INTEL® DISTRIBUTION FOR PYTHON*
INTEL® DISTRIBUTION FOR PYTHON* 2017
Advancing Python performance closer to native speeds

Easy, out-of-the-box access to high performance Python
- Prebuilt, optimized for numerical computing, data analytics, HPC
- Drop in replacement for your existing Python. No code changes required

High performance with multiple optimization techniques
- Accelerated NumPy*/SciPy*/Scikit-Learn* with Intel® MKL
- Data analytics with pyDAAL, enhanced thread scheduling with TBB, Jupyter* Notebook interface, Numba*, Cython*
- Scale easily with optimized MPI4Py and Jupyter notebooks

Faster access to latest optimizations for Intel architecture
- Distribution and individual optimized packages available through conda and Anaconda Cloud: anaconda.org/intel
- Optimizations upstreamed back to main Python trunk

© 2017 Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.
For more complete information about compiler optimizations, see our Optimization Notice.
Installing Intel® Distribution for Python* 2017

Stand-alone installer and anaconda.org/intel

Download full installer from https://software.intel.com/en-us/intel-distribution-for-python

OR

> conda config --add channels intel
> conda install intelpython3_full
> conda install intelpython3_core

docker pull intelpython/intelpython3_full
2017 UPDATE 2 CHANGES

- Scikit-learn* accelerated with Intel® Data Analytics Acceleration Library (DAAL) for faster machine learning
- Increased optimizations on Fast Fourier Transforms (FFT) for NumPy* and SciPy* FFTs
- Changes in NumPy* to arithmetic and transcendental functions via umath optimizations and vectorization (AVX2, AVX-512 with MKL), can utilize multiple cores, memory management
- pyDAAL extensions for neural networks, advanced tensor inputs, distributed computing primitives
MEMORY OPTIMIZATIONS FOR NUMPY* ARRAYS

- Optimized array allocation/reallocation, copy/move
  - Memory alignment and data copy vectorization & threading
SCIKIT-LEARN* OPTIMIZATIONS WITH INTEL® MKL
Speedups of Scikit-Learn* Benchmarks (2017 Update 1)

Intel® Distribution for Python* 2017 Update 1 vs. system Python & NumPy*/Scikit-Learn*

System info: 32x Intel® Xeon® CPU E5-2698 v3 @ 2.30GHz; disabled HT, 64GB RAM; Intel® Distribution for Python* 2017 Gold; Intel® MKL 2017.0.0; Ubuntu 14.04.4 LTS; NumPy 1.11.1; scikit-learn 0.17.1. See Optimization Notice.
MORE SCIKIT-LEARN* OPTIMIZATIONS WITH INTEL® DAAL

Speedups of Scikit-Learn* Benchmarks (2017 Update 2)

- Accelerated key Machine Learning algorithms with Intel® DAAL
  - Distances, K-means, Linear & Ridge Regression, PCA
  - Up to 160x speedup on top of MKL initial optimizations

### Scikit-Learn Optimizations

Due to Intel(R) DAAL

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Kx150K Correlation Distance</td>
<td>158.91</td>
</tr>
<tr>
<td>1Kx150K Cosine Distance</td>
<td>2.56</td>
</tr>
<tr>
<td>100Kx50, 10 clusters K-means</td>
<td>157.94</td>
</tr>
<tr>
<td>10Mx25, training Linear Regression</td>
<td>39.65</td>
</tr>
<tr>
<td>10Mx25, training Ridge Regression</td>
<td>5.39</td>
</tr>
<tr>
<td>1Mx50, 3 components PCA</td>
<td>1.57</td>
</tr>
</tbody>
</table>

**Intel Python 2017 U2 vs. U1**
FFT Accelerations with Intel® Distribution for Python®

FFT Accelerations on i5 processors (2017 Update 2)

Python* FFT Performance as a Percentage of C/Intel® Math Kernel Library (Intel® MKL) for Intel® Core™ i5 Processor (Higher is Better)

