NUMBA/HPAT AND DAAL4PY: THE PAINLESS ROUTE IN PYTHON TO FAST AND SCALABLE DATA-ANALYTICS/MACHINE-LEARNING

LEGAL DISCLAIMER & OPTIMIZATION NOTICE

Performance results are based on testing as of August and September 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 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

THE REALITY OF "DATA CENTRIC COMPUTING"

Software Challenges:

Performance Limited	• Software is slow and single-node for many organizations • Only sample a small portion of the data
Productivity Limited	• More performant/scalable implementations require significantly more development & deployment skills & time
Compute Limited	• Performance bottleneck often in compute, not storage/memory

A typical data scientist only analyzes a small portion (probably 10%) of your data that they think has the most potential of bringing you great insights. This means you may miss out on valuable insights in the remaining 90% — insights that may be mission-critical for your business.

PRODUCTIVITY WITH PERFORMANCE VIA INTEL® PYTHON*

Intel® Distribution for Python*

scikit learn | NumPy | SciPy | Numba | pandas $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$ | mpi4py | SMP | •••

Easy, out-of-the-box access to high performance Python

- Prebuilt accelerated solutions for data analytics, numerical computing, etc.
- Drop in replacement for your existing Python. No code changes required.

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
pip install intel-numpy intel-scipy intel-scikit-learn
docker pull intelpython/intelpython3_full
```

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

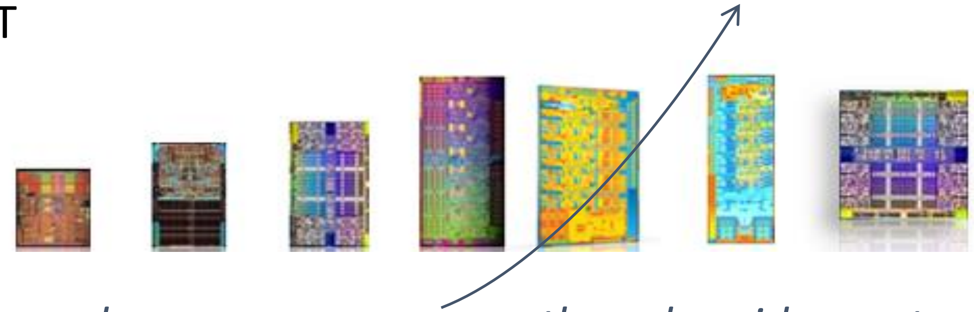
https://software.intel.com/en-us/distribution-for-python/benchmarks

DATA ANALYSIS AND MACHINE LEARNING

Data Input → Data Preprocessing → Model Creation → Prediction

Pandas
Spark
HPAT

Scikit-learn
Spark
DL-frameworks
daal4py



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

ACCELERATING MACHINE LEARNING

scikit-learn

Intel® Data Analytics Acceleration Library (DAAL)

Intel® Math Kernel Library (MKL)

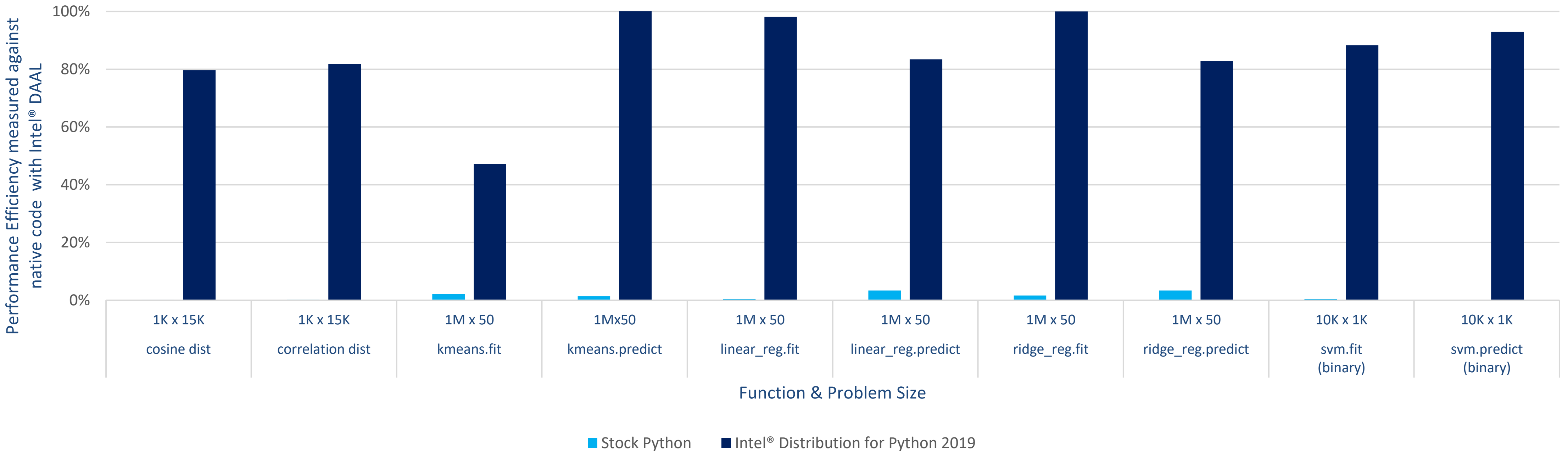
Intel® Threading Building Blocks (TBB)

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

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

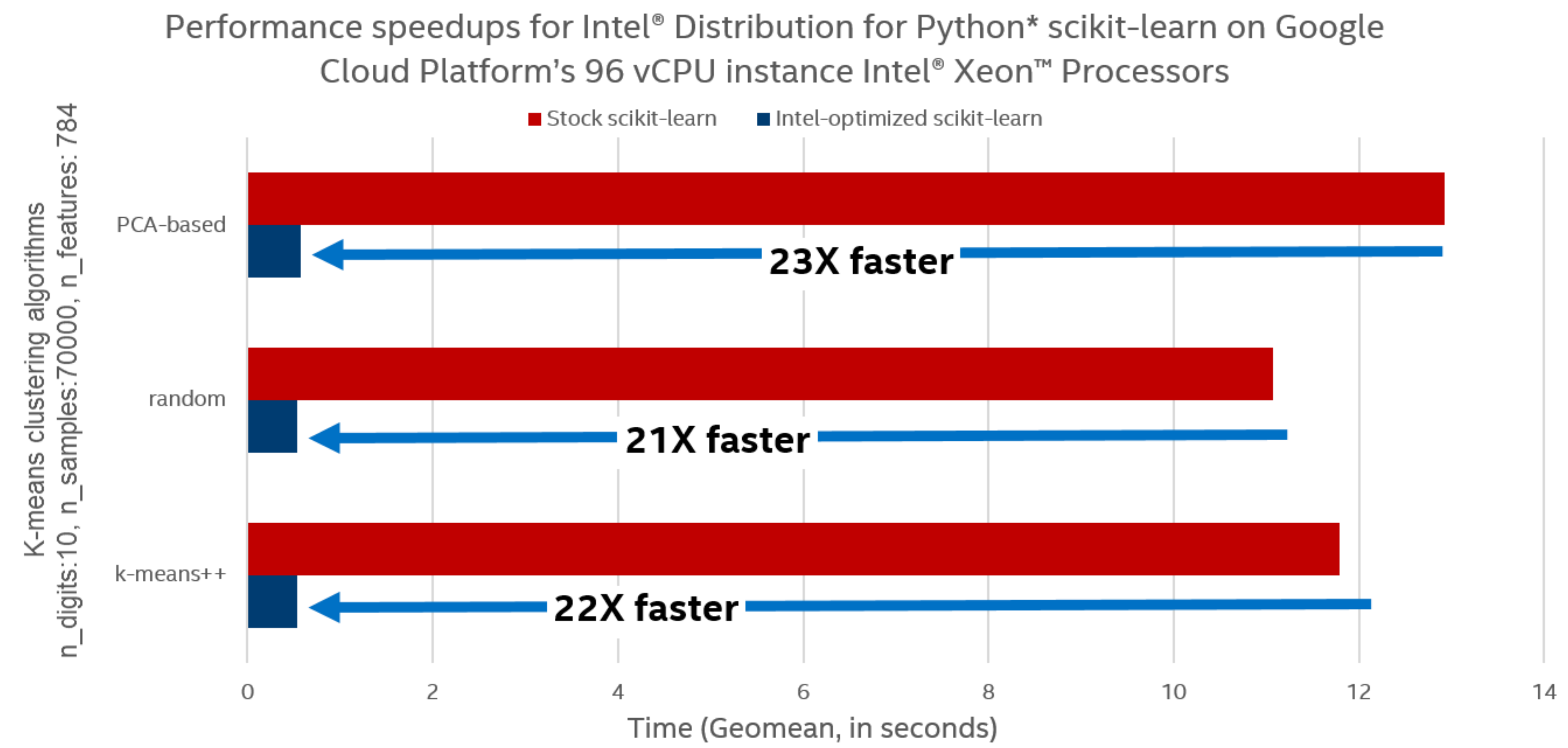
CLOSE TO NATIVE CODE SCIKIT-LEARN PERFORMANCE WITH INTEL PYTHON 2019

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

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

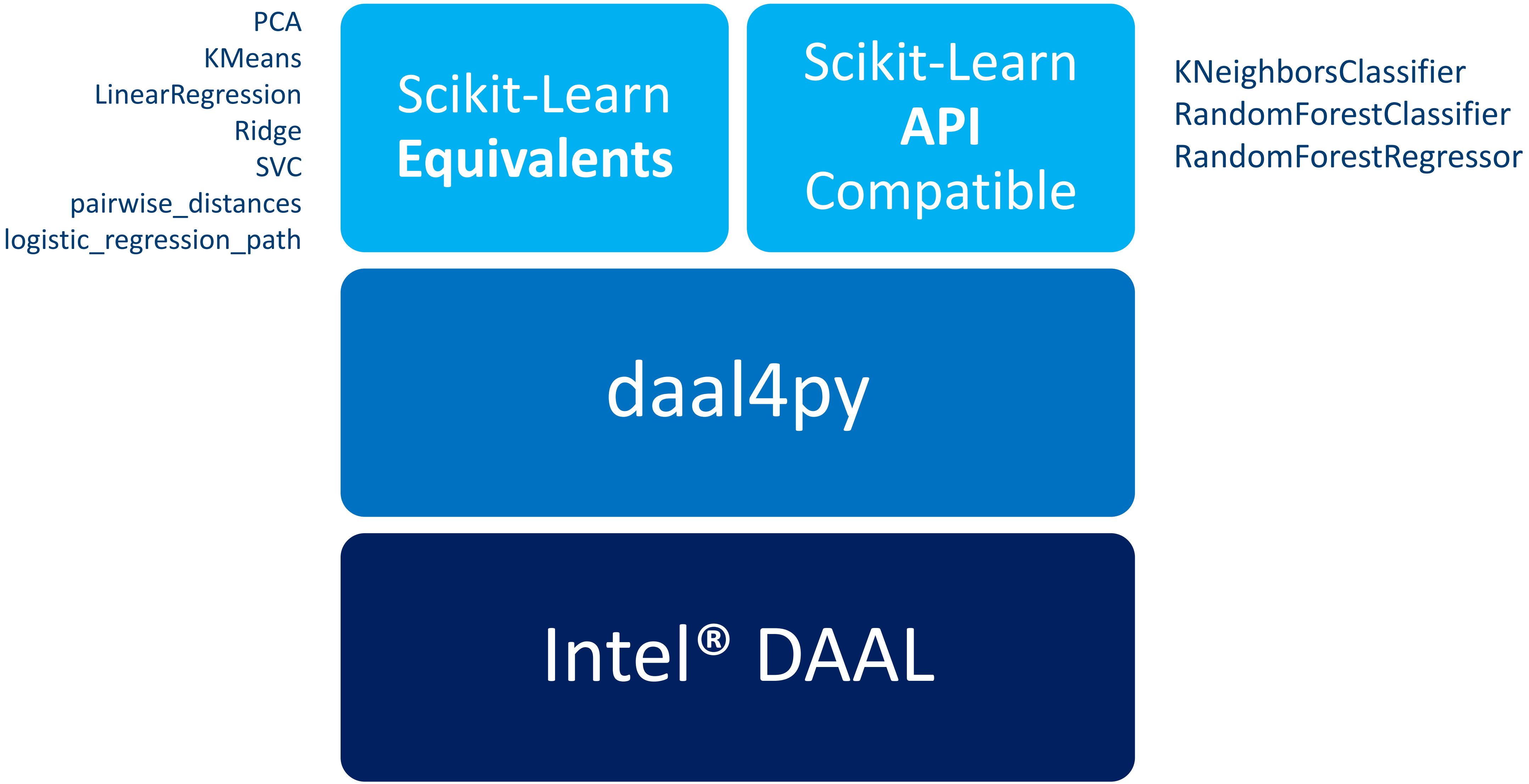
```
> python -m daal4py <your-scikit-learn-script>
```

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

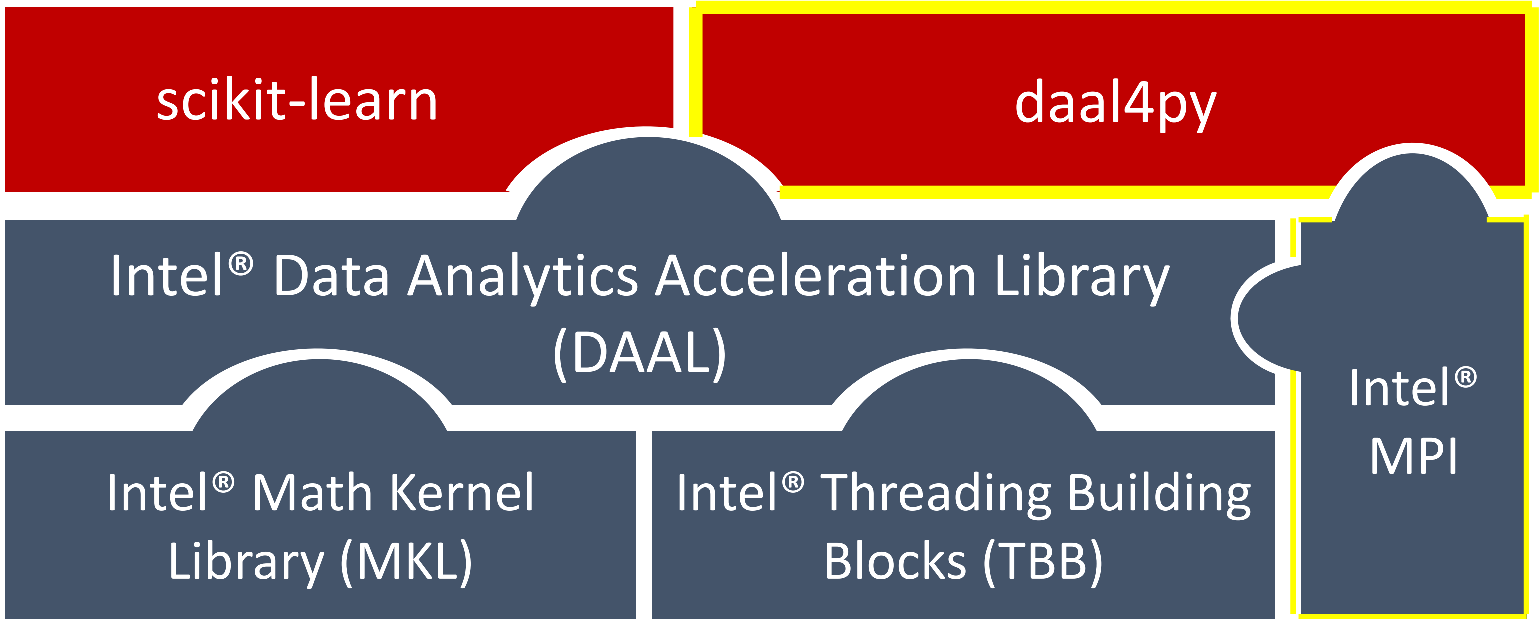
```
import daal4py.sklearn
daal4py.sklearn.patch_sklearn()
```

Monkey-patch any scikit-learn programmatically

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


PCA
KMeans
LinearRegression
Ridge
SVC
pairwise_distances
logistic_regression_path
Scikit-Learn Equivalents
Scikit-Learn API Compatible
KNeighborsClassifier
RandomForestClassifier
RandomForestRegressor
daal4py
Intel® DAAL

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

