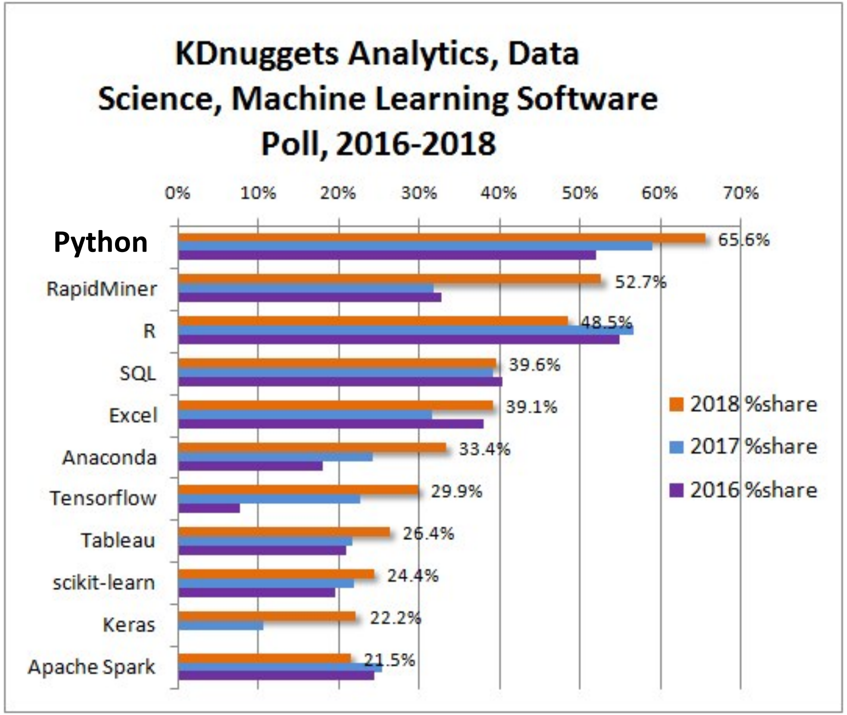
Software

INTEL DISTRIBUTION FOR PYTHON

Oleksandr Pavlyk

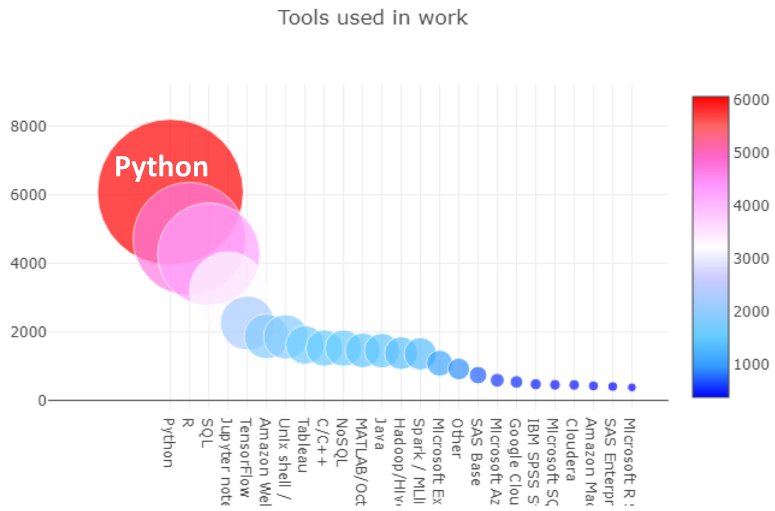
Intel® Distribution for Python* team
Machine Learning Engineer

Python - lingua franca of data science



https://www.kdnuggets.com/2018/05/poll-tools-analytics-data-science-machine-learning-results.html

Kaggle ML and Data Science Survey, 2017



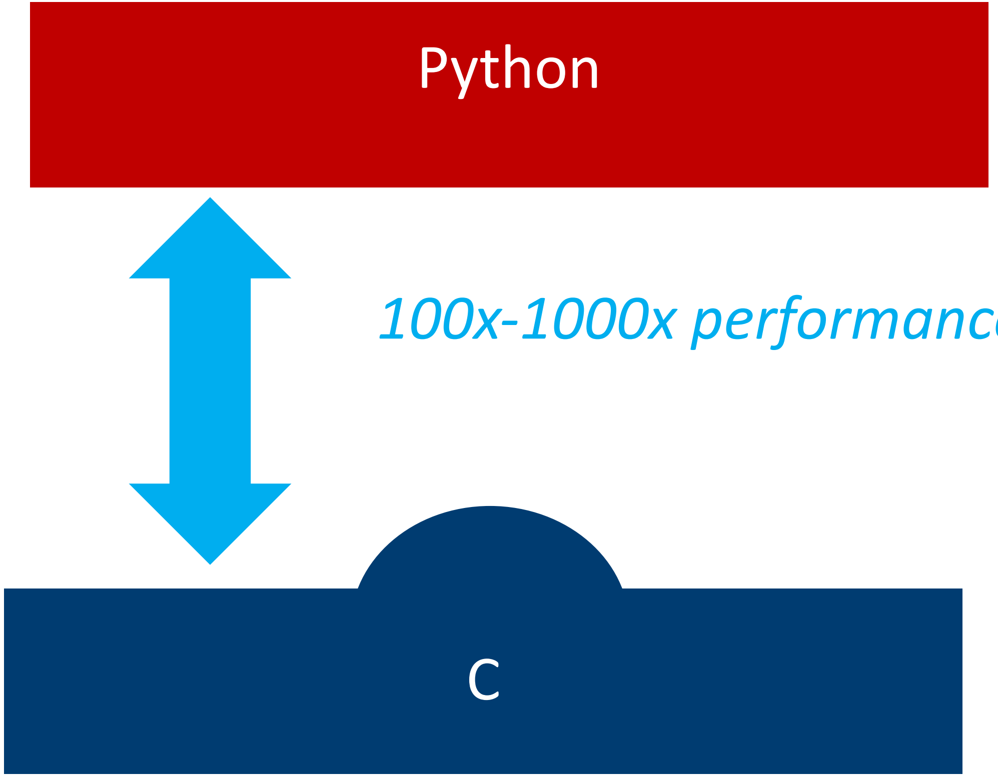
https://www.kaggle.com/sudalairajkumar/an-interactive-deep-dive-into-survey-results/data

