Software

SPEEDING UP SCIKIT-LEARN ON INTEL ARCHITECTURES

Laurent DUHEM – Intel Software EMEA

Credit: Intel® Distribution for Python* engineering team

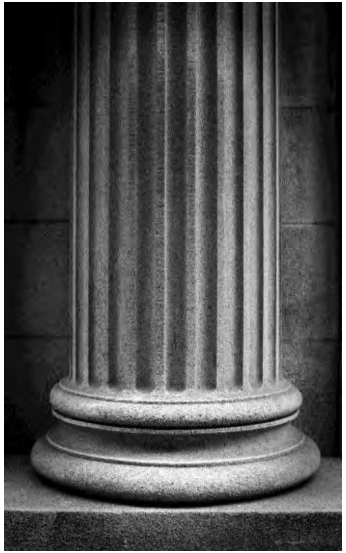
How Intel works in Open Source

Technical Contributions

Software Architects
Maintainers
Thousands of Software Engineers

Working Groups

Security
High Performance
Virtualization & Manageability
Power Efficiency
Connectivity
Graphics
Storage & Networking
Orchestration

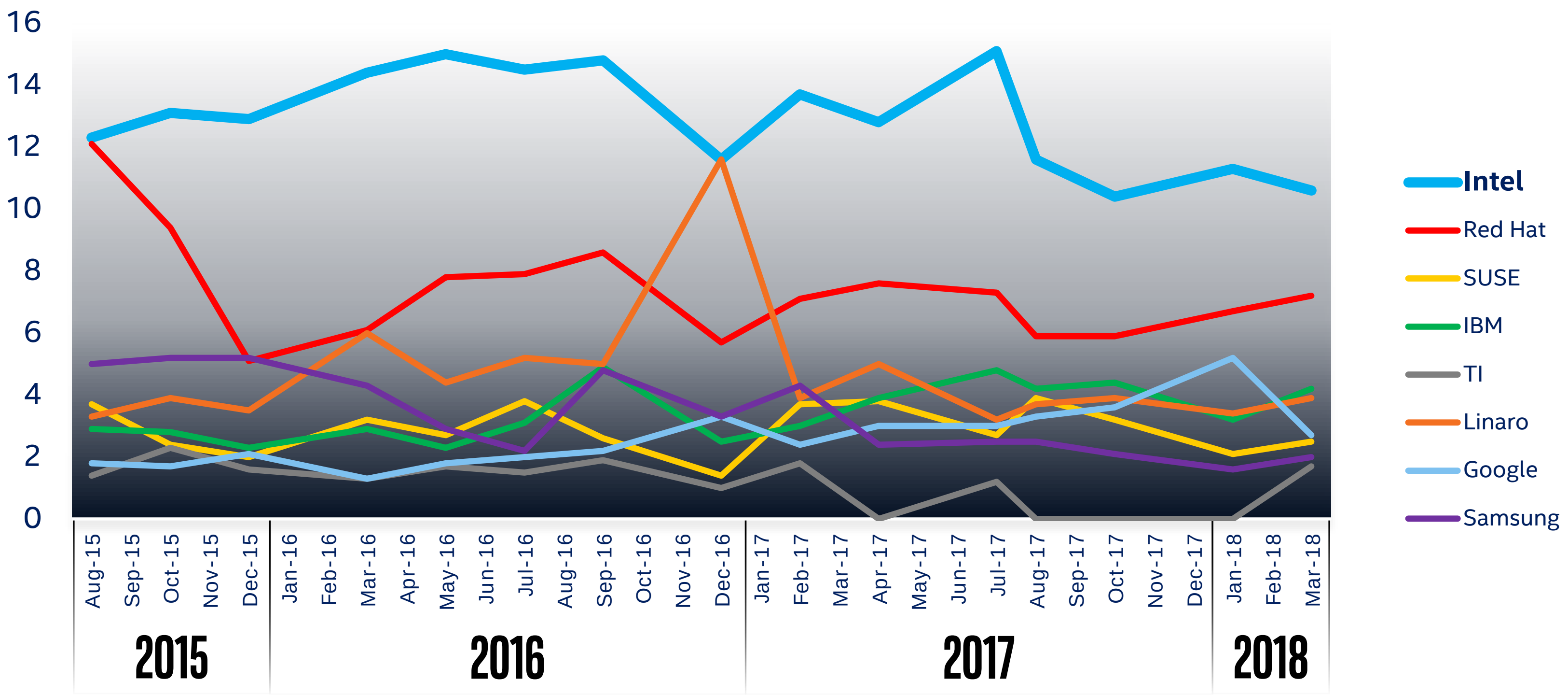


Foundation Participation

Apache Foundation
CNCF
LF Edge
Linux Foundation
OpenStack
...

Linux Kernel Contributions

By percentage



Source: lwn.net

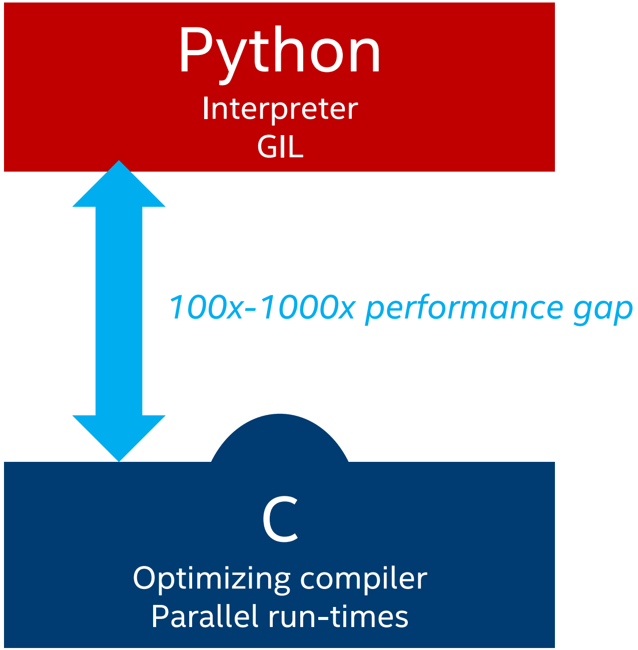
We Contribute Across System Software Stack

Software layers

Application layer	WebRTC, PyTorch, Caffe, theano, TensorFlow, BigDL on Spark, hadoop, Spark
Libraries and Runtimes	OpenJDK, node JS, WA, python, php, R
Orchestration	openstack, kubernetes, Azure
Hypervisor/ container	KVM, vmware, QEMU, kata containers, Microsoft Hyper-v
OS Enabling	AliOS, WIND, Zephyr, yocto PROJECT, redhat

The Reality of "Data Centric Computing"

Performance Limited	• Existing analytics solutions are inefficient (and non-scalable) • E.g. pandas
Productivity Limited	• Existing analytics scale-out solutions are complex or do not exist • E.g. nothing exists for pandas
Scalability Limited	• Typical data scientist only analyzes a small portion (probably 10%) of available data



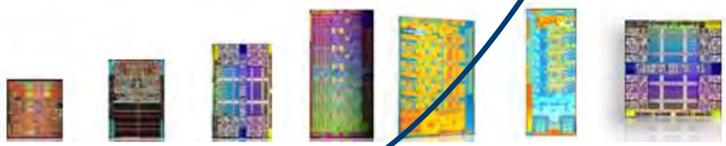
High Performance Python

Python Libraries

Thin layer in Python or Cython

Intel® Performance Libraries

Native highly optimized libraries: Intel® MKL[1], DAAL[2]
Vectorization support
threading runtimes



(generations of processors)

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

Deliver Python technologies that scale-up/out entire data analytics pipeline in productive way = Intel® Distribution for Python

