THE PAINLESS ROUTE IN PYTHON TO FAST AND SCALABLE MACHINE LEARNING

Victoriya Fedotova, Frank Schlimbach

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 analyzes only a small portion of data that they think has the most potential of bringing the great insights. This means they may miss out on valuable insights from the remaining bigger portion of the data — insights that may be crucial for the business.

Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

PRODUCTIVITY WITH PERFORMANCE VIA INTEL® PYTHON*

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

Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

TWO INGREDIENTS TO GET CLOSE-TO-NATIVE PERFORMANCE IN PYTHON

Pure Python	Python + Libraries	Libraries + JIT	C++
Serial Interpreted	Partially Ninja-level Partially Interpreted	Largely Ninja-level 100% native	100% Ninja-level 100% native

Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

DATA ANALYSIS AND MACHINE LEARNING

Data Input → Data Preprocessing → Model Creation → Prediction

Data Input, Data Preprocessing:

Pandas
Spark
SDC

Model Creation, Prediction:

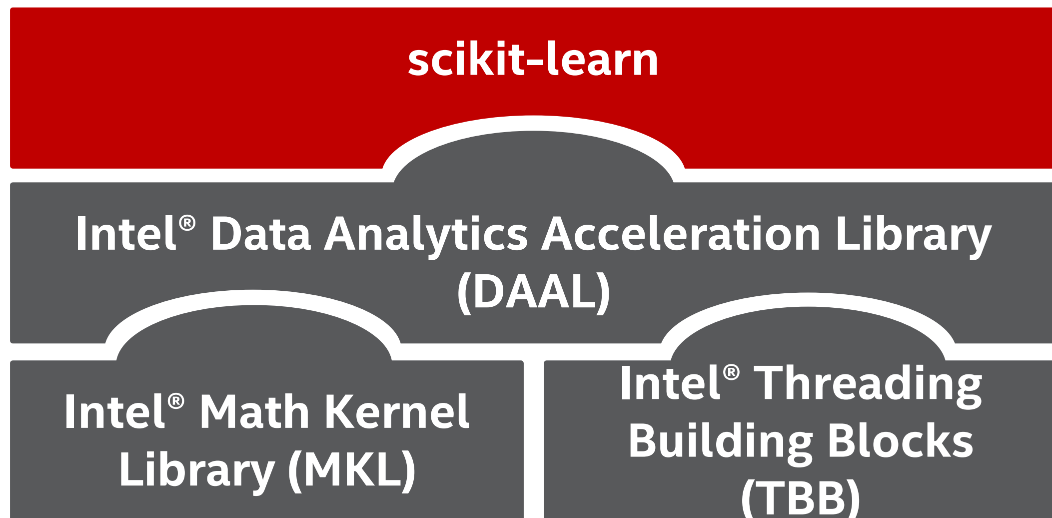
Scikit-learn
Spark
DL frameworks
daal4py

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

Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

INTEL® DATA ANALYTICS ACCELERATION LIBRARY (DAAL)

ACCELERATING MACHINE LEARNING



- ➢Efficient memory layout
- ➢Chunking for optimal cache performance
- ➢Computations mapped to most efficient matrix operations (in MKL)
- ➢Parallelization via TBB
- ➢Vectorization

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

Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

CLOSE TO NATIVE SCIKIT-LEARN PERFORMANCE WITH INTEL PYTHON 2020

COMPARED TO STOCK PYTHON PACKAGES ON INTEL® XEON PROCESSORS

Performance efficiency measured against native code with Intel® DAAL

120%
100%
80%
60%
40%
20%
0%

Correlation, Cosine, KMeans.fit, KMeans.predict, Linear.fit, Linear.predict, LogisticRegression.fit, LogisticRegression.predict, PCA.fit, PCA.transform, Ridge.fit, Ridge.predict, SVM.fit, SVM.predict, RandomForestClassifier.fit, RandomForestClassifier.predict, RandomForestRegressor.fit, RandomForestRegressor.predict

■ Intel® Distribution for Python* 2020 ■ Stock Python

Configuration: Testing by Intel as of **November 27, 2019**. Stock Python: Python 3.7.5 h_0371630_0 installed from conda, numpy 1.17.4, numba 0.46.0, llvmlite 0.30.0, scipy 1.3.2, scikit-learn 0.21.3 installed from pip; Intel Python: Intel® Distribution for Python* 2020 Gold: Python 3.7.4 hf484d3e_3, numpy 1.17.3 py37ha68da19_4, mkl 2020 intel_133, mkl_fft 1.0.15 py37ha68da19_3, mkl_random 1.1.0 py37ha68da19_0, numba 0.45.1 np117py37_1, llvmlite 0.29.0 py37hf484d3e_9, scipy 1.3.1 py37ha68da19_2, scikit-learn 0.21.3 py37ha68da19_14, daal 2020 intel_133, daal4py 2020 py37ha68da19_4; Cent OS Linux 7.3.1611, kernel 3.10.0-514.el7.x86_64; Hardware: Intel Xeon® Platinum® 8280 CPU @ 2.70 GHz (2 sockets, 28 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 https://www.intel.com/benchmarks