The Reality of "Data Centric Computing"

Performance Limited	• Software is slow and single-node for many organizations • Only sample a small portion of data
Productivity Limited	• More performant/scalable implementations require significantly more development & deployment skills & time
Compute/ Bandwidth Limited	• Performance bottleneck often in compute or in memory bandwidth

Our mission: Deliver Python technologies that scale-up/out entire data analytics pipeline in productive way

Performance of Python



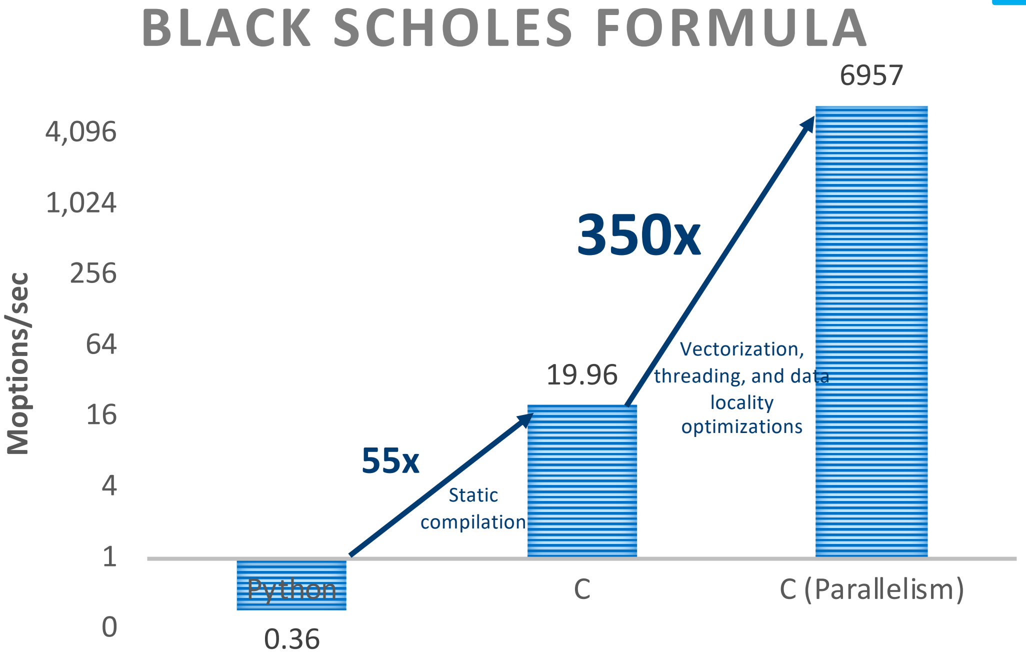
Interpreter
GIL

100x-1000x performance gap

Optimizing compiler
OpenMP*/TBB/pthreads

Parallelism Matters Most

Unlocking parallelism is essential to make Python useful in production



$$V_{call} = S_0 \cdot \mathrm{CDF}(d_1) - e^{-rT} \cdot X \cdot \mathrm{CDF}(d_2)$$

$$V_{put} = e^{-rT} \cdot X \cdot \mathrm{CDF}(-d_2) - S_0 \cdot \mathrm{CDF}(-d_1)$$

$$d_1 = \frac{\ln\left(S_0/X\right) + \left(r + \sigma^2/2\right)T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{\ln\left(S_0/X\right) + \left(r - \sigma^2/2\right)T}{\sigma\sqrt{T}}$$

Configuration info: - Versions: Intel® Distribution for Python 2.7.10 Technical Preview 1 (Aug 03, 2015), icc 15.0; Hardware: Intel® Xeon® CPU E5-2698 v3 @ 2.30GHz (2 sockets, 16 cores each, HT=OFF), 64 GB of RAM, 8 DIMMS of 8GB@2133MHz; Operating System: Ubuntu 14.04 LTS.

Performance of Python

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

LLVM-based compiler
Multiple threading runtimes

Small %% performance gap

C

Optimizing compiler
OpenMP*/TBB/pthreads

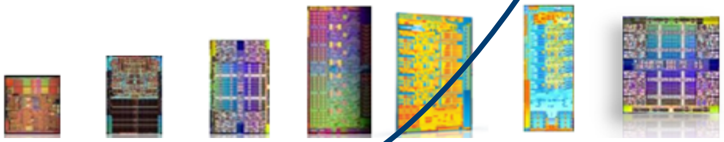
```python
@numba.jit(nopython=True, parallel=True)
def logistic_regression(Y, X, w0, step, iterations):
    """SGD solver for binary logistic regression."""
    w = w0.copy()
    for i in range(iterations):
        w += step * np.dot((1.0/(1.0 + np.exp(Y * np.dot(X, w)))) * Y, X)
    return w

```

https://www.anaconda.com/blog/developer-blog/parallel-python-with-numba-and-parallelaccelerator/

High Performance Python

Python Libraries

Intel® Performance Libraries



Thin layer in Python or Cython

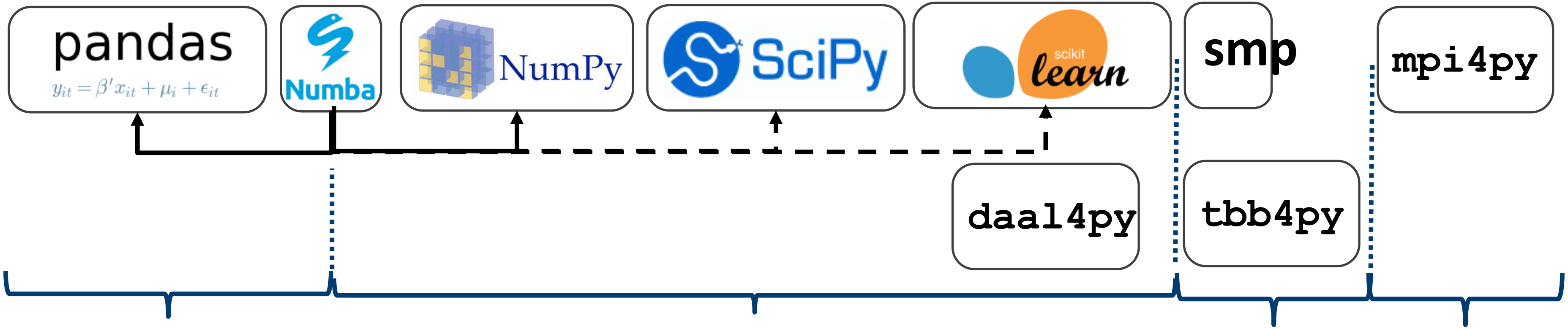
Native highly optimized libraries (Intel MKL, Intel DAAL, Intel IPP)