```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)
```

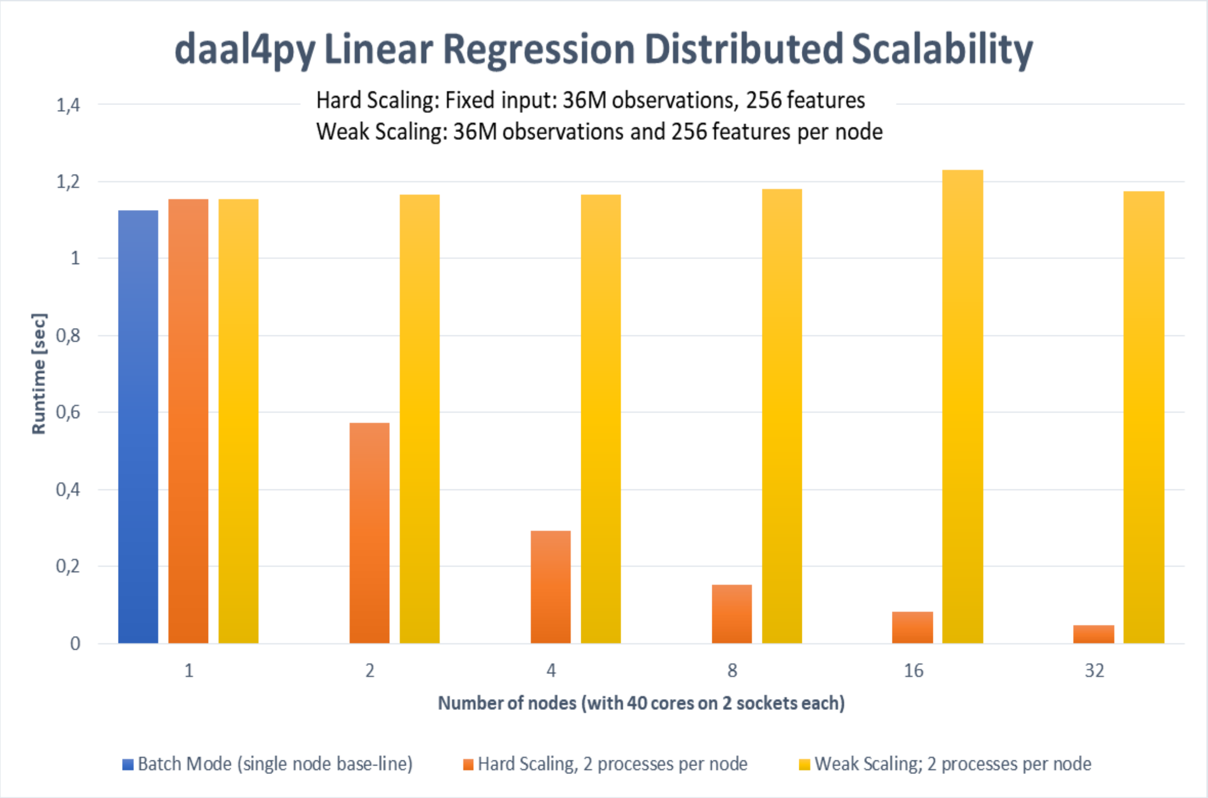
DISTRIBUTED K-MEANS USING DAAL4PY

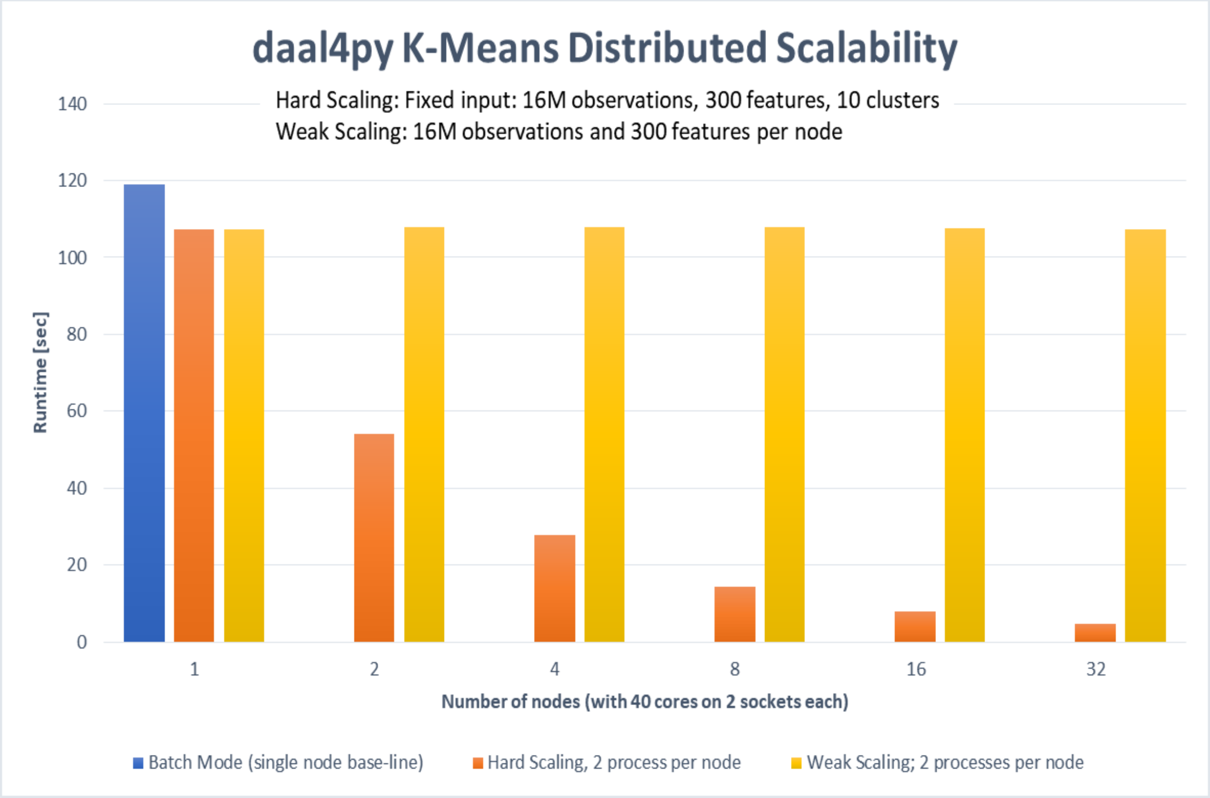
```
import daal4py as d4p