[1]Intel® Math Kernel Library
[2]Intel® Data Analytics Acceleration Library

What's Inside Intel® Math Kernel Library

LINEAR ALGEBRA	FFTS	VECTOR RNGS	SUMMARY STATISTICS	VECTOR MATH	& MORE
BLAS	Multidimensional	Congruential	Kurtosis	Trigonometric	Splines
LAPACK	FFTW interfaces	Wichmann-Hill	Variation coefficient	Hyperbolic	Interpolation
ScaLAPACK	Cluster FFT	Mersenne Twister	Order statistics	Exponential	Trust Region
Sparse BLAS		Sobol	Min/max	Log	Fast Poisson Solver
Iterative sparse solvers		Neirderreiter	Variance-covariance	Power	
PARDISO*		Non-deterministic		Root	
Cluster Sparse Solver					

[1]Available only in Intel® Parallel Studio Composer Edition.

Speedup Analytics & Machine Learning with Intel® Data Analytics Acceleration Library (Intel® DAAL)

- Highly tuned functions for classical machine learning & analytics performance from datacenter to edge running on Intel® processor-based devices
- Simultaneously ingests data & computes results for highest throughput performance
- Supports batch, streaming & distributed usage models to meet a range of application needs
- Includes Python*, C++, Java* APIs, & connectors to popular data sources including Spark* & Hadoop

What's New in the 2019 Release

New Algorithms

- **Logistic Regression**, most widely-used classification algorithm
- **Extended Gradient Boosting Functionality** for inexact split calculations & user-defined callback canceling for greater flexibility
- **User-defined Data Modification Procedure** supports a wide range of feature extraction & transformation techniques

Learn More: software.intel.com/daal

Pre-processing → **Transformation** → **Analysis** → **Modeling** → **Validation** → **Decision Making**

Decompression, Filtering, Normalization

Aggregation, Dimension Reduction

Summary Statistics Clustering, etc.

Machine Learning (Training) Parameter Estimation Simulation

Hypothesis Testing Model Errors

Forecasting Decision Trees, etc.

Accelerate Python* with Intel® Distribution for Python*

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

FASTER PERFORMANCE	GREATER PRODUCTIVITY	ECOSYSTEM COMPATIBILITY
Performance Libraries, Parallelism, Multithreading, Language Extensions	**Prebuilt & Accelerated Packages**	**Supports Python 2.7 & 3.6, Conda & PIP**
▪ Accelerated NumPy/SciPy with Intel® MKL[1] & Intel® DAAL[2] ▪ Data analytics, machine learning & deep learning with scikit-learn, daal4py ▪ Scale with Numba* & Cython* ▪ Includes optimized mpi4py, works with Dask* & PySpark* ▪ Optimized for latest Intel® architecture	▪ Prebuilt & optimized packages for numerical computing, machine/deep learning, HPC, & data analytics ▪ Drop in replacement for existing Python- No code changes required ▪ Jupyter* notebooks, Matplotlib included ▪ Free download & free for all uses including commercial deployment	▪ Supports Python 2.7 & 3.6, optimizations integrated in Anaconda* Distribution ▪ Distribution & optimized packages available via Conda, PIP, APT GET, YUM, & DockerHub, numerical performance optimizations integrated in Anaconda Distribution ▪ Optimizations upstreamed to main Python trunk ▪ Priority Support with Intel® Parallel Studio XE