(generations of processors)

more nodes,
more cores,
more threads,
wider vectors, …

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 for Python*

https://software.intel.com/en-us/distribution-for-python

conda create –c intel intelpython3_full

docker pull intelpython/intelpython3_full

pip install intel-numpy intel-scipy intel-scikit-learn

Accelerated NumPy, SciPy

- Intel® MKL
- Intel® C and Fortran compilers
- Linear algebra, universal functions, FFT

Accelerated Scikit-Learn

- Intel® MKL
- Intel® C and Fortran compilers
  } via NumPy/Scipy
- Intel® Data Analytics Acceleration Library (DAAL)

Solutions for efficient parallelism

- TBB4py
- github.com/IntelPython/smp
- Intel® MPI library

Python APIs for Intel® MKL functions

- github.com/IntelPython/mkl_fft
- github.com/IntelPython/mkl_random
- github.com/IntelPython/mkl-service

Python APIs for Intel® DAAL

- github.com/IntelPython/daal4py

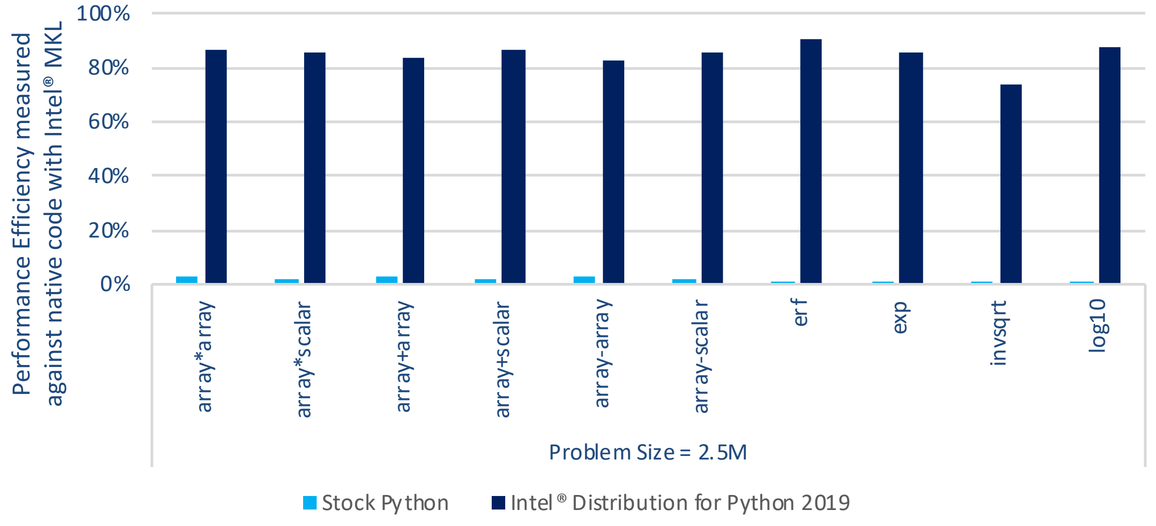
Numba with upstreamed Intel contributions

- Parallel Accelerator
- support for SVML
- support for TBB/OpenMP threading runtimes

https://software.intel.com/en-us/distribution-for-python/benchmarks

Xeon

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_np114py36_6,mkl_random 1.0.1 intel_np114py36_6, numba 0.39.0 intel_np114py36_0, llvmlite 0.24.0 intel_py36_0, scipy 1.1.0 intel_np114py36_6, scikit-learn 0.19.1 intel_np114py36_35; 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.

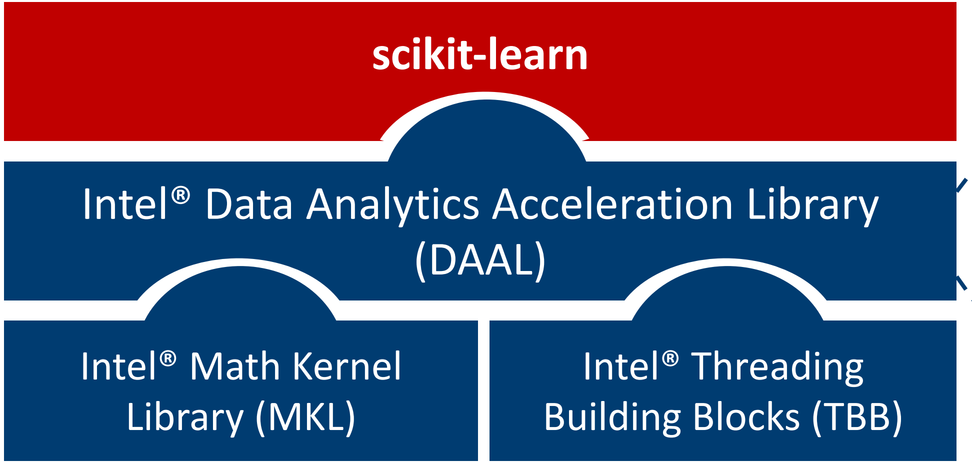
NumPy sources/recipes

https://github.com/IntelPython/numpy/

Sources with IDP patches applied

conda recipes included

Accelerating Machine Learning



- Efficient memory layout via `Numeric Tables`
- Blocking for optimal cache performance
- Computation mapped to efficient operations (MKL)
- Vectorization
- Parallelization via TBB