# 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.

STREAMING DATA (LINEAR REGRESSION) USING DAAL4PY

```python
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

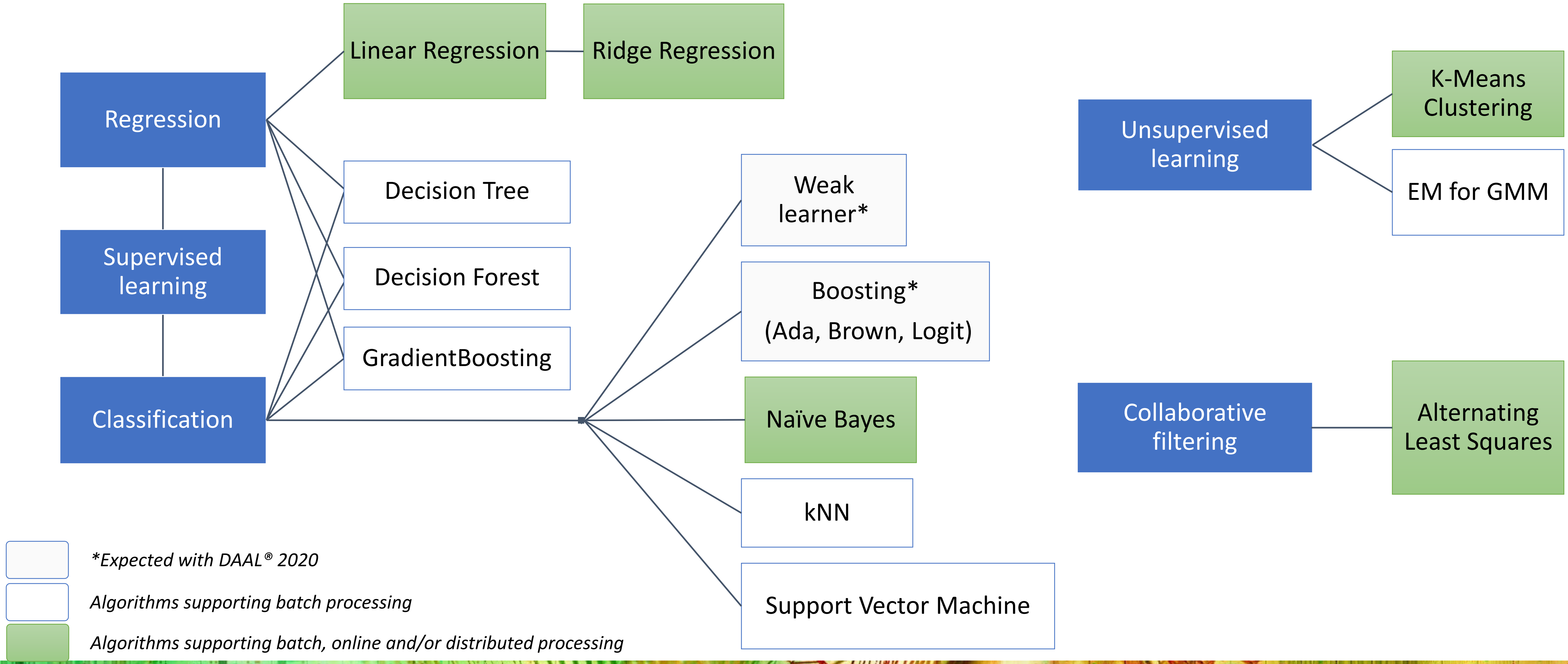
- **Basic statistics for datasets**
  - Low order moments
  - Quantiles
  - Order statistics
- **Correlation and dependence**
  - Cosine distance
  - Correlation distance
  - Variance-Covariance matrix
- **Matrix factorizations**
  - SVD
  - QR
  - Cholesky
- **Dimensionality reduction**
  - PCA
- Association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

Algorithms supporting batch processing

Algorithms supporting batch, online and/or distributed processing

INTEL® DAAL ALGORITHMS SUPPORTED BY DAAL4PY

MACHINE LEARNING



DAAL4PY

Fast & Scalable	• Close to native performance through Intel® DAAL • Efficient MPI scale-out • Streaming
Easy to use	• Known usage model • Picklable