Operating System: Windows*, Linux*, MacOS[1]*

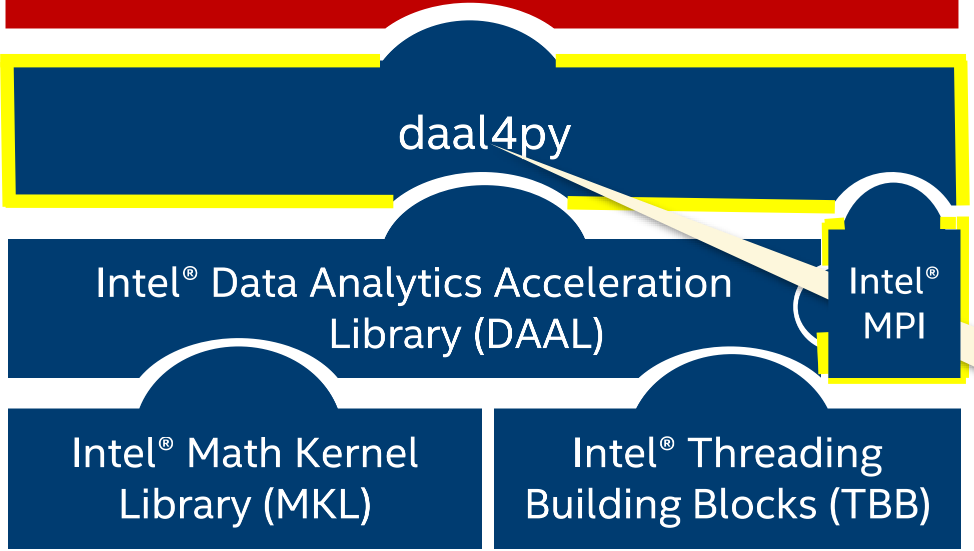
Intel® Architecture Platforms

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

[1]Intel® Math Kernel Library
[2]Intel® Data Analytics Acceleration Library

[1]Available only in Intel® Parallel Studio Composer Edition.

Accelerating Machine Learning



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

Scikit-Learn Equivalents

PCA
KMeans
LinearRegression
Ridge
SVC
pairwise_distances
logistic_regression_path

Scikit-Learn **API** Compatible

daal4py.sklearn

KNeighborsClassifier
RandomForestClassifier
RandomForestRegressor

Use directly for

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

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

Scikit-learn with daal4py patches applied passes scikit-learn test-suite

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)

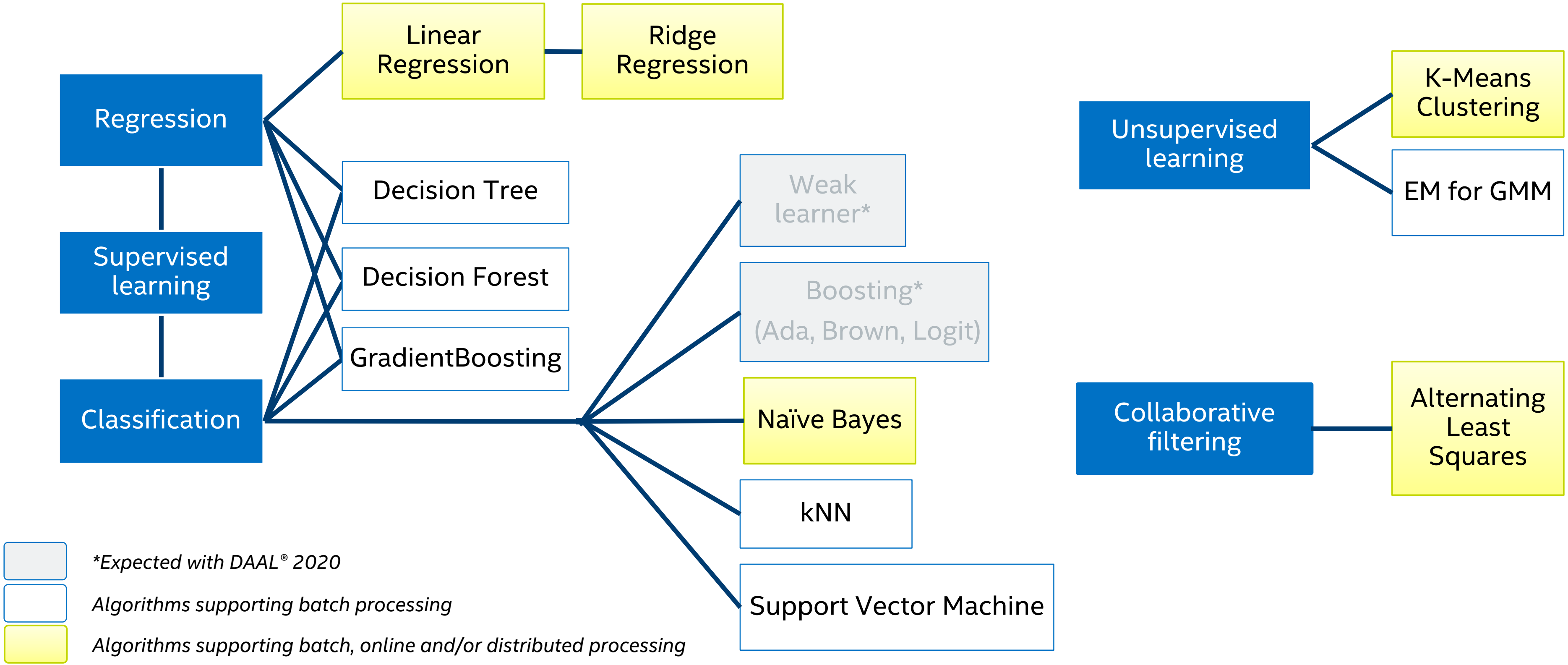
Math functions (exp, log,…)