Try it out! `conda install -c intel scikit-learn`

Intel® DAAL Algorithms supported by daal4py

Data Transformation and Analysis

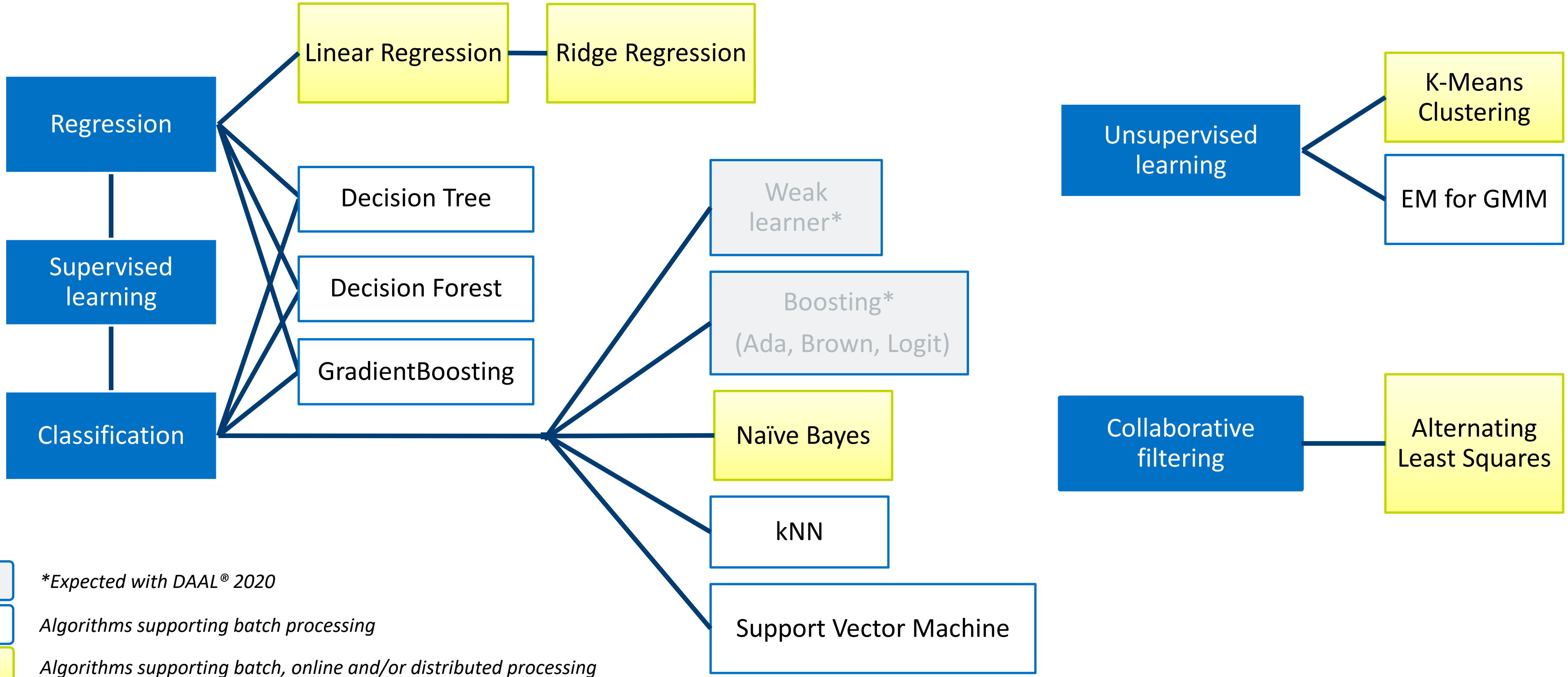
- Basic statistics for datasets
  - Low order moments
  - Quantiles
  - Order statistics
- Correlation and dependence
  - Cosine distance
  - Correlation distance
  - Variance-Covariance matrix
- Matrix factorizations
  - SVD
  - QR
  - Cholesky
- Dimensionality reduction
  - PCA
- Outlier detection
  - Univariate
  - Multivariate
- Association rule mining (Apriori)
- Optimization solvers (SGD, AdaGrad, lBFGS)
- Math functions (exp, log,…)

