter_op_parallelism_threads = 2
>>> tf.Session(config=config)
```

```
os.environ["KMP_BLOCKTIME"] = "1"
os.environ["KMP AFFINITY"] = "granularity=fine,compact,1,0"
os.environ["KMP_SETTINGS"] = "0"
os.environ["OMP_NUM_THREADS"] = "56"
```
PROFILING
Intel MKL-DNN verbose mode overview

Simple yet powerful analysis tool:

- Similar to Intel MKL verbose
- Enabled via environment variable or function call
- Output is in CSV format

Output includes:

- The marker, state and primitive kind
- Implementation details (e.g. jit:avx2)
- Primitive parameters
- Creation or execution time (in ms)

Example below (details here)

```
$ # MKLDNN_VERBOSE is unset
$ ./.examples/simple-net-c passed

$ export MKLDNN_VERBOSE=1 # report only execution parameters and runtime
$ ./.examples/simple-net-c # | grep "mkldnn_verbose"
mkldnn_verbose,exec,reorder,jit:uni,undef,in:f32_oihw out:f32_Ohwi8o,num:1,96x3x11x11,12.2249
mkldnn_verbose,exec,eltwise,jit:avx2,forward_training,fdata:nChw8c,alg:eltwise_relu,mb8ic96ih55iw55,0.437988
mkldnn_verbose,exec,lrn,jit:avx2,forward_training,fdata:nChw8c,alg:lrn_across_channels,mb8ic96ih55iw55,1.70093
mkldnn_verbose,exec,reorder,jit:uni,undef,in:f32_nChw8c out:f32_nchw,num:1,8x96x27x27,0.924805
passed
```
Integration with Intel VTune Amplifier

Full application analysis

Report types:
- CPU utilization
- Parallelization efficiency
- Memory traffic

Profiling of run-time generated code must be enabled at compile time

```bash
# building Intel MKL-DNN using cmake
cmake -DVTUNEROOT=/opt/intel/vtune_amplifier_2018 .. && make install
```

```bash
# an alternative: building Intel MKL-DNN using sources directly, e.g. in TensorFlow
CFLAGS="-I$VTUNEROOT/include -DJIT_PROFILING_VTUNE" LDFLAGS="-L$VTUNEROOT/lib64 -ljitprofiling" bazel build
```
Intel-Optimized Frameworks: How To Get?

Check out our intel.ai for the framework optimizations page

https://www.intel.ai/framework-optimizations
### Intel® Optimization of Tensorflow*: How To Get?


<table>
<thead>
<tr>
<th>Method</th>
<th>Install Command</th>
</tr>
</thead>
</table>
| **Conda Package** | Main trunk → conda install tensorflow –c anaconda  
                          Intel Channel → conda install tensorflow –c intel  
                          IDP Full → conda create –n IDP intelpython3_full |
| **PyPI Package** | pip install intel-tensorflow              |
| **Docker Images** | $ docker pull docker.io/intelaipg/intel-optimized-tensorflow:latest  
                      $ docker run -it -p 8888:8888 intelaipg/intel-optimized-tensorflow |
| **Build from Source** | [https://github.com/tensorflow/tensorflow](https://github.com/tensorflow/tensorflow)  
                          Refer install guide for more details |
Intel® Optimization of MxNet* and Pytorch*: How To Get?

**PyPI Package**
- `pip install mxnet-mkl==1.2.0 [-user]`

**Build from Source**
- [https://github.com/apache/incubator-mxnet](https://github.com/apache/incubator-mxnet)
- Refer install guide for more details

---

Intel® Math Kernel Library for Deep Neural Network (MKL-DNN) has been integrated into official release of PyTorch by default, thus users can get performance benefit on Intel platform without additional installation steps.

[https://pytorch.org/](https://pytorch.org/)
SCALE AI WORKLOADS WITH INTEL OPTIMIZATIONS
SCALE AI WORKLOADS: BIGDL

Designed for Spark* or Apache* Hadoop* clusters running on Intel® Xeon® processors!