Algorithms supporting batch processing

Algorithms supporting batch, online and/or distributed processing

Intel® DAAL Algorithms supported by daal4py

Machine Learning



K-Means using daal4py

```python
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)
```

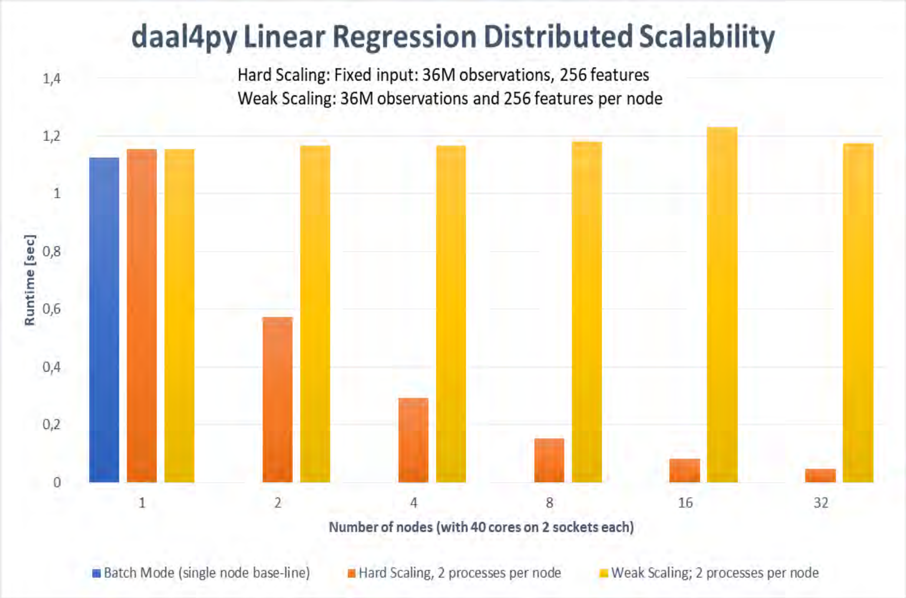
Distributed K-Means using daal4py

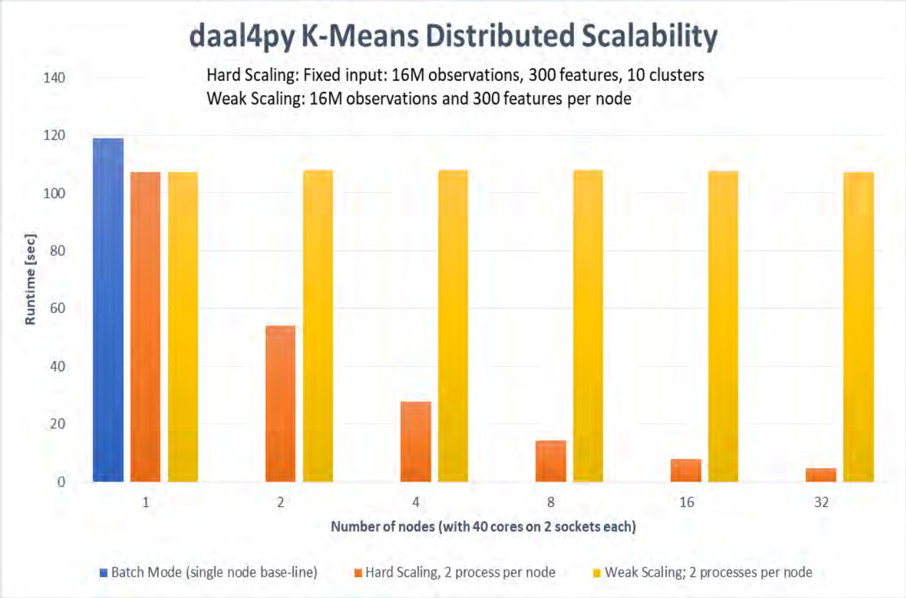