- **pip/numpy**
- **Intel Python**

<table>
<thead>
<tr>
<th></th>
<th>1 core</th>
<th>2 cores</th>
<th>1 core</th>
<th>2 cores</th>
<th>1 core</th>
<th>2 cores</th>
<th>1 core</th>
<th>2 cores</th>
<th>1 core</th>
<th>2 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D FFT in-place</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>2D FFT</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
</tr>
<tr>
<td>3D FFT in-place</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
</tr>
<tr>
<td>2D FFT out-of-place</td>
<td>70%</td>
<td>70%</td>
<td>70%</td>
<td>70%</td>
<td>70%</td>
<td>70%</td>
<td>70%</td>
<td>70%</td>
<td>70%</td>
<td>70%</td>
</tr>
<tr>
<td>3D FFT</td>
<td>60%</td>
<td>60%</td>
<td>60%</td>
<td>60%</td>
<td>60%</td>
<td>60%</td>
<td>60%</td>
<td>60%</td>
<td>60%</td>
<td>60%</td>
</tr>
</tbody>
</table>

*Other names and brands may be claimed as the property of others. For more complete information about compiler optimizations, see our [Optimization Notice](#).
FFT ACCELERATIONS WITH INTEL® DISTRIBUTION FOR PYTHON*

FFT Accelerations on Xeon processors (2017 Update 2)

Python* FFT Performance as a Percentage of C/Intel® Math Kernel Library (Intel® MKL) for Intel® Xeon™ Processor Family (Higher is Better)

- pip/numpy
- Intel Python

Percentage of native C

<table>
<thead>
<tr>
<th></th>
<th>1 core</th>
<th>32 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D FFT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2D FFT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3D FFT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1D FFT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2D FFT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3D FFT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Other names and brands may be claimed as the property of others.
For more complete information about compiler optimizations, see our Optimization Notice.
Python* FFT Performance as a Percentage of C/Intel® Math Kernel Library (Intel® MKL)
for Intel® Xeon Phi™ Product Family (Higher is Better)

- pip/numpy
- Intel Python

Percentage of native C

<table>
<thead>
<tr>
<th>1D FFT</th>
<th>2D FFT in-place</th>
<th>3D FFT</th>
<th>1D FFT</th>
<th>2D FFT out-of-place</th>
<th>3D FFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 core</td>
<td>256 cores</td>
<td>1 core</td>
<td>256 cores</td>
<td>1 core</td>
<td>256 cores</td>
</tr>
</tbody>
</table>
BLACK SCHOLES* BENCHMARKS
Black Scholes algorithm on i5 processors (2017 Update 2)

Performance Speedups for Intel® Distribution for Python* for Black Scholes* Formula on Intel® Core™ i5 Processor (Higher is Better)

© 2017 Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others. For more complete information about compiler optimizations, see our Optimization Notice.
BLACK SCHOLES* BENCHMARKS

Black Scholes algorithm on Xeon processors (2017 Update 2)

Performance Speedups for Intel® Distribution for Python* for Black Scholes* Formula on Intel® Xeon™ Processors ((Higher is Better)

- Pip/numpy
- Intel Python

Size: Number of options

Speedup

0 20 40 60 80 100 120 140 160 180

1024 2048 4096 8192 16384 32768 65536 131072 262144 524288 1048576 2097152 4194304 8388608 16777216

© 2017 Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.
For more complete information about compiler optimizations, see our Optimization Notice.
Performance Speedups for Intel® Distribution for Python® for Black Scholes® Formula on Intel® Xeon Phi™ Product Family (Higher is Better)

- pip/numpy
- Intel Python

Size: Number of options

- 1024
- 2048
- 4096
- 8192
- 16384
- 32768
- 65536
- 131072
- 262144
- 524288
- 1048576
- 2097152
- 4194304
- 8388608
- 16777216

Speedup

Xeon Phi
Scikit-learn* Benchmarks

Python* Performance as a Percentage of C++ Intel® Data Analytics Acceleration Library (Intel® DAAL) on Intel® Core™ i5 Processors (Higher is Better)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Correlation Distance</th>
<th>Cosine Distance</th>
<th>Linear Regression (Training)</th>
<th>Ridge Regression (Training)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Size</td>
<td>1K x 50K</td>
<td>1K x 50K</td>
<td>1M x 25</td>
<td>1M x 25</td>
</tr>
<tr>
<td>Cores</td>
<td>1 core</td>
<td>2 cores</td>
<td>1 core</td>
<td>2 cores</td>
</tr>
<tr>
<td>Optimization</td>
<td>pip/scikit-learn</td>
<td>Intel Python</td>
<td>pip/scikit-learn</td>
<td>Intel Python</td>
</tr>
</tbody>
</table>

Percentage of native C

© 2017 Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.
For more complete information about compiler optimizations, see our Optimization Notice.
Python* Performance as a Percentage of C++ Intel® Data Analytics Acceleration Library (Intel® DAAL) on Intel® Xeon® Processors (Higher is Better)