Algorithms supporting batch processing

Algorithms supporting batch, online and/or distributed processing

Intel® DAAL Algorithms supported by daal4py

Machine Learning



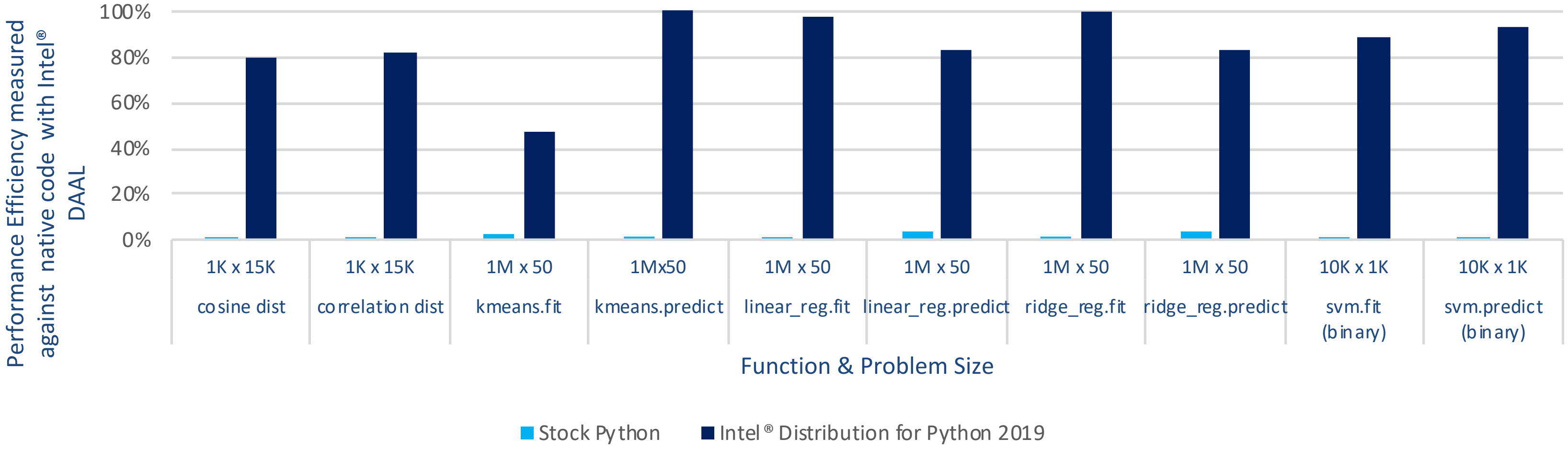
**Expected with DAAL® 2020*

Algorithms supporting batch processing

Algorithms supporting batch, online and/or distributed processing

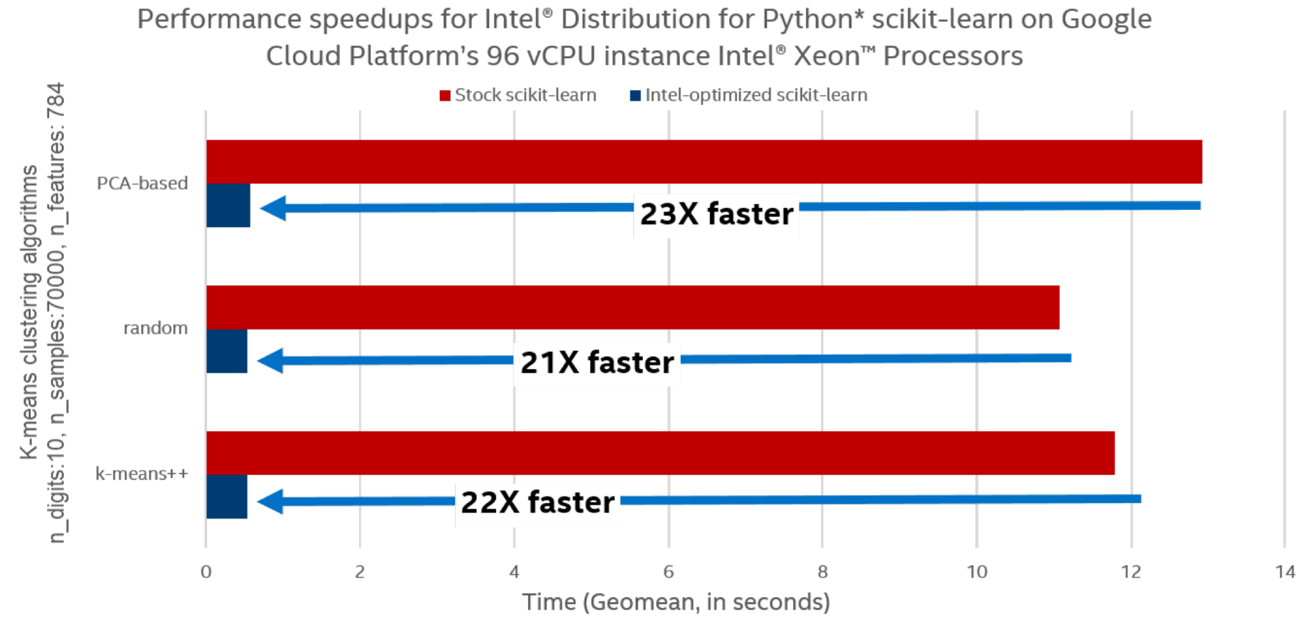
Close to native code scikit-learn Performance with Intel Python 2019

Compared to Stock Python packages on Intel® Xeon processors



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_np114py36_6,mkl_random 1.0.1 intel_np114py36_6, numba 0.39.0 intel_np114py36_0, llvmlite 0.24.0 intel_py36_0, scipy 1.1.0 intel_np114py36_6, scikit-learn 0.19.1 intel_np114py36_35; 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.

Accelerating K-Means



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 2017-09-12T20:10:42.862965559Z); Stock Python*: pip install scikit-learn

https://cloudplatform.googleblog.com/2017/11/Intel-performance-libraries-and-python-distribution-enhance-performance-and-scaling-of-Intel-Xeon-Scalable-processors-on-GCP.html

Accelerating scikit-learn through daal4py

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

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

```
import daal4py.sklearn
daal4py.sklearn.patch_sklearn()
```

Monkey-patch any scikit-learn programmatically

```
import daal4py.sklearn
daal4py.sklearn.linear_model.Ridge().fit(X, y)
```

Use alongside scikit-learn

Scikit-learn with daal4py patches applied passes scikit-learn test-suite (run on public CI)

PCA
KMeans
LinearRegression
Ridge
SVC
pairwise_distances
logistic_regression_path

Scikit-Learn **Equivalents**

Scikit-Learn **API** Compatible

KNeighborsClassifier
RandomForestClassifier
RandomForestRegressor

Use directly for

- Scaling to multiple nodes
- Streaming data
- Non-homogeneous dataframes

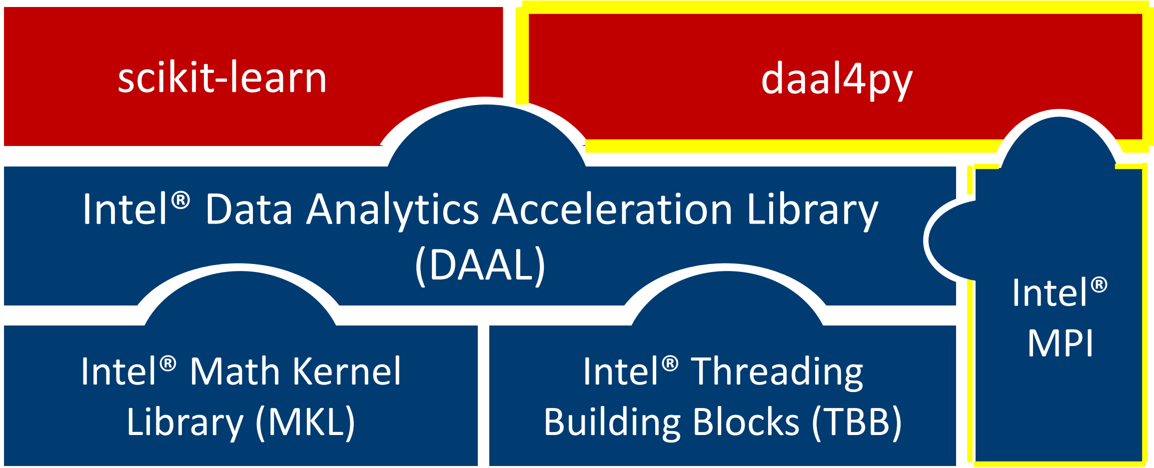
daal4py

Intel® DAAL

Get to 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`

K-Means using daal4py

```
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)
```