```
import daal4py as d4p

# 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.

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

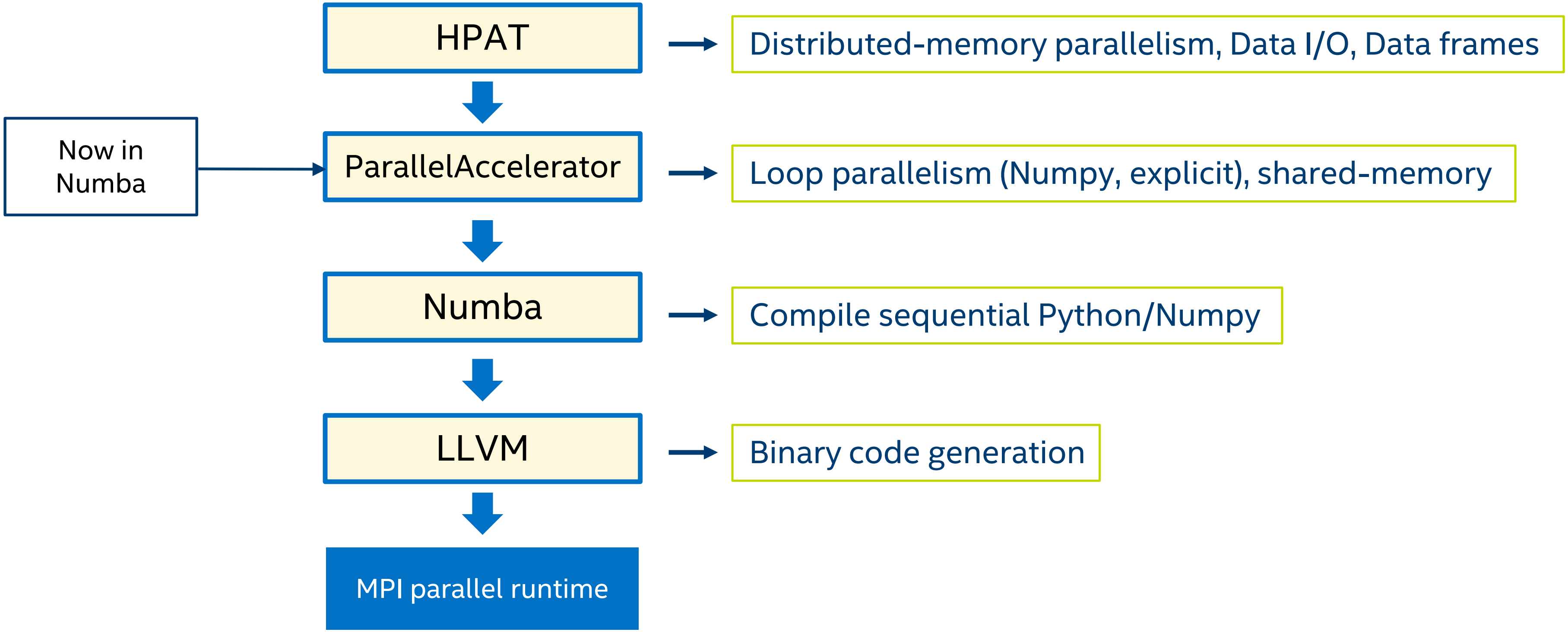
daal4py

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

Software Architecture



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 PYTHON