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

Optimization Notice

Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

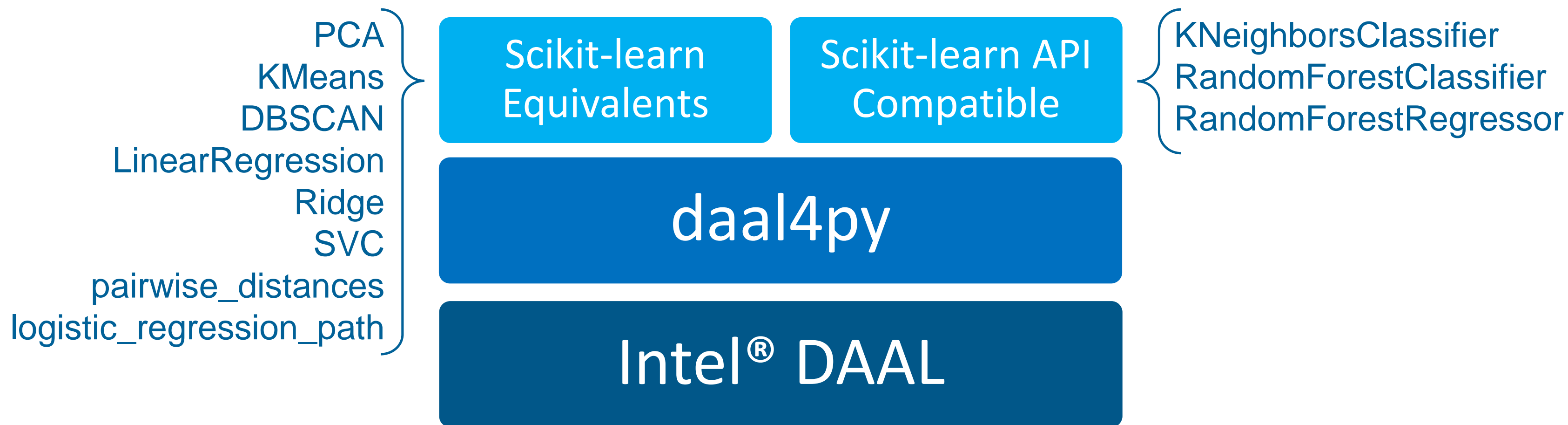
ACCELERATING SCIKIT-LEARN THROUGH DAAL4PY

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

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

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

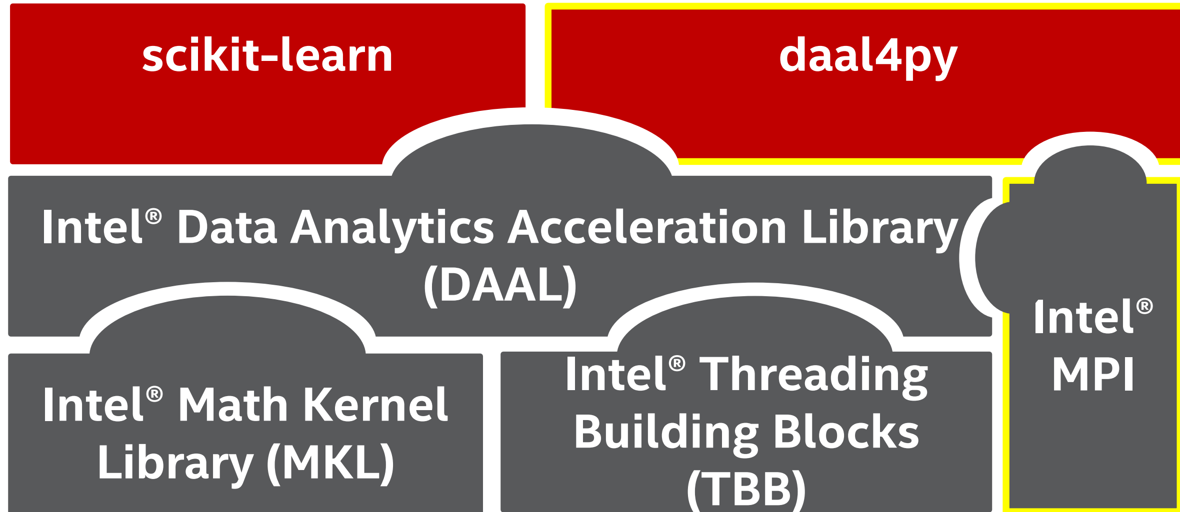
Monkey-patch any scikit-learn programmatically