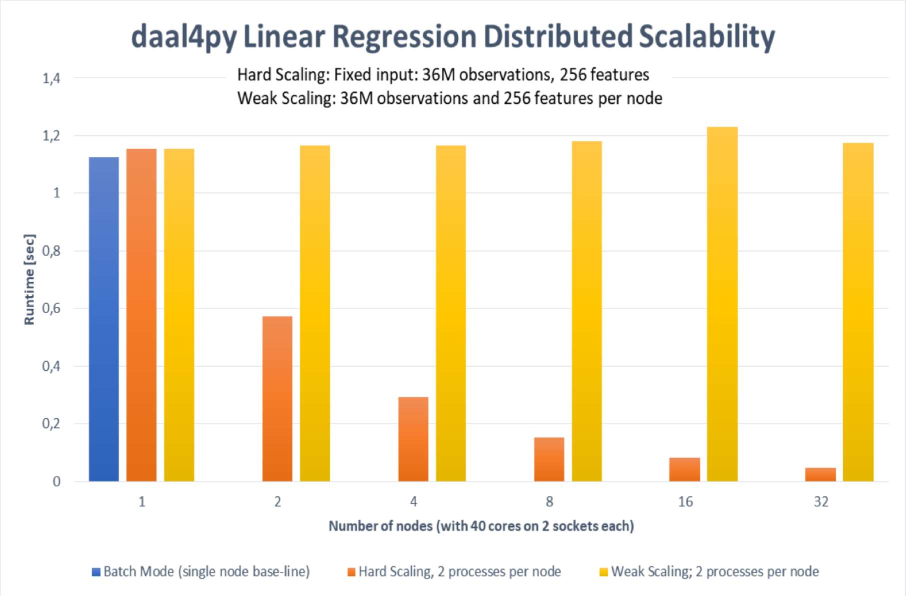
Distributed K-Means using daal4py

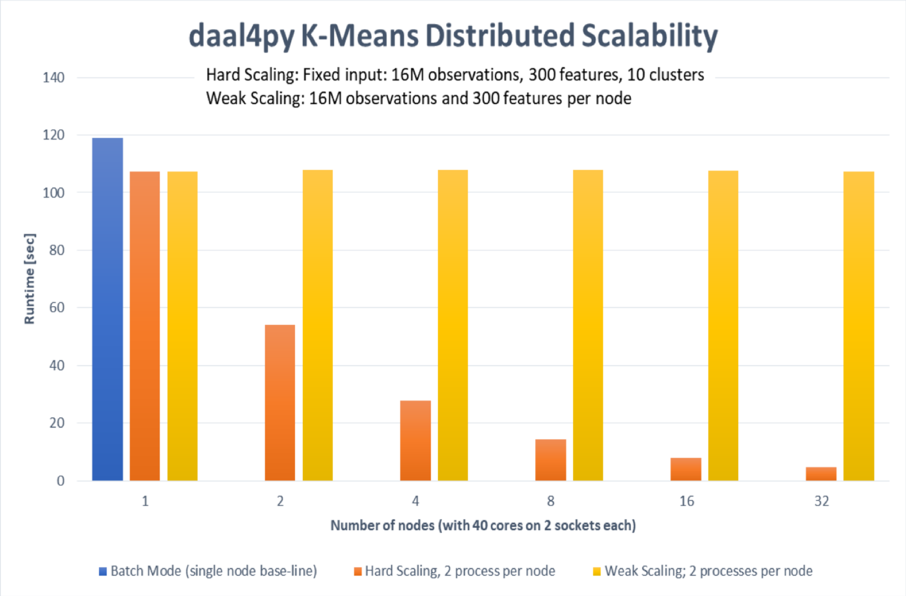
```
import daal4py as d4p

# 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
```

Strong & Weak Scaling via daal4py

Hardware	Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz, EIST/Turbo on
	2 sockets, 20 Cores per socket
	192 GB RAM
	16 nodes connected with Infiniband
Operating System	Oracle Linux Server release 7.4
Data Type	double