Rich deep learning support
Extremely high performance.

High Performance Deep Learning for FREE on CPU Infrastructure

https://software.intel.com/en-us/articles/bigdl-distributed-deep-learning-on-apache-spark

software.intel.com/bigdl

Optimization Notice

Feature Parity with Caffe* and Torch*
Lower TCO and improved ease of use with existing infrastructure
Deep Learning on Big Data Platform, Enabling Efficient Scale-Out

1Open-source software is available for download at no cost; ‘free’ is also contingent upon running on existing idle CPU infrastructure where the operating cost is treated as a ‘sunk’ cost.
Distributed Tensorflow with Parameter Server

The parameter server model for distributed training jobs can be configured with different ratios of parameter servers to workers, each with different performance profiles.

Available on AWS (DL AMI)

Intel optimized AI frameworks support Horovod seamlessly

Uber's open source Distributed training framework for TensorFlow

The ring all-reduce algorithm allows worker nodes to average gradients and disperse them to all nodes without the need for a parameter server.

Source: https://eng.uber.com/horovod/
SCALE AI: INTEL® MACHINE LEARNING SCALING LIBRARY (MLSL)

Scale Out Deep Learning: Requirements

- Choosing optimal work partitioning strategy
- Enabling scalability for small/large batch size
- Reducing communication volume
- Choosing optimal communication algorithm
- Prioritizing latency-bound communication
- Portable / efficient implementation
- Workload coverage across CNNs, RNNs, LSTMs, ...
- Integration with Deep Learning Frameworks

Data Parallelism

Input data  Weights or model  Output or activations

Model Parallelism

Input data  Weights or model  Partial outputs or activations  Output or activations

Hybrid Parallelism

Node Group 2  Weight transfer  Node Group 1  Activation transfer

Numerous DL Frameworks

Multiple NW Fabrics

Ethernet OmniPath® Infiniband®

Optimization Notice

SCALE AI: Intel® machine learning scaling library (MLSL)

Communication dependent on work partitioning strategy
Data parallelism = Allreduce (or) Reduce_Scatter + Allgather
Model parallelism = AlltoAll

https://github.com/intel/MLSL
Begin your AI journey efficiently, now with even more agility...

- IMT – Intel® Infrastructure Management Technologies
- ADQ – Application Device Queues
- SST – Intel® Speed Select Technology

Built-in Acceleration with Intel® Deep Learning Boost...

- Up to 30X deep learning throughput!¹

Hardware-Enhanced Security...

- Intel® Security Essentials
- Intel® SecL: Intel® Security Libraries for Data Center
- TDT – Intel® Threat Detection Technology

¹ Based on Intel internal testing: 1X, 5.7X, 14X and 30X performance improvement based on Intel® Optimization for Cafe ResNet-50 inference throughput performance on Intel® Xeon® Scalable Processor. See Configuration Details.

Performance results are based on testing as of 7/11/2017 (1X), 11/8/2018 (5.7X), 2/20/2019 (14X) and 2/26/2019 (30X) and may not reflect all publicly available security updates. No product can be absolutely secure. See configuration disclosure for details.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations may not be provided for microprocessors that are not manufactured by Intel. microarchitecture-based optimizations and other optimizations vary by product and may require Intel-specific instruction sets and other hardware features.