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

Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

SCALING MACHINE LEARNING BEYOND A SINGLE NODE



Simple Python API
Powers Scikit-learn

Powered by DAAL

Scalable to multiple nodes

Open source

Try it out! `conda install -c intel daal4py`

Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

K-MEANS USING SCIKIT-LEARN AND DAAL4PY

Scikit-learn

```
from sklearn.cluster import KMeans
import pandas as pd

data = pd.read_csv("./kmeans.csv")

algo = KMeans(n_clusters=20,
  init='k-means++', max_iter=5)

result = algo.fit(data)

result.labels_
result.cluster_centers_
```

daal4py

```
from daal4py import kmeans_init, kmeans
import pandas as pd

data = pd.read_csv("./kmeans.csv")          # Load the data

init = kmeans_init(nClusters=20,             # Compute initial
  method="plusPlusDense").compute(data)      # centroids

algo = kmeans(nClusters=20,                  # Configure K-means
  maxIterations=5, assignFlag=True)          # main object

result = algo.compute(data,                  # Compute the
                      init.centroids)        # clusters and labels

result.assignments                           # Print the results
result.centroids
```

Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

DISTRIBUTED K-MEANS USING DAAL4PY

```
from daal4py import kmeans_init, kmeans, daalinit, daalfini, my_procid
import pandas as pd

# Optionally initialize distributed execution environment
daalinit()

# Load the data. Daal4py accepts data as CSV files, numpy arrays or pandas dataframes
data = pd.read_csv("./kmeans_dense_{}.csv".format(my_procid() + 1))

# compute initial centroids
init_res = kmeans_init(nClusters=10, method="plusPlusDense", distributed=True).compute(data)

# configure kmeans main object: we also request the cluster assignments
algo = kmeans(nClusters=10, maxIterations=25, distributed=True)

# compute the clusters/centroids
result = algo.compute(data, init_res.centroids)

daalfini()
```

```
mpirun -n 4 python kmeans_distributed.py
```


Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

DAAL4PY API GENERATION

```
#include <mpi.h>
#include "daal.h"
#include "service.h"

using namespace std;
using namespace daal;
using namespace daal::algorithm

typedef float algorithmFPType;

/* Input data set parameters */
const size_t nBlocks = 4;
size_t nFeatures;

int rankId, comm_size;
#define mpi_root 0

const string fileNames[] = { "./pca_1.csv", "./pca_2.csv", "./pca_3.csv", "./pca_4.csv" };

int main(int argc, char * argv[])
{
    checkArguments(argc, argv, 4, &fileNames[0], &fileNames[1], &fileNames[2], &fileNames[3]);

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rankId);

    /* Initialize FileDataSource<CSVFeatureManager> to retrieve the input data from a .csv file */
    FileDataSource<CSVFeatureManager> dataSource(datasetFileNames[rankId], DataSource::doAllocateNumericTable,
aSource::doDictionaryFromContext);

    /* Retrieve the input data
    dataSource.loadDataBlock()

    /* Create an algorithm for
    pca::Distributed<step1Local, algorithmFPType, pca::svdDense> localAlgorithm;

    /* Set the input data set to the algorithm */
    localAlgorithm.input.set(pca::data, dataSource.getNumericTable());

    /* Compute PCA decomposition */
    localAlgorithm.compute();
```

```
    /* Serialize partial results required by step 2 */
    services::SharedPtr<byte> serializedData;
    InputDataArchive dataArch;
    localAlgorithm.getPartialResult()->serialize(dataArch);
    size_t perNodeArchLength = dataArch.getSizeOfArchive();

    /* Serialized data is of equal size on each node if each node called compute() equal number of times */

        serializedData = services::SharedPtr<byte>(new byte[perNodeArchLength * nBlocks]);

    dataArch.copyArchiveToArray(nodeResults, perNodeArchLength);

, MPI_CHAR, mpi_root, MPI_COMM_WORLD);

    delete[] nodeResults;

    if (rankId == mpi_root)
    {
        /* Create an algorithm for principal component analysis using the SVD method on the master node */
        pca::Distributed<step2Master, algorithmFPType, pca::svdDense> masterAlgorithm;

        for (size_t i = 0; i < nBlocks; i++)
        {
            /* Deserialize partial results from step 1 */
            OutputDataArchive dataArch(serializedData.get() + perNodeArchLength * i, perNodeArchLength);

            services::SharedPtr<pca::PartialResult<pca::svdDense> > dataForStep2FromStep1 =
                services::SharedPtr<pca::PartialResult<pca::svdDense> >(new pca::PartialResult<pca::svdDense>());
            dataForStep2FromStep1->deserialize(dataArch);

            /* Set local partial results as input for the master-node algorithm */
            masterAlgorithm.input.add(pca::partialResults, dataForStep2FromStep1);
        }

        /* Retrieve the algorithm results */
        pca::ResultPtr result = masterAlgorithm.getResult();

        /* Print the results */
        printNumericTable(result->get(pca::eigenvalues), "Eigenvalues:");
        printNumericTable(result->get(pca::eigenvectors), "Eigenvectors:");
    }

    MPI_Finalize();

    return 0;
}
```