Flexible	• Object model separating concerns • Plugs into scikit-learn • Plugs into HPAT
Open	• Open source: https://github.com/IntelPython/daal4py

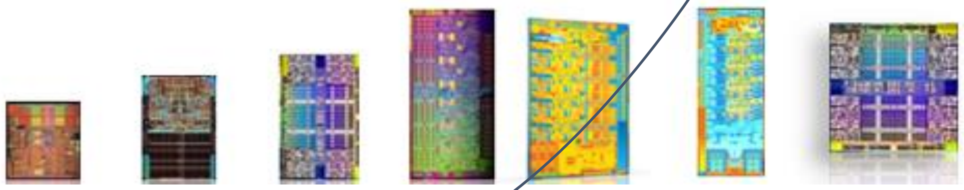
https://intelpython.github.io/daal4py/

DATA ANALYSIS AND MACHINE LEARNING

Data Input → Data Preprocessing → Model Creation → Prediction

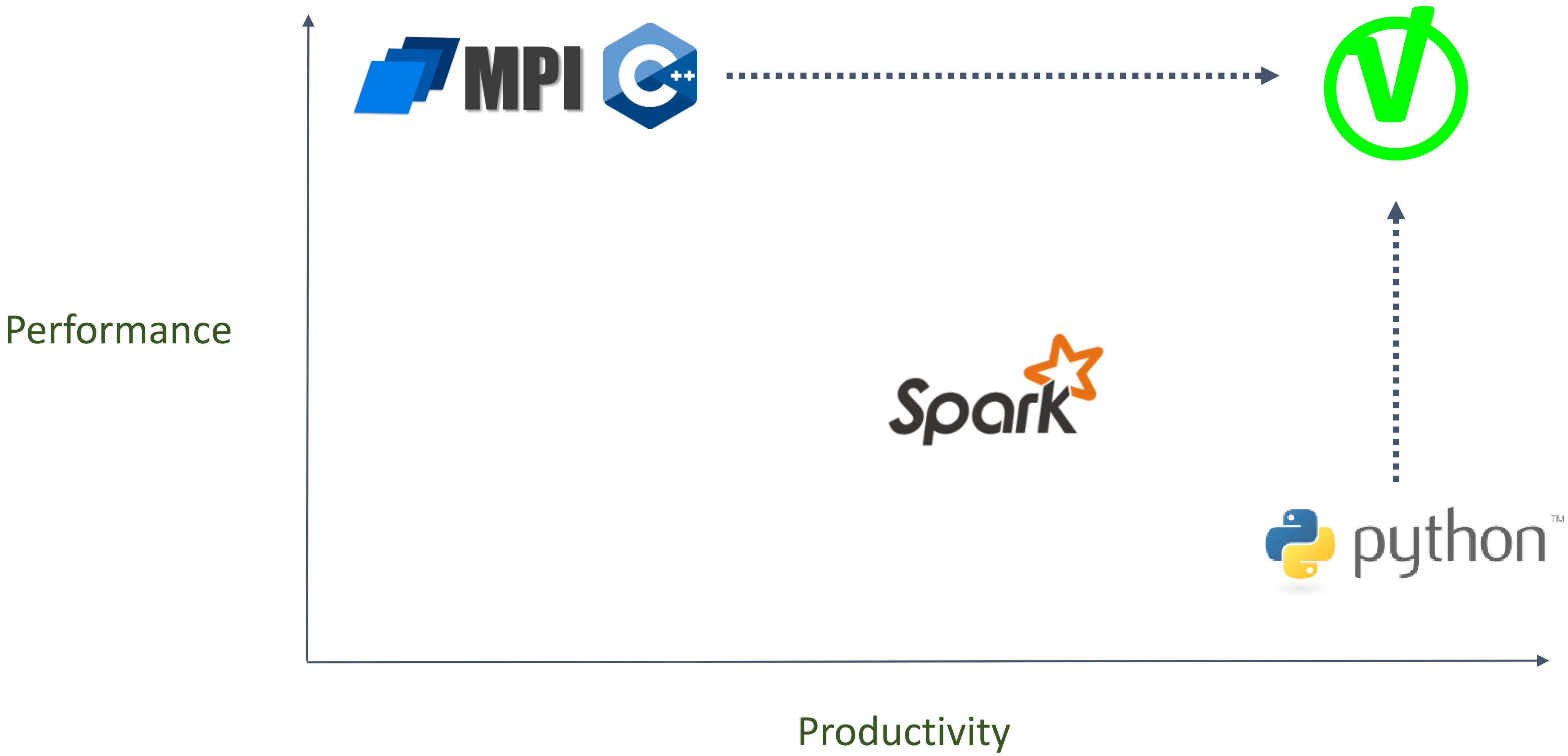
Pandas
Spark
HPAT

Scikit-learn
Spark
DL-frameworks
daal4py



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

DATA ANALYTICS PERFORMANCE VS PRODUCTIVITY



HIGH PERFORMANCE ANALYTICS TOOLKIT (HPAT)

Open source project by Intel Labs

- https://github.com/IntelPython/hpat

Technical Preview

- In beta by end of 2019

HIGH PERFORMANCE ANALYTICS TOOLKIT (HPAT)

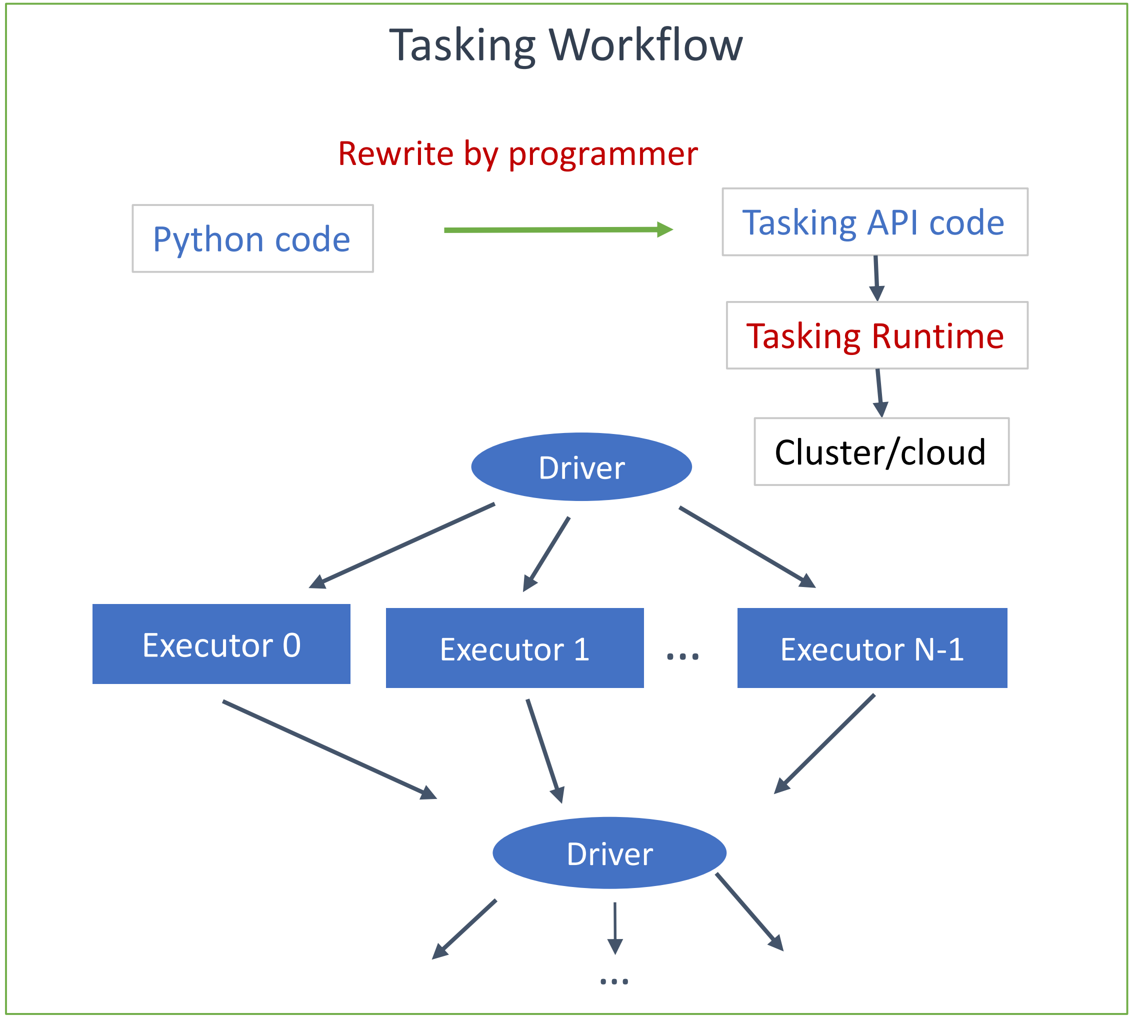
Technical Preview in open source

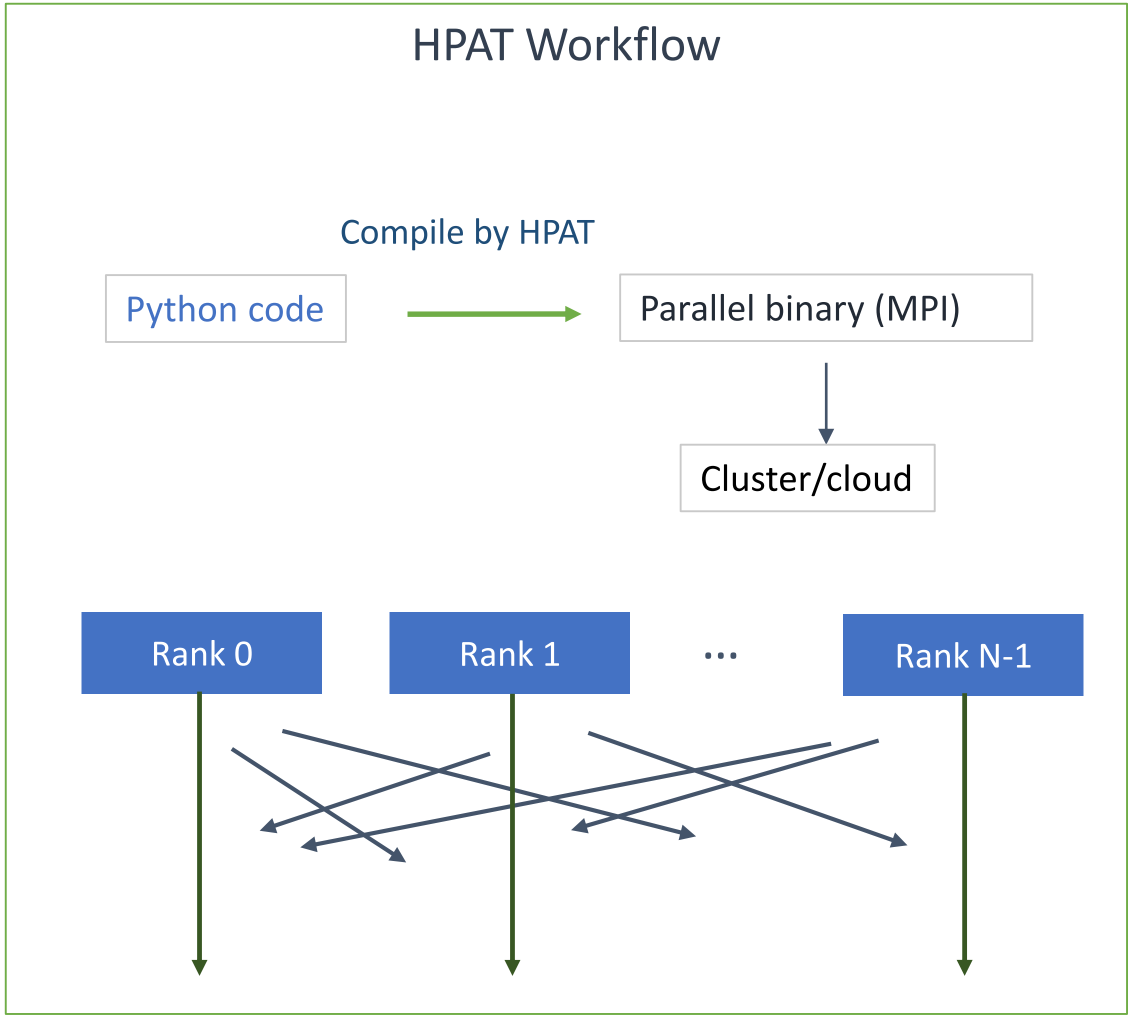
Decorator @hpat.jit → Parallel/distributed Analysis → Compile → Efficient MPI-binary