EXAMPLES AND OTHER RESOURCES FOR INTEL DISTRIBUTION FOR PYTHON

[HTTPS://GITHUB.COM/INTELPYTHON](https://github.com/intelpython)

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

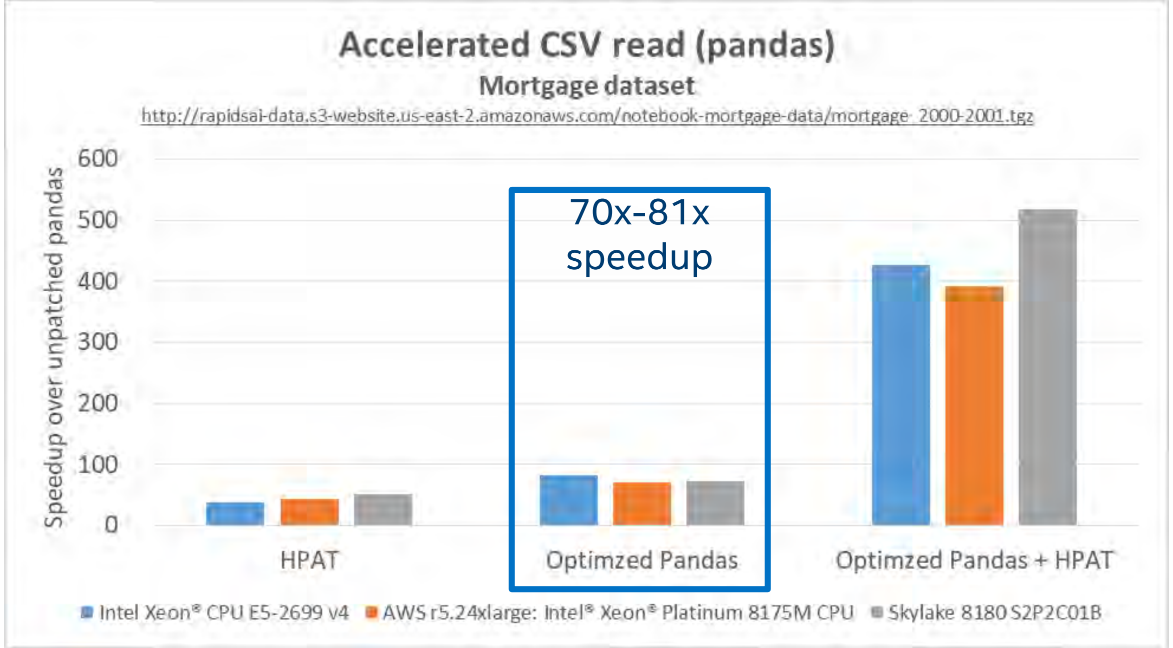
Questions?

BACKUP

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

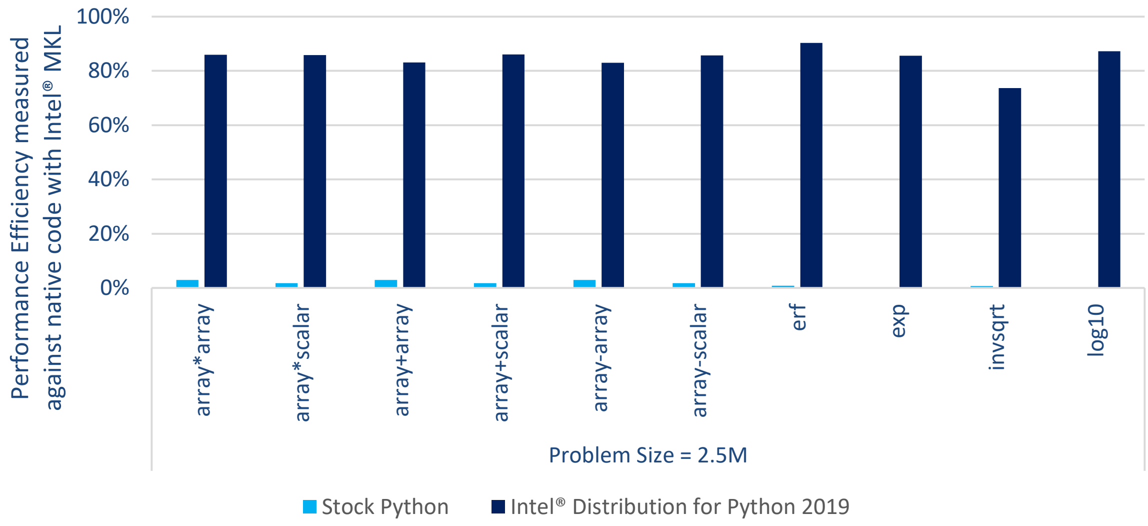
```
import pandas as pd
import hpat