- **Correlation Distance**: 1K x 200K
- **Cosine Distance**: 1K x 200K
- **Linear Regression (Training)**: 10M x 50
- **Ridge Regression (Training)**: 10M x 50

**Graph Details**:
- **X-axis**: Core Count (1 core, 32 cores)
- **Y-axis**: Percentage of native C
- **Data Series**: pip/scikit-learn, Intel Python

© 2017 Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.
For more complete information about compiler optimizations, see our Optimization Notice.
**SCIKIT-LEARN** BENCHMARKS

Python* Performance as a Percentage of C++ Intel® Data Analytics Acceleration Library (Intel® DAAL) for Intel® Xeon Phi™ Product Family (Higher is Better)

- Correlation Distance
- Cosine Distance
- Linear Regression (Training)
- Ridge Regression (Training)

<table>
<thead>
<tr>
<th>Test Case</th>
<th>1 core</th>
<th>64 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K x 200K</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Correlation</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Distance</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Cosine Distance</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Linear Regression (Training)</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Ridge Regression (Training)</td>
<td>100%</td>
<td>98%</td>
</tr>
</tbody>
</table>

*Other names and brands may be claimed as the property of others.
For more complete information about compiler optimizations, see our [Optimization Notice](#).
CONFIGURATION INFORMATION

Software

- Pip*/NumPy*: Installed with Pip, Ubuntu*, Python* 3.5.2, NumPy=1.12.1, scikit-learn*=0.18.1
- Windows*, Python 3.5.2, Pip/NumPy=1.12.1, scikit-learn=0.18.1
- Intel® Distribution for Python 2017, Update 2

Hardware

- Intel® Core™ i5-4300M processor @ 2.60 GHz  2.59 GHz, (1 socket, 2 cores, 2 threads per core), 8GB DRAM
- Intel® Xeon® E5-2698 v3 processor @ 2.30 GHz (2 sockets, 16 cores each, 1 thread per core), 64GB of DRAM
- Intel® Xeon Phi™ processor 7210 @ 1.30 GHz (1 socket, 64 cores, 4 threads per core), DRAM 32 GB, MCDRAM (Flat mode enabled) 16GB

Modifications

- Scikit-learn: conda installed NumPy with Intel® Math Kernel Library (Intel® MKL) on Windows (pip install scipy on Windows contains Intel® MKL dependency)
- Black Scholes* on Intel Core i5 processor/Windows: Pip installed NumPy and conda installed SciPy
Intel® Distribution for Python*

Intel® Performance Libraries
- Intel® IPP
- Intel® TBB
- Intel® MKL
- Intel® DAAL
- Intel® MPI Library

Intel®
DAAL

Intel®
IPP

Intel®
TBB

Intel®
MKL

Intel®
DAAL

Intel®
MPI Library

Numpy*
Scipy*
Scikit-learn*
pyDAAL
Pandas*
Mpi4py*

...
FASTER DATA ANALYTICS
AND MACHINE LEARNING WITH PYDAAL
INTEL PYTHON LANDSCAPE

Intel® Distribution for Python*

Intel® Performance Libraries
- Intel® IPP
- Intel® TBB
- Intel® MKL
- Intel® DAAL
- Intel® MPI Library

Other Libaries
- Numpy*
- Scipy*
- Scikit-learn*
- Pandas*
- Mpi4py*
- ...
INTEL® DAAL: HETEROGENEOUS ANALYTICS

- Targets both data centers (Intel® Xeon® and Intel® Xeon Phi™) and edge-devices (Intel® Atom™)
- Perform analysis close to data source (sensor/client/server) to optimize response latency, decrease network bandwidth utilization, and maximize security
- Offload data to server/cluster for complex and large-scale analytics

Available also in open source: https://software.intel.com/en-us/articles/opendaal

- (De-)Compression
- (De-)Serialization
- PCA
- Statistical moments
- Quantiles
- Variance matrix
- QR, SVD, Cholesky
- Apriori
- Outlier detection
- Regression
  - Linear
  - Ridge
- Classification
  - Naïve Bayes
  - SVM
  - Classifier boosting
  - kNN
- Clustering
  - Kmeans
  - EM GMM
- Collaborative filtering
  - ALS
- Neural Networks
Classification

Problems

- An emailing service provider wants to build a spam filter for the customers
- A postal service wants to implement handwritten address interpretation