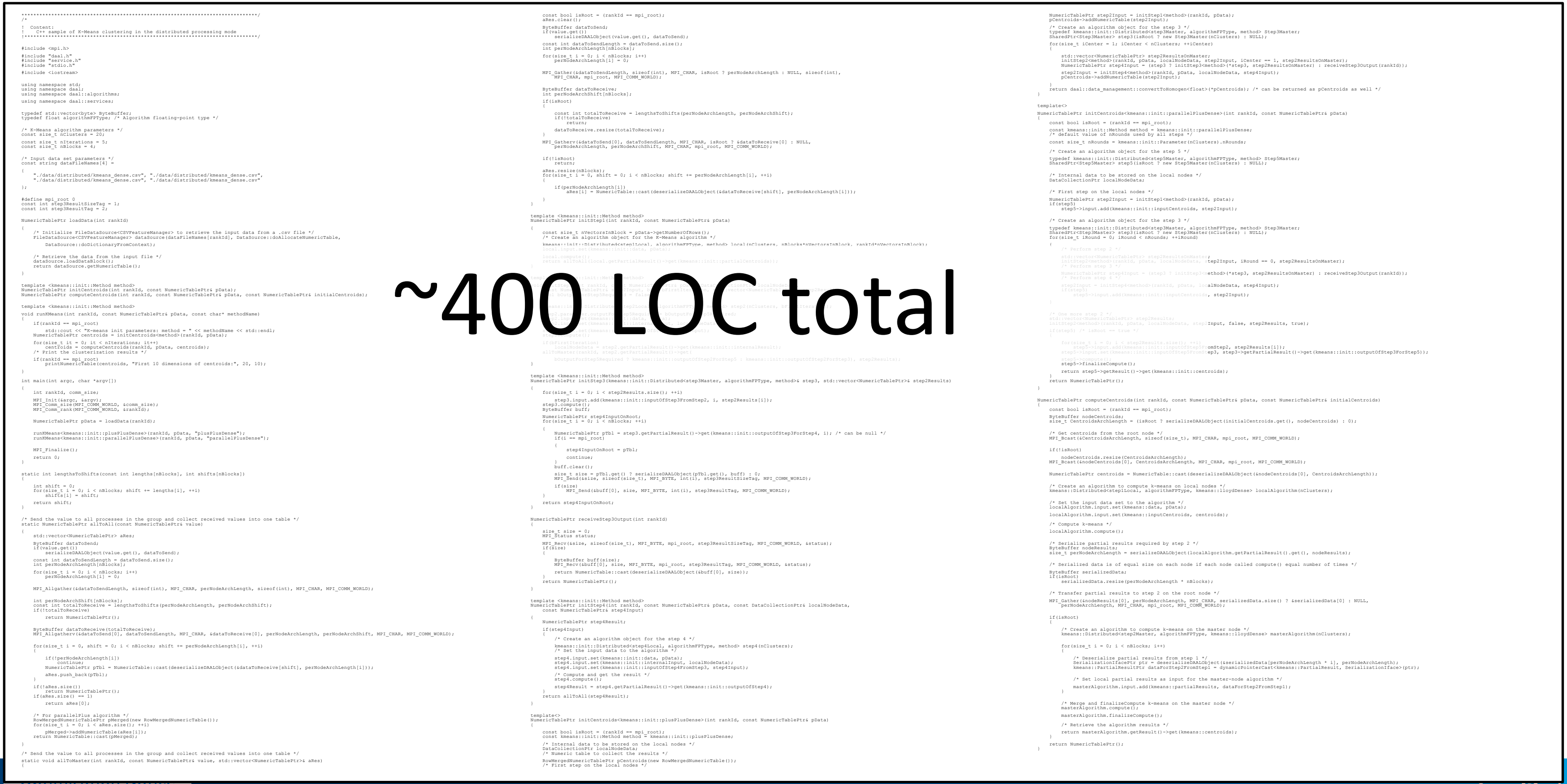




On a 32-node cluster (1280 cores) daal4py computed linear regression of 2.15 TB of data in 1.18 seconds and 68.66 GB of data in less than 48 milliseconds.

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.

Distributed K-Means Using DAAL (C++ API)



Streaming data (linear regression) using daal4py

```
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()
```

Scalable Python Solutions in Incubation

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

daal4py

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

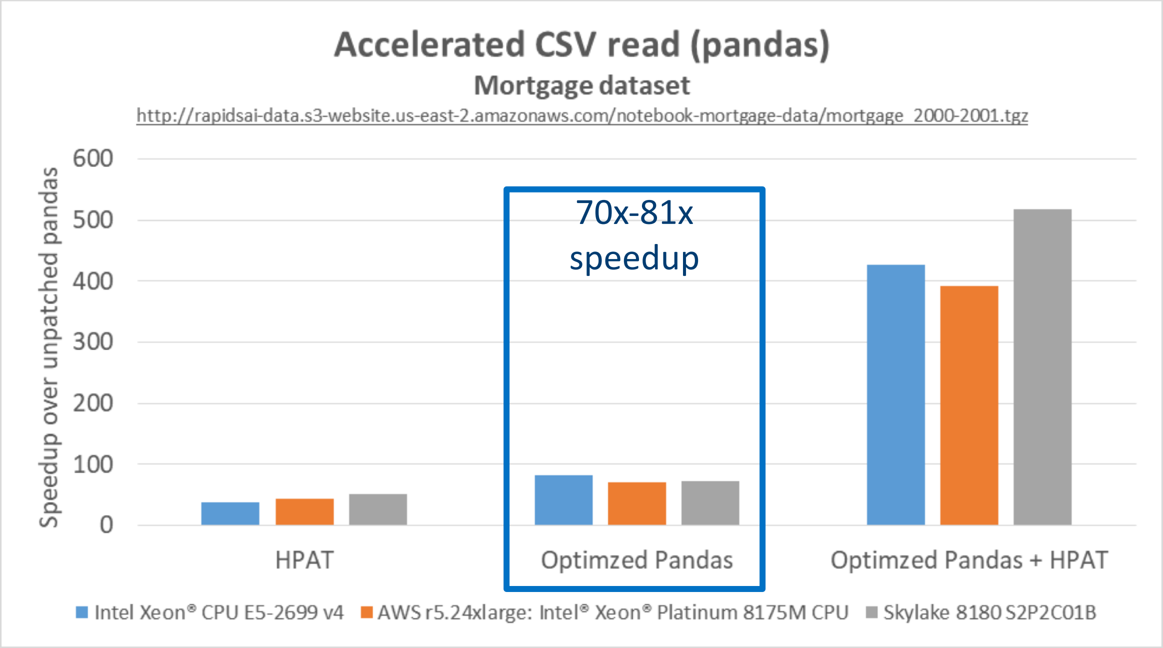
- High-level Python API for DAAL
- 10x fewer LOC wrt DAAL for single node,
  100x fewer LOC wrt DAAL for multi-node

Automatically scales to multiple nodes with MPI

Accelerating pandas CSV read

Patches merged to pandas mainline:

https://github.com/pandas-dev/pandas/pull/25804
https://github.com/pandas-dev/pandas/pull/25784



Intel(R) Xeon(R) CPU E5-2699 v4: 2.20GHz; 1chreads per core; 22 cores per socket; 2 sockets
Intel(R) Xeon(R) Platinum 8175M CPU: 2.50GHz; 2 threads per core; 24 cores per socket; 2 sockets
Skylake 8180 S2P2C01B: 2.5GHz
1 thread per core; 28 cores per socket; 2 sockets

Accelerating Pandas using HPAT

```
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
```

HPAT's Scope of Functionalities (Technical Preview)

Operations	• Python/Numpy basics • Statistical operations (mean, std, var, …) • Relational operations (filter, join, groupby) • Custom Python functions (apply, map)
Data	• Missing values • Time series, dates • Strings, unicode • Dictionaries • Pandas Extend Numba to support
Interoperability	• I/O integration (CSV, Parquet, HDF5, Xenon) • Daal4py integration

Intel® Distribution for Python*

https://anaconda.org/intel
https://software.intel.com/en-us/distribution-for-python
https://intelpython.github.io/daal4py
https://github.com/IntelLabs/hpat

Questions?