@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
```

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.***

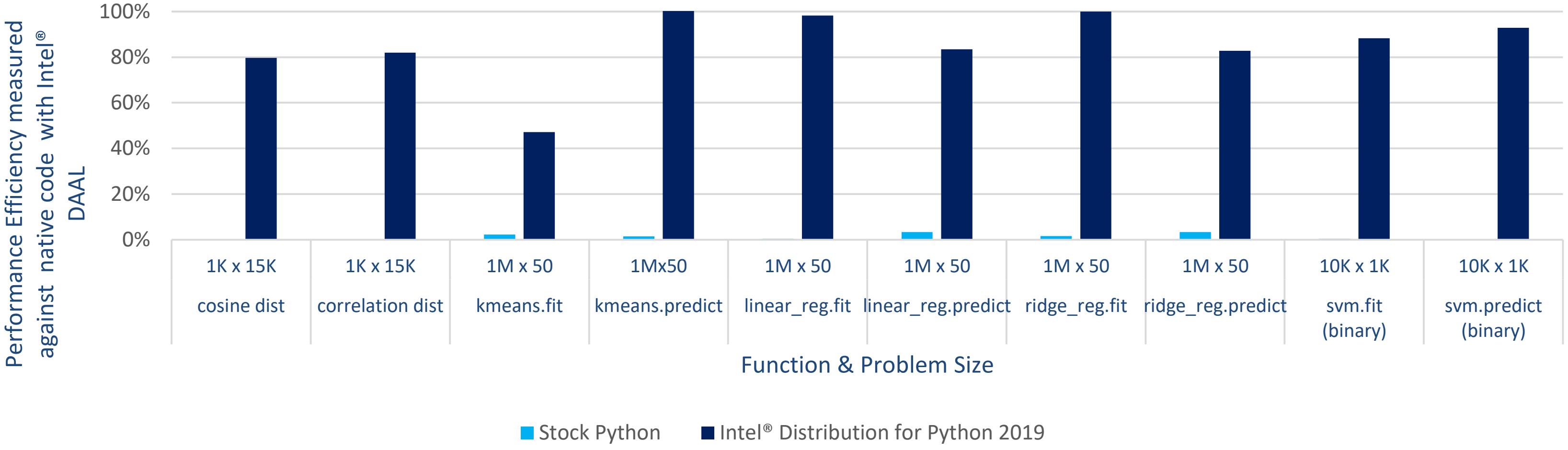
Configuration: Stock Python: python 3.6.6 hc3d631a_0 installed from conda, numpy 1.15, numba 0.39.0, llvmlite 0.24.0, scipy 1.1.0, scikit-learn 0.19.2 installed from pip;Intel Python: Intel Distribution for Python 2019 Gold: python 3.6.5 intel_11, numpy 1.14.3 intel_py36_5, mkl 2019.0 intel_101, mkl_fft 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.

Xeon

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.

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_

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