```python
@hpat.jit
def get_stats():
    …
    df['latency'].sum()
    df['latency'].mean()
    …
```

→

```
        vucomisd     %xmm0, %xmm0
        setnp        %dl
        jp           .LBB0_11
        vaddsd       %xmm0, %xmm2, %xmm2
.LBB0_11:
        vaddsd       %xmm0, %xmm3, %xmm1
        vcmpunordsd  %xmm0, %xmm0, %xmm0
        vblendvpd    %xmm0, %xmm3, %xmm1,
```





WAVES OF TINY TASKS

Long running processes

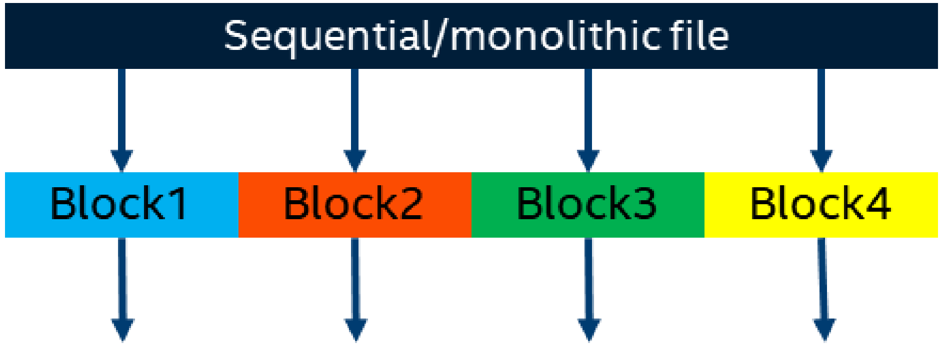
Totoni et al. "A Case Against Tiny Tasks in Iterative Analytics", *HotOS'17*

PARALLEL FILE-READ

Currently supports CSV, Parquet and HDF5

Block-parallel read parallelizes following operations

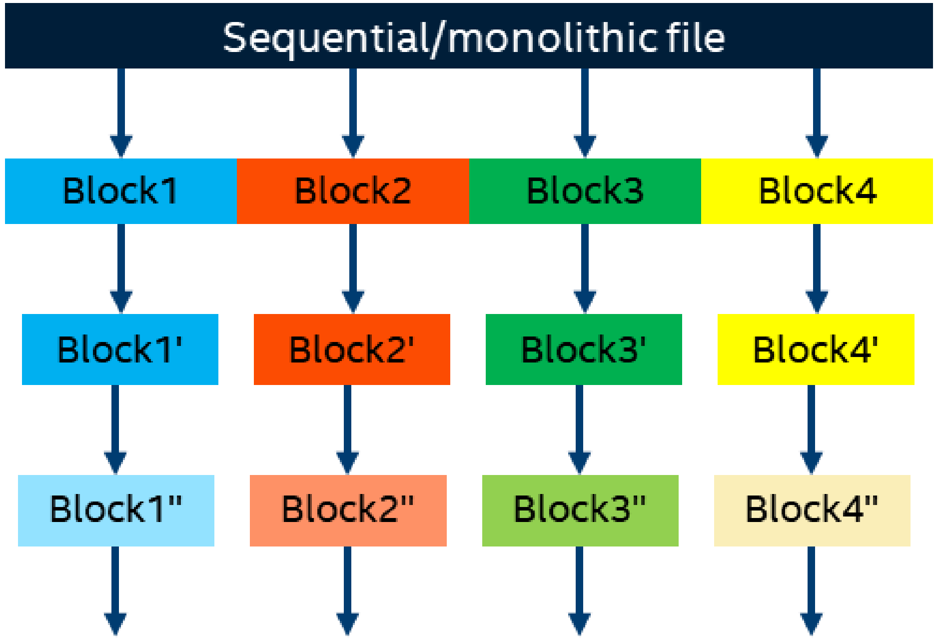
```python
import pandas as pd
import hpat

@hpat.jit
def read_pq():
    df = pd.read_parquet('cycling_dataset.pq')
    ...
    return result
```



DATA PARALLEL OPERATIONS

Data parallel operations (like filters, operations on individual rows) require no communication.