**INTEL® DEEP LEARNING Boost (DL Boost)**

featuring Vector Neural Network Instructions (VNNI)

### INT8

<table>
<thead>
<tr>
<th>Sign</th>
<th>Mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>07</td>
<td>06 05 04 03 02 01 00</td>
</tr>
</tbody>
</table>

Current AVX-512 instructions to perform INT8 convolutions:
- vpaddusbw, vpmaddwd, vpadd

**Future** AVX-512 (VNNI) instruction to accelerate INT8 convolutions:
- vpdpbusd

1. Fused multiply-add instruction
2. MKLDNN is optimized for VNNI

Speeds-up image classification, speech recognition, language translation, object detection and more
DEEP LEARNING PERFORMANCE ON CPU

Hardware + Software Improvements for Intel® Xeon® Processors

Baseline vs. Baseline

5.7x

285x

Baseline

2S Intel® Xeon® Scalable Processor (Skylake)

July 2017

July 2017 Skylake launch

February 2019

Caffe

Resnet-50 1.9x

Inception v3 1.8x

SSD-VGG16 2.0x

TensorFlow

Resnet-50 1.9x

Inception v3 1.8x

1 5.7x inference throughput improvement with Intel® Optimizations for Caffe ResNet-50 on Intel® Xeon® Platinum 8180 Processor in Feb 2019 compared to performance at launch in July 2017. See configuration details on Config 1.

Performance results are based on testing as of dates shown in configuration and may not reflect all publicly available security updates.

2 8/24/2018) Results have been estimated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Product can be absolutely secure. See configuration disclosure for details.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction set extensions and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. 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 purchases, including the performance of that product when combined with other products.

https://www.intel.com/performance
VNNI usage verification by dumping JIT kernel

- Please refer to: https://intel.github.io/mkl-dnn/perf_profile.html

  - $export MKLDNN_JIT_DUMP=1
  - $./build/simple_net_int8_cpp
  - $ objdump -D -b binary -mi386:x86-64 mkldnn_dump__jit_avx512_core_x8s8s32x_conv_fwd_ker_t.2.bin | grep vpdpbusd
  - The sample output with VNNI instruction sets
Training Performance on AWS C5 instance

Data collected from AWS Machine learning blog

From the blog post

“Training a ResNet-50 benchmark with synthetic ImageNet dataset using our optimized build of TensorFlow 1.6 on a c5.18xlarge instance type was 7.4X faster than training on the stock TensorFlow 1.6 binaries”

- Sumit Thakur, AWS

System configuration and hardware specs at:


Now intel-optimized TensorFlow 1.13 is available on AWS DLAMI and 9.5x faster

TF SCALING EFFICIENCY W/ HOROVOD

Up to 94% of scaling w/ 2 MPI process

InceptionResNet-v2 was able to maintain at least 80 percent scalability for up to 32 nodes w/ 1 MPI Process/node

Up to 89 percent (ResNet-50*) of scaling efficiency w/ 1 MPI Process

INFERNECE THROUGHPUT PERFORMANCE
CPU OPTIMIZED TENSORFLOW COMPARED WITH UNOPTIMIZED (STOCK) TENSORFLOW

For More Details: https://www.anaconda.com/tensorflow-cpu-optimizations-in-anaconda/
INTEL AI OOB TOOLS FOR CASCADE LAKE

INTEL MODEL ZOO
1. Demonstrate the AI workloads and deep learning models (fp32 and int8) Intel has optimized and validated to run on Intel hardware
2. Deliver E2E inference solution at scale on selective use cases
   (Data, AI model) → model zoo inference script → inference results
3. Make it easy to benchmark model performance on Intel hardware both in the cloud or baremetal

https://github.com/IntelAI/models

INTEL QUANTIZATION TOOLS
(int8 inference)
1. Post-training model optimization+ quantization process on Intel® Xeon processors
2. Reduced model size resulting in FASTER inference utilizing VNNI instructions, while maintaining accuracy