Solution: Support Vector Machine (SVM)

- Works well for non-linear decision boundary
- Two kernel functions are provided:
  - Linear kernel
  - Gaussian kernel (RBF)
- Multi-class classifier
  - One-vs-One

Source: Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2014). An Introduction to Statistical Learning. Springer
**PYDAAL EXAMPLE – SUPPORT VECTOR MACHINE**

**Model training**

```python
from daal.algorithms.svm import training, prediction
import daal.algorithms.kernel_function.linear
import daal.algorithms.classifier.training

kernel = kernel_function.linear.Batch()

def trainModel():
    dataSource = FileDataSource('train_data.csv', ...)
    labelsSource = FileDataSource('train_labels.csv', ...)
    dataSource.loadDataBlock()
    labelsSource.loadDataBlock()

    algorithm = training.Batch()

    algorithm.parameter.kernel = kernel
    algorithm.parameter.cacheSize = 600000000

    algorithm.input.set(classifier.training.data, dataSource.getNumericTable())
    algorithm.input.set(classifier.training.labels, labelsSource.getNumericTable())

    return algorithm.compute()
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel function to use with SVM algorithm</td>
<td><code>kernel = kernel_function.linear.Batch()</code></td>
</tr>
<tr>
<td>Initialize data source to retrieve data from CSV files</td>
<td><code>dataSource = FileDataSource('train_data.csv', ...)</code></td>
</tr>
<tr>
<td>Retrieve the data from the input files</td>
<td><code>dataSource.loadDataBlock()</code></td>
</tr>
<tr>
<td>Create an algorithm object to train the SVM model</td>
<td><code>algorithm = training.Batch()</code></td>
</tr>
<tr>
<td>Set the parameters of the algorithm</td>
<td><code>algorithm.parameter.kernel = kernel</code></td>
</tr>
<tr>
<td><code>algorithm.parameter.cacheSize = 600000000</code></td>
<td></td>
</tr>
<tr>
<td>Pass the training data set and labels to the algorithm</td>
<td><code>algorithm.input.set(classifier.training.data, dataSource.getNumericTable())</code></td>
</tr>
<tr>
<td><code>algorithm.input.set(classifier.training.labels, labelsSource.getNumericTable())</code></td>
<td></td>
</tr>
<tr>
<td>Build the SVM model</td>
<td><code>return algorithm.compute()</code></td>
</tr>
</tbody>
</table>
```python
def testModel(trainingResult):
    dataSource = FileDataSource('test_data.csv', ...)
    dataSource.loadDataBlock()

    algorithm = prediction.Batch()

    algorithm.parameter.kernel = kernel

    svmModel = trainingResult.get(classifier.training.model)
    algorithm.input.setTable(classifier.prediction.data,
                              dataSource.getNumericTable())
    algorithm.input.setModel(classifier.prediction.model, svmModel)

    return algorithm.compute()
```

Initialize data source to retrieve data from CSV file and retrieve the data from the input file

Create an algorithm object to predict the results

Set the parameters of the algorithm

Pass a testing data set and the trained model to the algorithm

Predict the SVM results
PERFORMANCE EXAMPLE: READ AND COMPUTE
SVM Classification with RBF kernel

- Training dataset: CSV file (PCA-preprocessed MNIST, 40 principal components) \( n=42000, p=40 \)
- Testing dataset: CSV file (PCA-preprocessed MNIST, 40 principal components) \( n=28000, p=40 \)

System Info: Intel® Xeon® CPU E5-2680 v3 @ 2.50GHz, 504GB, 2x24 cores, HT=on, OS RH7.2 x86_64, Intel® Distribution for Python* 2017 Update 1 (Python* 3.5)
ANOMALY DETECTION PROBLEM EXAMPLE

- State of the art solution
  - The data collected by sensors sent to mainland for analysis and decision making
    - Excessive amount of data is transferred, communication channel is overloaded
SOLUTION: ONLINE PROCESSING

- Update the decision when the new portion of data is available
- The whole data set may not fit into memory but still can be processed on one machine

```plaintext
Solution: Online Processing

- Update the decision when the new portion of data is available
- The whole dataset may not fit into memory but still can be processed on one machine

Wait for data

More data?