```
import pandas as pd
import hpat

@hpat.jit
def read_pq():
    df = pd.read_parquet('cycling_dataset.pq')

    df = df[df.power!=0]

    df['hr'] = df['hr'] * 2
    …
```

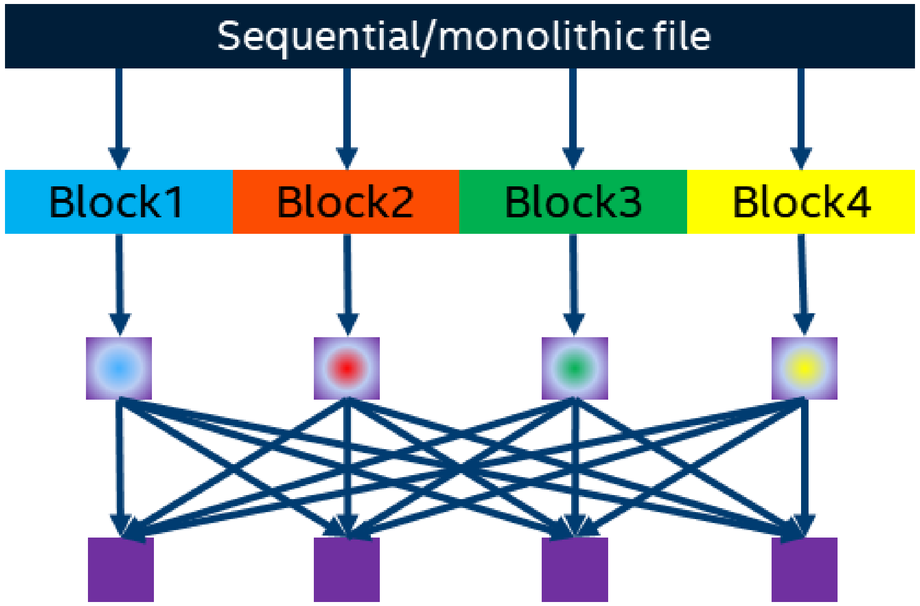


PARALLEL REDUCTION

Reductions (like mean, avg etc) are transformed to efficient MPI code as known from HPC.

Results from reductions get replicated on all processes

```
@hpat.jit
def read_pq():
    df = pd.read_parquet('cycling_dataset.pq')


    result = df.hr.mean()


    ...
```

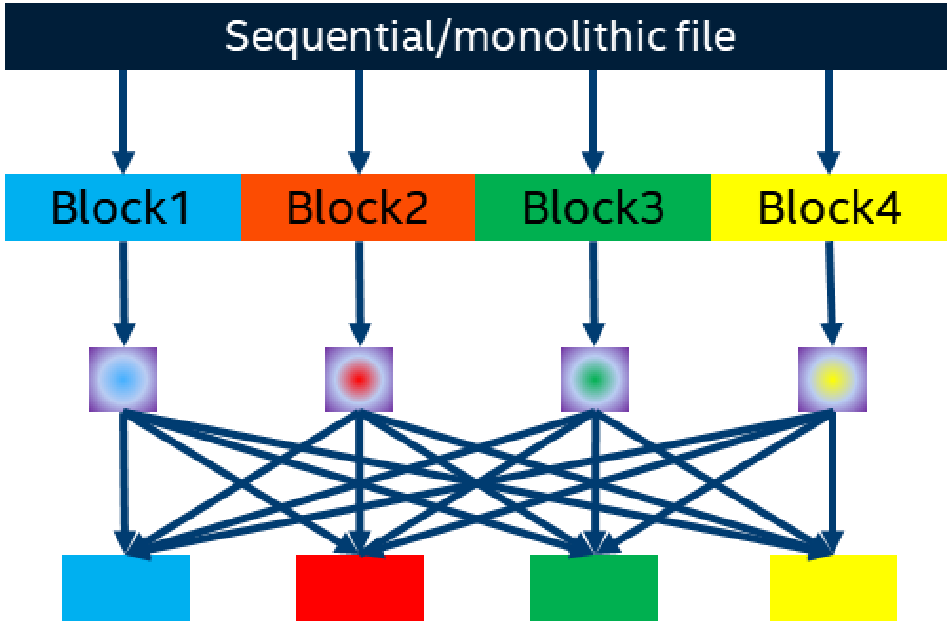


PARALLEL GROUPBY+AGGREGATION

Potentially more complex communication than simple reductions.

Result will be block distributed (potentially with variable block sizes)

```
@hpat.jit
def read_pq():
    df = pd.read_parquet('cycling_dataset.pq')

    ...
    grp = df.groupby('hour')
    mean = grp['power'].mean()

    ...
```

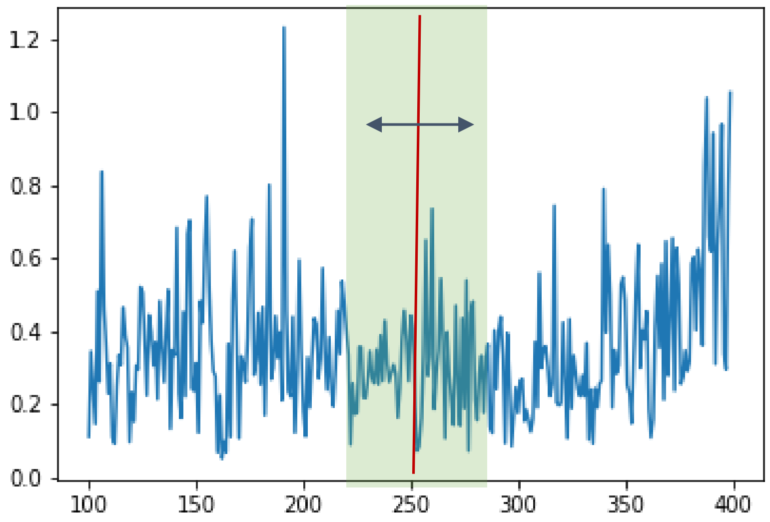


TIME SERIES ANALYTICS

Time series data naturally produced from many sources (video, IoT, finance, …)

- Key underlying problem: handling parallel algorithms with fine-grained communication
  - HPAT maps high-level semantics to MPI asynchronous primitives
- Example: ‘window’ functions

```
df.rolling('5min’, on='time')['pid'].apply(
        lambda a: pd.Series(a).nunique())
```

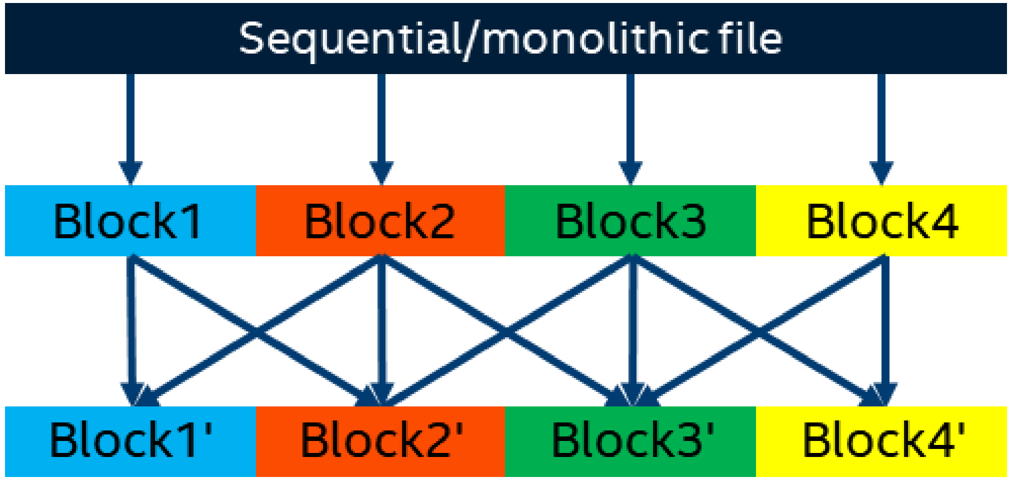


Communication across data partitions

PARALLEL ROLLING (WINDOWS)

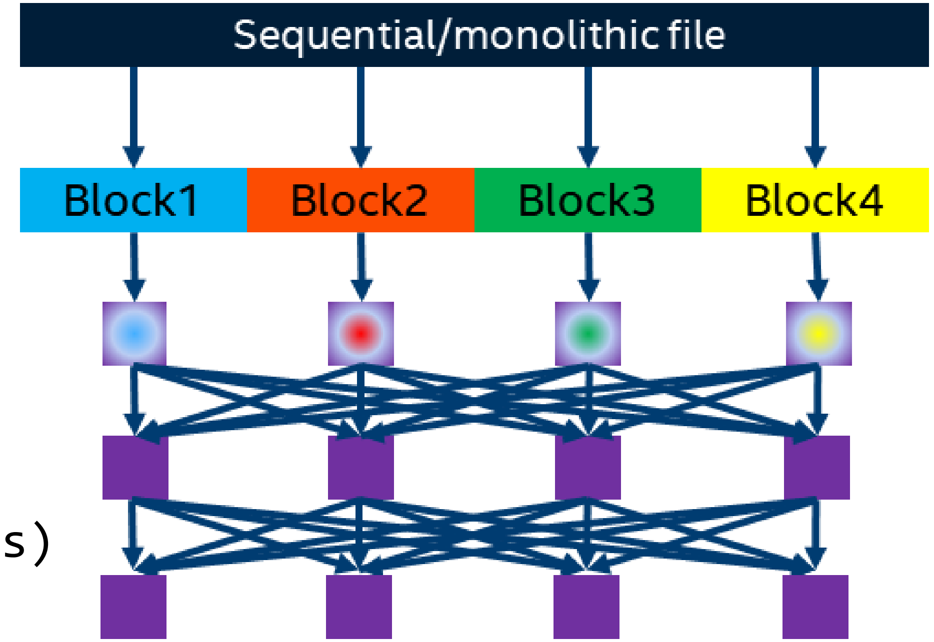
Requires neighbor communication only