Backup

New in Intel Distribution for Python 2019 Update 3

- Updated version of scikit-learn (0.20.2)
- All scikit-learn patches are now in daal4py, see daa4lpy.sklearn
- Algorithmically:
  - daa4lpy.sklearn.neighbor.KNeighborsClassifier (DAAL's nearest neighbor classifier based on KDTrees)
  - Patched scikit-learn passes upstream's full test suite, with a few documented exceptions.

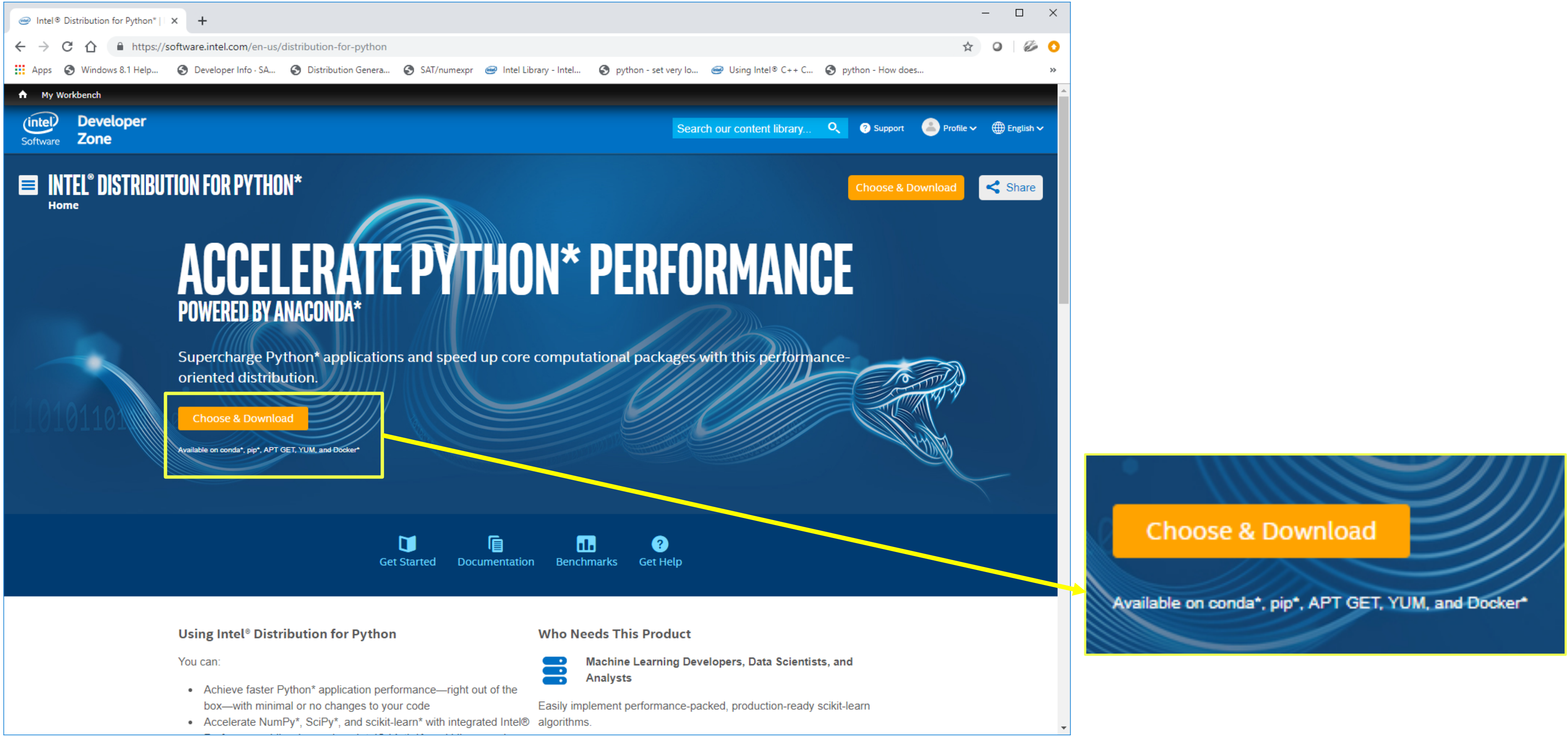
Logistic Regression acceleration

- Presently accelerated .fit method for solver='lbfgs' only (is to become the default in scikit-learn 0.21)
- Acceleration applies
  - for any number of classes
  - for multiclass='multinomial' or 'ovr'
  - to LogisticRegression, LogisticRegressionCV, logistic_path

RandomForest

- daal4py.sklearn.ensemble.RandomForestClassifier
- daal4py.sklearn.ensemble.RandomForestRegressor

- only support dense features, and single response
- produce similar output to scikit-learn's own classes, i.e. populate estimators_ attribute, so that it can be used in existing Python viz. pipeline.
- prediction is using DAAL's model, rather than estimators_


Intel® Distribution for Python*
https://software.intel.com/en-us/distribution-for-python
Apps
Windows 8.1 Help...
Developer Info · SA...
Distribution Genera...
SAT/numexpr
Intel Library - Intel...
python - set very lo...
Using Intel® C++ C...
python - How does...
My Workbench
intel Software
Developer Zone
Search our content library...
Support
Profile
English
INTEL® DISTRIBUTION FOR PYTHON*
Home
Choose & Download
Share
ACCELERATE PYTHON* PERFORMANCE
POWERED BY ANACONDA*
Supercharge Python* applications and speed up core computational packages with this performance-oriented distribution.
Choose & Download
Available on conda*, pip*, APT GET, YUM, and Docker*
Get Started
Documentation
Benchmarks
Get Help
Using Intel® Distribution for Python
You can:
Achieve faster Python* application performance—right out of the box—with minimal or no changes to your code
Accelerate NumPy*, SciPy*, and scikit-learn* with integrated Intel®
Who Needs This Product
Machine Learning Developers, Data Scientists, and Analysts
Easily implement performance-packed, production-ready scikit-learn algorithms.
Choose & Download
Available on conda*, pip*, APT GET, YUM, and Docker*

On Cori

module load/python3.6-anaconda-4.4

conda create -n idp -c intel python=3.6 numpy scipy scikit-learn

source activate idp

Software

Legal Disclaimer & Optimization Notice

The benchmark results reported above may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations.

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.

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