Yes

Data Block(i)

alg.compute()

No

alg.finalize Compute()

Final result

Partial result(i+1) = F(Partial result(i), DataBlock(i))
```
PROJECTION METHODS FOR OUTLIER DETECTION

Principal Component Analysis (PCA)

- Computes principal components: the directions of the largest variance, the directions where the data is mostly spread out

PCA for outlier detection

- Project new observation on the space of the first k principal components
- Calculate score distance for the projection using first k singular values
- Compare the distance against threshold

http://i.stack.imgur.com/uYaTv.png
from daal.algorithms.pca import (  
    Online_Float64CorrelationDense, data,  
    eigenvalues, eigenvectors  
)

dataSource = FileDataSource(  
    dataFileName,  
    DataSourceIface.doAllocateNumericTable,  
    DataSourceIface.doDictionaryFromContext  
)

algorithm = Online_Float64CorrelationDense()  

while dataSource.loadDataBlock(nVectorsInBlock) > 0:
    algorithm.input.setDataset(data,  
    dataSource.getNumericTable())
    algorithm.compute()

result = algorithm.finalizeCompute()

printNumericTable(result.get(eigenvalues))
printNumericTable(result.get(eigenvectors))

Data sets that does not fit into memory could be processed effectively on a single machine
SOLUTION: DISTRIBUTED PROCESSING

Worker Process i

Data Block(i) → alg. compute() → Partial result(i)

Worker Process j

Data Block(j) → alg. compute() → Partial result(j)

Communication API

Master Process

Partial result(i) → alg. compute() → Partial result(j) → alg.finalize

Compute(Compute()) → Result
DISTRIBUTED COMPUTING LANDSCAPE

- mpi4py
- pySpark
- Dask/distributed

- New distributed computing technologies appear almost every year
- Intel® DAAL provides communication layer agnostic building blocks to create distributed algorithms
**DISTRIBUTED PROCESSING: STEP 1 ON WORKER**

```python
define Distributed Processing: Step 1 on Worker:

```from daal import step1Local, step2Master
import daal.algorithms.pca as pca
from daal.data_management import (OutputDataArchive, InputDataArchive, FileDataSource, DataSourceIface)

dataSource = FileDataSource(datasetFileNames[RankId], ...)
Initialize data source to retrieve data from CSV files

dataSource.loadDataBlock()
Retrieve the input data

algorithm = pca.Distributed(step1Local)
Create a PCA algorithm for using the correlation method on the local node

algorithm.input.setDataset(pca.data,
Set input data to the algorithm
dataSource.getNumericTable())

pres = algorithm.compute()
Compute partial results of the PCA algorithm

dataArch = InputDataArchive()

pres.serialize(dataArch)

nodeResults = dataArch.getArchiveAsArray()

serializedData = comm.gather(nodeResults)
Gather the partial results on master node
```
DISTRIBUTED PROCESSING: STEP 2 ON MASTER

if rankId == MPI_ROOT:
    masterAlgorithm = pca.Distributed(step2Master)
    Create a PCA algorithm for using the correlation method on the master node

    for i in range(nBlocks):
        dataArch =
            OutputDataArchive(serializedData[i])
        dataForStep2FromStep1 =
            pca.PartialResult(pca.correlationDense)
        dataForStep2FromStep1.deserialize(dataArch)
        De-serialize partial results that were gathered from the local nodes

        masterAlgorithm.input.add(  
            pca.partialResults, dataForStep2FromStep1)
        Set local partial results as inputs for the master algorithm

        masterAlgorithm.compute()  
        Compute partial results of the PCA algorithm

        res = masterAlgorithm.finalizeCompute()  
        Compute final results of the PCA algorithm

        printNumericTable(res.get(pca.eigenvalues))
        printNumericTable(res.get(pca.eigenvectors))  
        Print the results
REFERENCES


- Intel® Distribution for Python* Documentation
WHAT’S NEXT - TAKEAWAYS

- Learn more about Intel® DAAL
  - It supports C++ and Java*, too!
  - We want you to use Intel® DAAL in your data analytics projects

- Learn more about Intel® Distribution for Python*
  - Beyond machine learning, many more benefits

- Keep an eye on the tutorial repository
  - [https://github.com/daaltces/pydaal-tutorials](https://github.com/daaltces/pydaal-tutorials)
  - We'll be adding more labs, samples, etc.
LEGAL DISCLAIMER & OPTIMIZATION NOTICE

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

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 about compiler optimizations, see our Optimization Notice at https://software.intel.com/en-us/articles/optimization-notice#opt-en.

Copyright © 2017, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.