```
@hpat.jit
def read_pq():
    df = pd.read_parquet('cycling_dataset.pq')

    ...
    mv_av = df.hr.rolling(4).mean()

    ...
```



MACHINE LEARNING WITH DAAL4PY

```python
import daal4py as d4p
import daal4py.hpat
import pandas as pd

# get inertia for various numbers of clusters
@hpat.jit
def find_clusters():
    X = pd.read_parquet(…).values
    distorsions = []
    for k in range(2, 20):
        kmi = d4p.kmeans_init(k)
        icenters = kmi.compute(X).centroids
        result = d4p.kmeans(k, 300).compute(X, icenters)
        distorsions.append(result.goalFunction[0][0])
    return distorsions
```



HPAT'S SCOPE OF FUNCTIONALITY

Operations	• Python/Numpy/Pandas 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 Extend Numba to support
Interoperability	• I/O integration (CSV, Parquet, HDF5) • Daal4py

HPAT LIMITATION: TYPE STABILITY

Input code to HPAT should be statically compilable (type stable)

- Dynamically typed code examples (rare in analytics):

Untypable variable:	Unresolvable function:	Nonstatic dataframe schema:

if flag1:
    a = 2
else:
    a = np.ones(n)
if isinstance(a, np.ndarray):
    doWork(a)
	if flag2:
	    f = np.zeros
	else:
	    f = np.ones
	b = f(m)
		if flag2:
		    df = pd.DataFrame({‘A’: [1,2,3]})
		else:
		    df = pd.DataFrame({‘A’: ['a', ‘b'. ‘c']})
		
		b = f(m)

PANDAS EXAMPLE (DATA PARALLEL)

115x speedup
on 4 nodes*

python™

```
@hpat.jit
def func():
    table = pd.read_parquet('data.parquet')
    data = table[table['A'].str.contains('ABC*', regex=True)]
    stats = data['B'].describe()
    print(stats)
```

```
$ mpirun -n 112 python ./process_times.py
```

Mean, std, min, max, 25/50/75% quantiles, count

*100M samples, 2U Intel(R) Xeon(R) Platinum 8180 nodes

PANDAS EXAMPLE (LOOP PARALLEL)