LOW PRECISION INFERENCE
FP32 model
MKL DNN Primitive
Quantized model
INT8 Model

https://github.com/IntelAI/tools
https://github.com/intel/Detectron
## PRETRAINED MODELS PUBLISHED ON TOP TOPOLOGIES

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Framework</th>
<th>Model</th>
<th>Mode</th>
<th>Bench Instr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adversarial Networks</td>
<td>TensorFlow</td>
<td>DCGAN</td>
<td>Inference</td>
<td>FP32</td>
</tr>
<tr>
<td>Content Creation</td>
<td>TensorFlow</td>
<td>DRAW</td>
<td>Inference</td>
<td>FP32</td>
</tr>
<tr>
<td>Face Detection and Alignment</td>
<td>TensorFlow</td>
<td>FaceNet</td>
<td>Inference</td>
<td>FP32</td>
</tr>
<tr>
<td>Face Detection and Alignment</td>
<td>TensorFlow</td>
<td>MTCC</td>
<td>Inference</td>
<td>FP32</td>
</tr>
<tr>
<td>Image Recognition</td>
<td>TensorFlow</td>
<td>Inception ResNet V2</td>
<td>Inference</td>
<td>Int8 FP32</td>
</tr>
<tr>
<td>Image Recognition</td>
<td>TensorFlow</td>
<td>Inception V3</td>
<td>Inference</td>
<td>Int8 FP32</td>
</tr>
<tr>
<td>Image Recognition</td>
<td>TensorFlow</td>
<td>Inception V4</td>
<td>Inference</td>
<td>Int8 FP32</td>
</tr>
<tr>
<td>Image Recognition</td>
<td>TensorFlow</td>
<td>MobileNet V1</td>
<td>Inference</td>
<td>FP32</td>
</tr>
<tr>
<td>Image Recognition</td>
<td>TensorFlow</td>
<td>ResNet 101</td>
<td>Inference</td>
<td>Int8 FP32</td>
</tr>
<tr>
<td>Image Recognition</td>
<td>TensorFlow</td>
<td>ResNet 50</td>
<td>Inference</td>
<td>Int8 FP32</td>
</tr>
<tr>
<td>Image Recognition</td>
<td>TensorFlow</td>
<td>SqueezeNet</td>
<td>Inference</td>
<td>FP32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Framework</th>
<th>Model</th>
<th>Mode</th>
<th>Bench Instr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Detection</td>
<td>TensorFlow</td>
<td>R-FCN</td>
<td>Inference</td>
<td>FP32</td>
</tr>
<tr>
<td>Object Detection</td>
<td>TensorFlow</td>
<td>Faster R-CNN</td>
<td>Inference</td>
<td>Int8 FP32</td>
</tr>
<tr>
<td>Object Detection</td>
<td>TensorFlow</td>
<td>SSD-MobileNet</td>
<td>Inference</td>
<td>Int8 FP32</td>
</tr>
<tr>
<td>Recommendation</td>
<td>TensorFlow</td>
<td>NCF</td>
<td>Inference</td>
<td>FP32</td>
</tr>
<tr>
<td>Recommendation</td>
<td>TensorFlow</td>
<td>Wide &amp; Deep Large Dataset</td>
<td>Int8 FP32</td>
<td></td>
</tr>
<tr>
<td>Text-to-Speech</td>
<td>TensorFlow</td>
<td>WaveNet</td>
<td>Inference</td>
<td>FP32</td>
</tr>
</tbody>
</table>
Article Plug
Intel–Optimized TensorFlow* Performance Considerations

Maximize TensorFlow* Performance on CPU: Considerations and Recommendations for Inference Workloads

By Nathan Greenelch (Intel), Jing X. (Intel), published on January 25, 2019

To fully utilize the power of Intel® architecture (IA) and thus yield high performance, TensorFlow* can be powered by Intel’s highly optimized math routines for deep learning tasks. This primitives library is called Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN). Intel MKL-DNN includes convolution, normalization, activation, inner product, and other primitives.

TensorFlow Runtime Options Affecting Performance


- intra_op_parallelism_threads
- Data layout

Free Support: Intel® AI Frameworks Forum

https://forums.intel.com

INTEL® OPTIMIZED AI FRAMEWORKS

Support for key Deep Learning Frameworks and Libraries optimized for Intel Hardware.

Intel TensorFlow Installation and Performance

Intel® Optimized AI Frameworks · NathanG_intel · February 6, 2019 at 2:01 PM

155 likes · 1 comment · 0 shares
Call to Action

More information at
www.intel.ai/framework-optimizations/

Use Intel’s performance-optimized libraries & frameworks

Use Our Free Support: forums.intel.com
Choose “Intel Optimized AI Frameworks” from list