Semi-automatic API generation process:

- Parse C++ headers to generate Cython code
- Use jinja2 to generate Python classes for algorithms, models, results, etc

100X fewer LOC for multi-node algorithms

Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

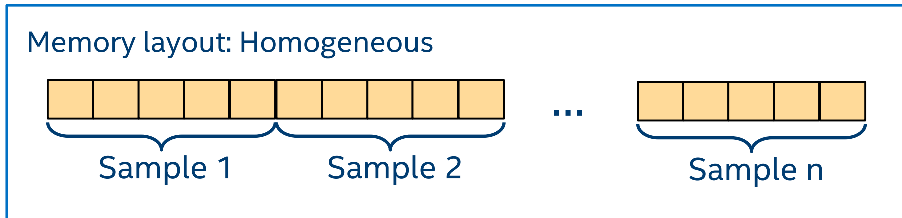
intel

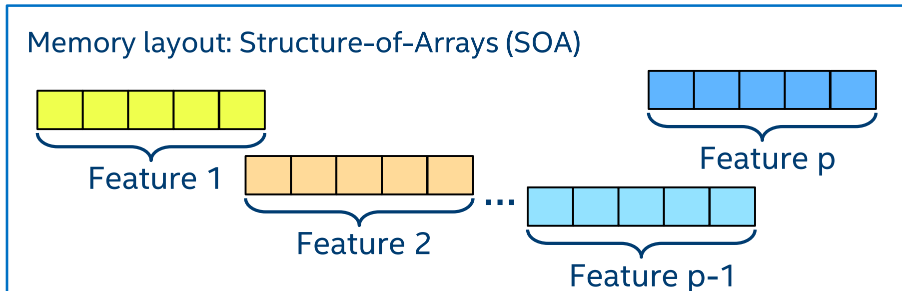
EFFECTIVE DATA TRANSFER: PYTHON ↔ NATIVE

Python data type

- numpy.ndarray
  - Homogeneous dense array
- pandas.DataFrame
  - Heterogeneous data

DAAL data type

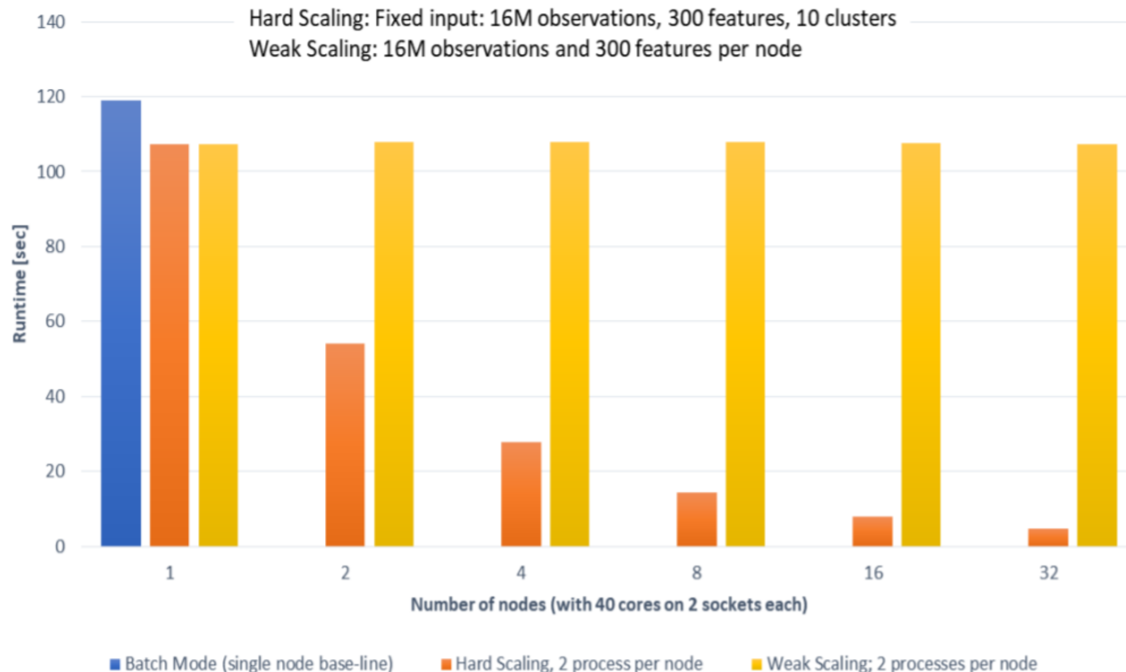




daal4py mostly avoids data copies and works optimally with various data layouts

Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

STRONG AND WEAK SCALING VIA DAAL4PY



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.