```
@hpat.jit(locals={'s_open': hpat.float64[:], …})
def intraday_mean_revert():
  f = h5py.File("stock_data.hdf5", "r"); …
  for i in prange(nsyms):
    symbol = sym_list[i]
    s_open = f[symbol+'/Open'][:]; …
    df = pd.DataFrame({'Open': s_open, …})
    df['Stdev'] = df['Close'].rolling(window=90).std()
    df['Moving Average'] = df['Close'].rolling(window=20).mean()
    df['Criteria1'] = (df['Open'] - df['Low'].shift(1)) < -df['Stdev']
    df['Criteria2'] = df['Open'] > df['Moving Average']
    df['BUY'] = df['Criteria1'] & df['Criteria2']
    df['Pct Change'] = (df['Close'] - df['Open']) / df['Open']
    df['Rets'] = df['Pct Change'][df['BUY'] == True]
    n_days = len(df['Rets'])
    res = np.zeros(max_num_days)
    if n_days:
      res[-n_days:] = df['Rets'].fillna(0)
    all_res += res
```

Explicit loop parallelism

100x speedup on 36 cores

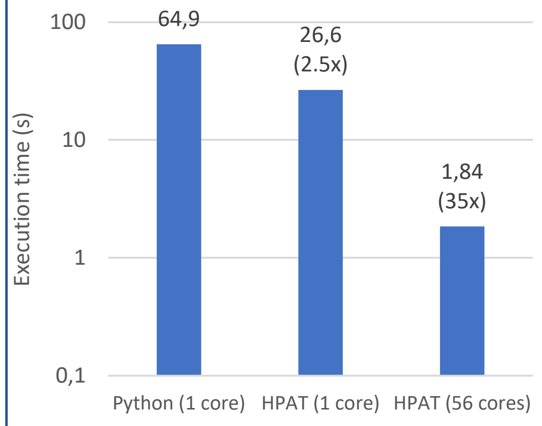
Intel Xeon E5-2699 v3 nodes

EARLY RESULTS

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.

Financial Exchange

Challenge: Ability to track several transaction statistics in real time

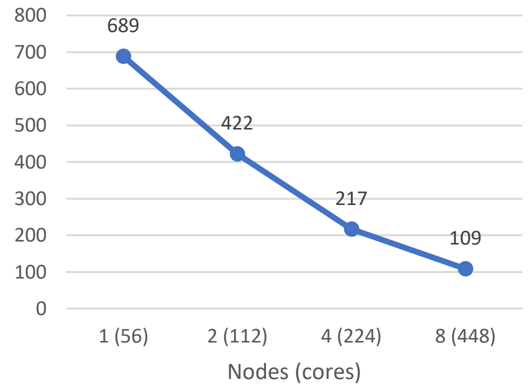


Intel Xeon E5-2699 v3 nodes

Autonomous Cars

Challenge: scale to massive time series data which need “window” computation.

Requires fine-grained comms not available in in Spark

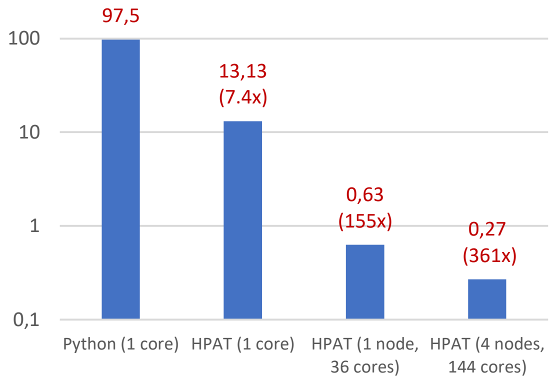


Intel(R) Xeon(R) Platinum 8180 nodes

Finance ISV

Challenge: User-defined compute kernels in Python

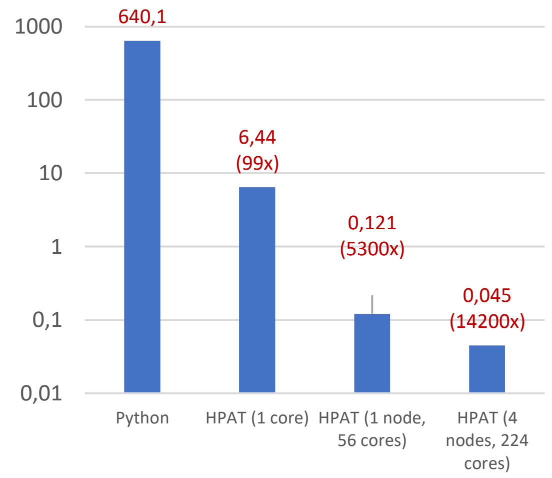
Spark can't improve user-defined code & infrastructure too complex for the target environment



Intel(R) Xeon(R) Platinum 8180 nodes

Telco

Challenge: user-defined functions for manipulating complex date/time data structures, not available in Spark



Intel(R) Xeon(R) Platinum 8180 nodes

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 Now in numba
Interoperability	• I/O integration (CSV, Parquet, HDF5, Xenon) • Daal4py integration

SCALABLE PYTHON SOLUTIONS IN INCUBATION

HPAT

Drop-in acceleration of Python analytics
(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 multi-node with MPI

https://github.com/IntelPython/hpat

https://intelpython.github.io/daal4py

Intel® Distribution for Python*

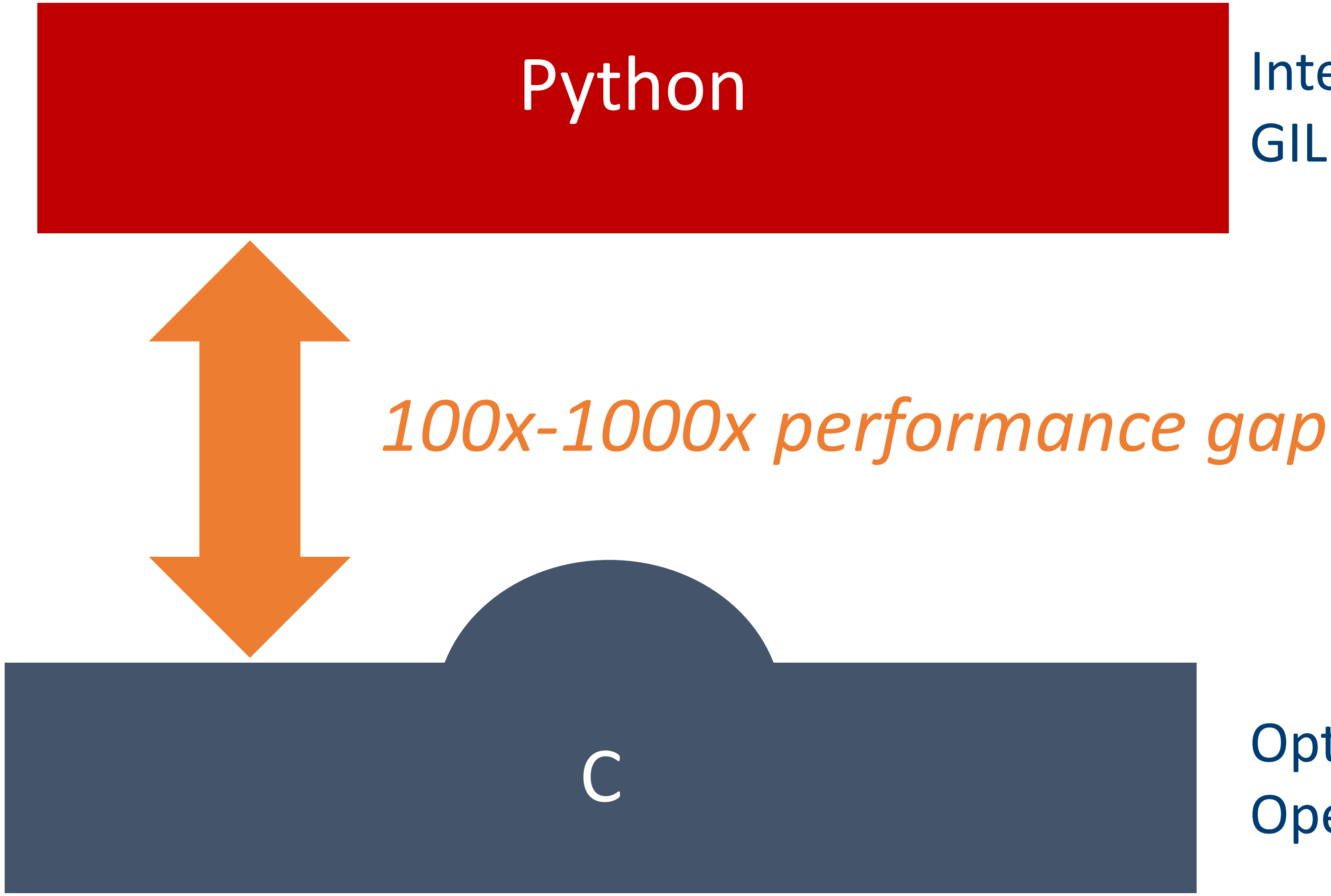
https://anaconda.org/intel
https://software.intel.com/en-us/distribution-for-python
https://intelpython.github.io/daal4py
https://github.com/IntelPython/hpat

Questions?

Tutorial: https://github.com/IntelPython/hpat/tree/tut2/tutorial

Docker container: intelpython/hpattut-test:cern

PERFORMANCE OF PYTHON



Interpreter
GIL

Optimizing compiler
OpenMP*/TBB/pthreads

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/

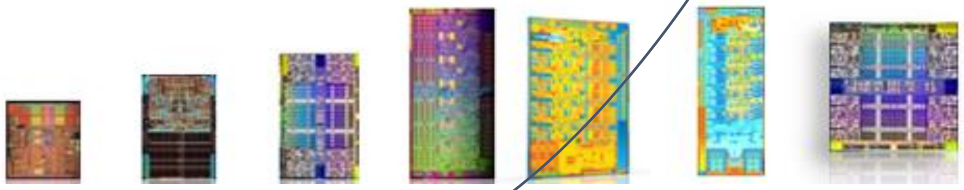
HIGH PERFORMANCE PYTHON

Python Libraries

Thin layer in Python or Cython

Intel® Performance Libraries

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

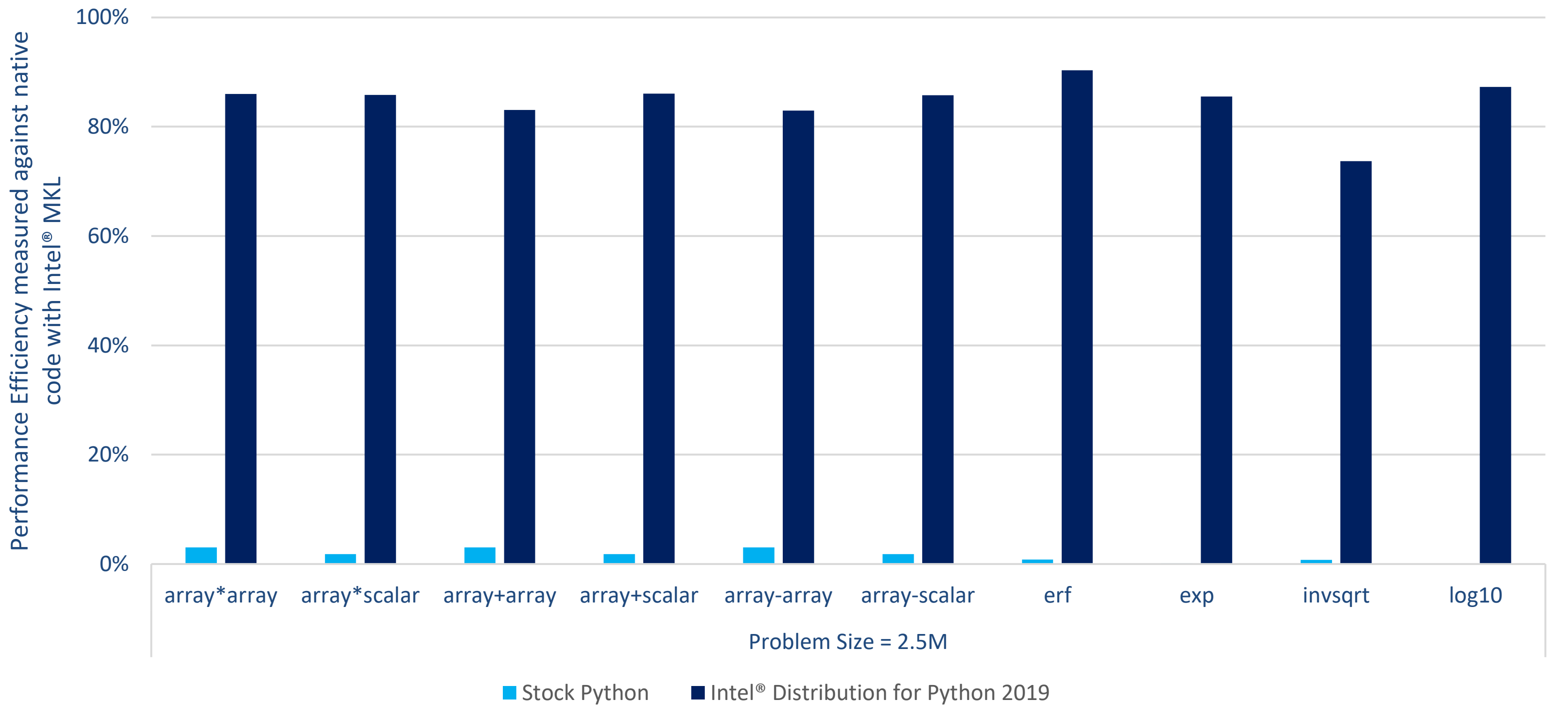


(generations of processors)

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

CLOSE TO NATIVE CODE UMATH PERFORMANCE WITH INTEL PYTHON 2019

COMPARED TO STOCK PYTHON PACKAGES ON INTEL® XEON PROCESSORS



87%

native efficiency on a full ***Black-Scholes*** *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 ARCHITECTURE