Configuration: Intel® Xeon® Gold 6148 CPU @ 2.40GHz, EIST/Turbo on 2 sockets, 20 cores per socket, 192 GB RAM, 16 nodes connected with Infiniband, Oracle Linux Server release 7.4, using 64-bit floating point numbers

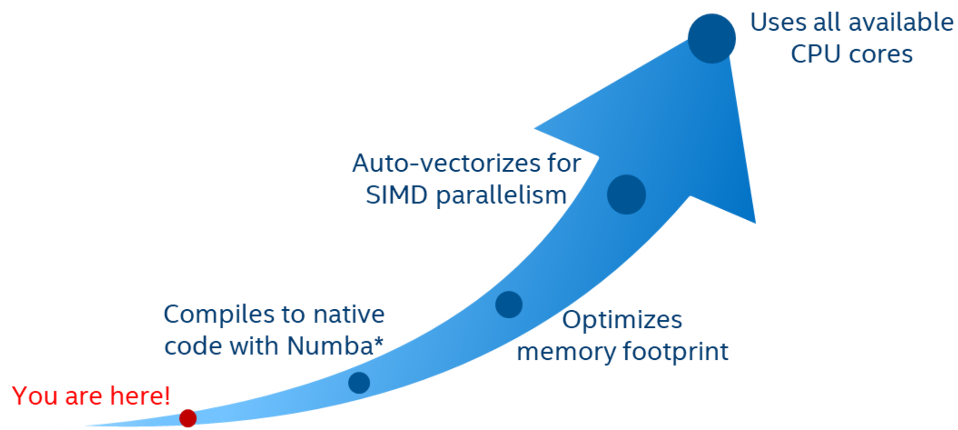
Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel® Scalable Dataframe Compiler (SDC)

INTEL® SCALABLE DATAFRAME COMPILER (SDC)

EVOLVED FROM HIGH-PERFORMANCE ANALYTICS TOOLKIT (HPAT)

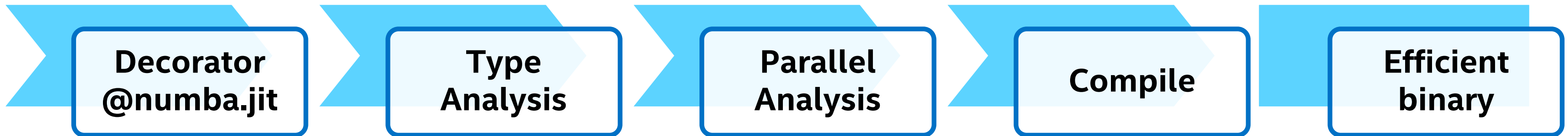
It's a compiler
A just-in-time compiler




```python
import pandas as pd
from numba import njit

# Dataset for analysis
FNAME = "employees.csv"


# This function gets compiled by Numba*
@njit
def get_analyzed_data():
    df = pd.read_csv(FNAME)
    s_bonus = pd.Series(df['Bonus %'])
    s_first_name = pd.Series(df['First Name'])
    m = s_bonus.mean()
    names = s_first_name.sort_values()
    return m, names


# Printing names and their average bonus percent
mean_bonus, sorted_first_names = get_analyzed_data()
print(sorted_first_names)
print('Average Bonus %:', mean_bonus)
```


Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

COMPILATION PIPELINE (HIGH-LEVEL VIEW)




```
@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,
```

Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

BASIC WORKFLOW EXAMPLE

```
import numpy as np
import pandas as pd
from numba import njit

# This function gets compiled by Numba*
@njit
def get_analyzed_data(file_name):
    df = pd.read_csv(file_name,
                     dtype={'Bonus %': np.float64, 'First Name': str},
                     usecols=['Bonus %', 'First Name’])
    s_bonus = pd.Series(df['Bonus %’])
    s_first_name = pd.Series(df['First Name’])
    m = s_bonus.mean()
    names = s_first_name.sort_values()
    return m, names

mean_bonus, sorted_first_names = get_analyzed_data(‘employees.csv’)
print(sorted_first_names)
print('Average Bonus %:', mean_bonus)
```


Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

INTEL® SDC AND NUMBA* LIMITATION: TYPE STABILITY

Input code to SDC must be statically compilable (type stable)

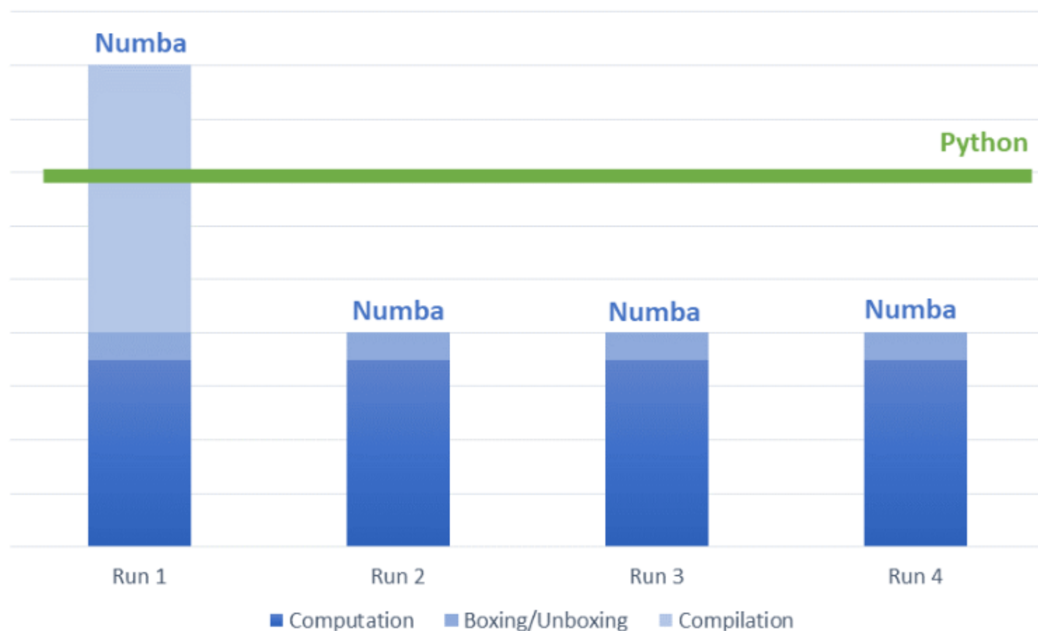
- Dynamically typed code examples (rare in analytics):

Untypable variable	Unresolvable function	Nonstatic DataFrame schema

<pre>if flag1:
  a = 2
else:
  a = np.ones(n)
 	
if isinstance(a,
    np.ndarray):
  doWork(a)</pre>
	<pre>if flag2:
	  f = np.zeros
	else:
	  f = np.ones
	
	b = f(m)</pre>
		<pre>if flag3:
		  df = pd.DataFrame({'A':
		    [1,2,3]})
		else:
		  df = pd.DataFrame({'A':
		    ['a', 'b', 'c']})
		b = f(m)</pre>

Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

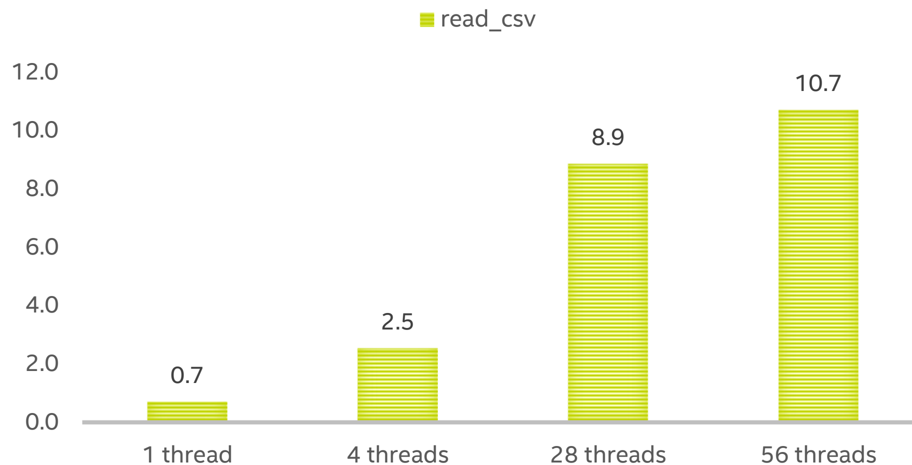
GETTING PERFORMANCE WITH INTEL® SDC



- Compile parts of code where parallelism resides
- Compile functions that are called multiple times
- Minimize number of columns in dataframes in the regions being compiled

Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

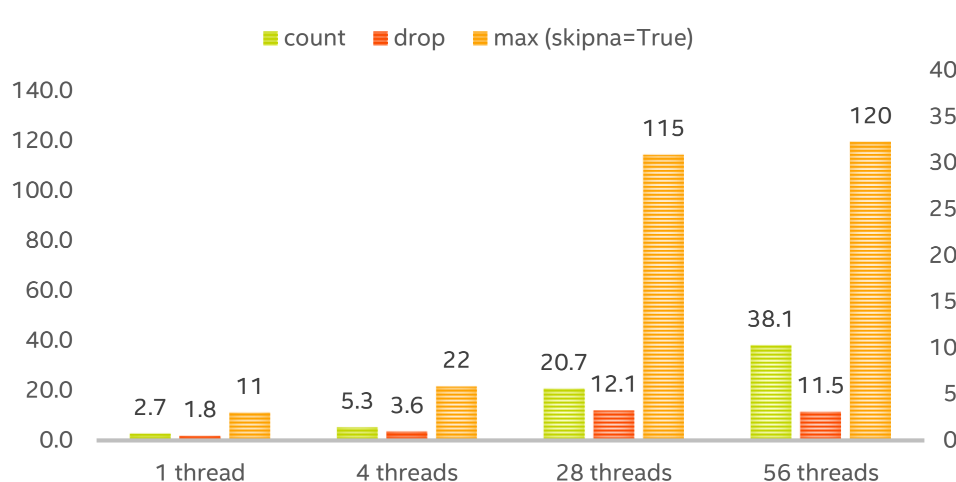
Intel SDC Performance – read_csv

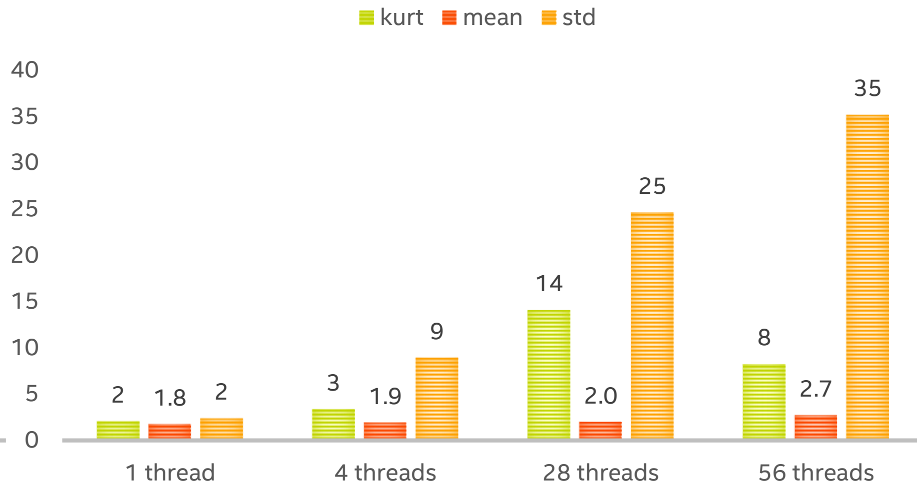


Intel® SDC Beta, Numba* 0.48, Pandas* 0.25.3
Intel® Xeon™ Platinum 8280L, 2.7 GHz, 2x28 cores, Hyperthreading=on, Turbo Mode=on

Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel SDC Performance – Dataframes

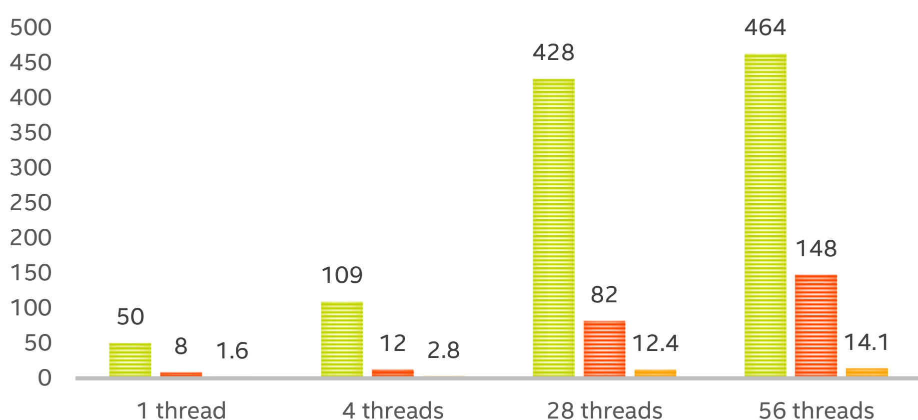


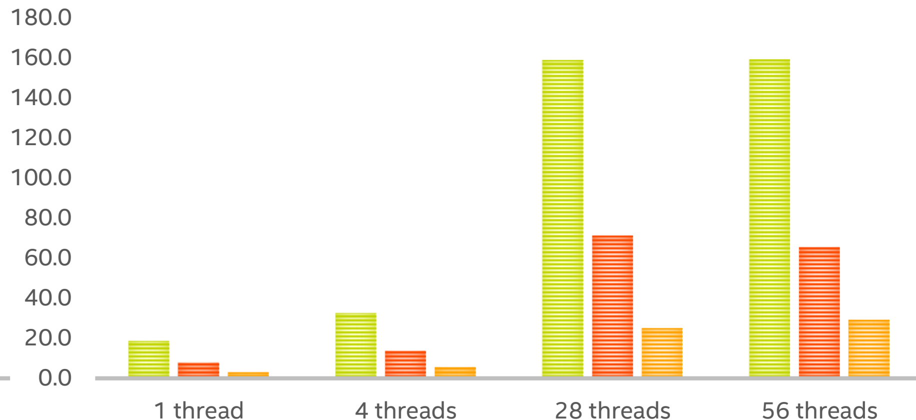


Intel® SDC Beta, Numba* 0.48, Pandas* 0.25.3
Intel® Xeon™ Platinum 8280L, 2.7 GHz, 2x28 cores, Hyperthreading=on, Turbo Mode=on

Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel SDC Performance – Series





Intel® SDC Beta, Numba* 0.48, Pandas* 0.25.3
Intel® Xeon™ Platinum 8280L, 2.7 GHz, 2x28 cores, Hyperthreading=on, Turbo Mode=on

Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

INTEL® SDC FUNCTIONALITY

Coming soon: time series and categoricals

Operations	• Python/Numpy/Pandas* basics • Statistical operations (max, std, median, …) • Relational operations (filter, groupby) • Rolling window (rolling)
Data	• Missing value • Dates • ASCII/Unicode strings • Data-Frames, Series, Lists, Dictionaries, Tuples
Interoperability	• I/O integration (CSV)

Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

INTEL® SCALABLE DATAFRAME COMPILER (SDC)

EVOLVED FROM HIGH-PERFORMANCE ANALYTICS TOOLKIT (HPAT)

Open source project

- https://github.com/IntelPython/sdc
- https://intelpython.github.io/sdc-doc/latest/index.html

In Beta till end of 2020

Available as conda packages and pip wheels (Python 3.6/3.7, Windows/Linux)

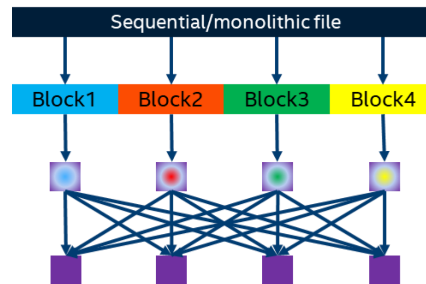
- `conda install -c intel/label/beta sdc`
- `pip install -i https://pypi.anaconda.org/intel/label/beta/simple sdc`

Optimization Notice
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

WHAT'S NEXT?

More Pandas features

Auto-scale-out to clusters of workstations



Compiling to and running on GPUS/accelerators

Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

COMPILING TO AND RUNNING ON GPUS/ACCELERATORS

```
1  @dppy.kernel
2  def data_parallel_sum(a, b, c):
3      i = dppy.get_global_id(0)
4      c[i] = a[i] + b[i]
```

```
1  @numba.vectorize(nopython=True)
2  def cndf2(inp):
3      out = 0.5 + 0.5 * math.erf((math.sqrt(2.0)/2.0) * inp)
4      return out
5
6  @numba.njit(parallel=True, fastmath=True)
7  def blackscholes(sptprice, strike, rate, volatility, timev):
8      logterm = np.log(sptprice / strike)
9      powterm = 0.5 * volatility * volatility
10     den = volatility * np.sqrt(timev)
11     d1 = (((rate + powterm) * timev) + logterm) / den
12     d2 = d1 - den
13     NofXd1 = cndf2(d1)
14     NofXd2 = cndf2(d2)
15     futureValue = strike * np.exp(- rate * timev)
16     c1 = futureValue * NofXd2
17     call = sptprice * NofXd1 - c1
18     put  = call - futureValue + sptprice
19     return put
```

```
1  with device_context(gpu, 0):
2      black_scholes(SP, S, R, V, T)
```


Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

END-TO-END PERFORMANCE OF ANALYTICS

daal4py and SDC help accelerate, scale-up and scale-out the entire analytics process in Python from preprocessing through machine learning

- https://anaconda.org/intel
- https://software.intel.com/distribution-for-python
- https://intelpython.github.io/daal4py
- https://github.com/IntelPython/sdc
- https://medium.com/intel-analytics-software

Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

LEGAL DISCLAIMER & OPTIMIZATION NOTICE

Performance results are based on testing as of November 27, 2019, May 18, 2020 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 intel.com/benchmarks.

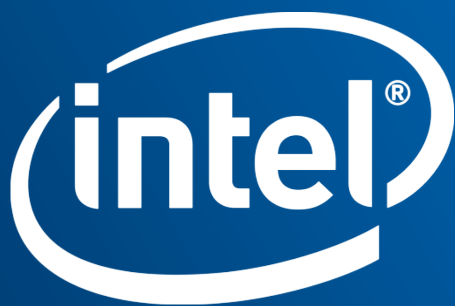
INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING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

Copyright © 2020, Intel Corporation. All rights reserved. Intel, Xeon, Core, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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

Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.


intel®