PERFORMING NUMERICAL ANALYSIS AND DATA ANALYTICS WITH PYTHON AT SCALE

Sergey Maidanov

Engineering Manager for Intel® Distribution for Python

Why Python?

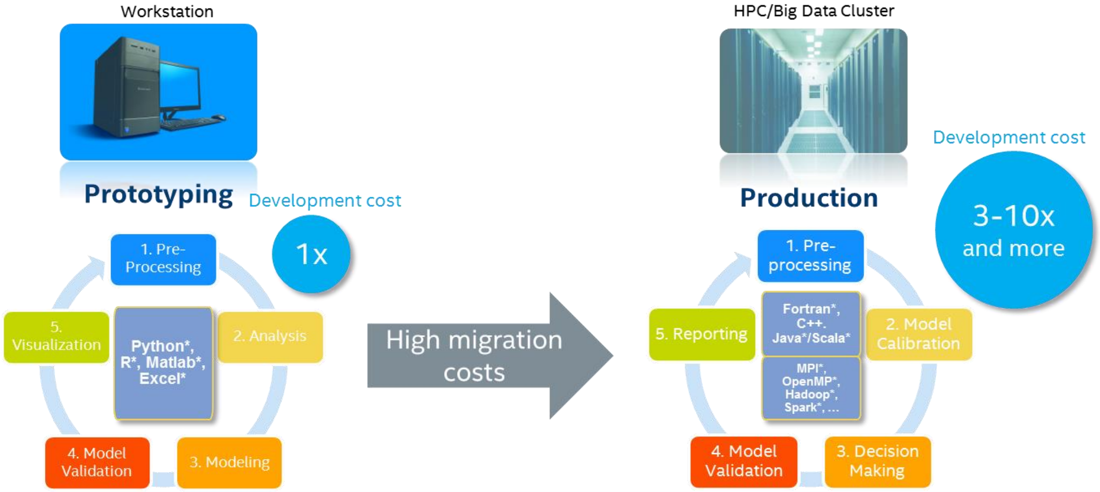
> “**Python wins the heart of developers** across all ages, according to our Love-Hate index. Python is also the most popular language that **developers want to learn** overall, and
> a **significant share already knows it**”
>
> 2018 Developer Skills Report

HackerRank

- Python, Java, R are top 3 languages in job postings for data science and machine learning jobs
  - https://www.kdnuggets.com/2017/01/most-popular-language-machine-learning-data-science.html

KDnuggets

Why scalability matters in (Data) Science



A TOAST for Next Generation CMB Experiments

Berkeley Lab Cosmology Software Scales Up to 658,784 Knights Landing Cores

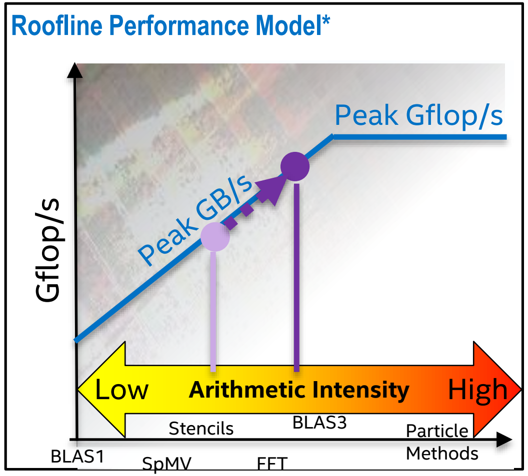
According to Kisner, the challenges to building a tool that can be used by the entire CMB community were both technical and sociological. Technically, the framework had to perform well at high concurrency on a variety of systems, including supercomputers, desktop workstations and laptops. It also had to be flexible enough to interface with different data formats and other software tools. Sociologically, parts of the framework that researchers interact with frequently had to be written in a high-level programming language that many scientists are familiar with.

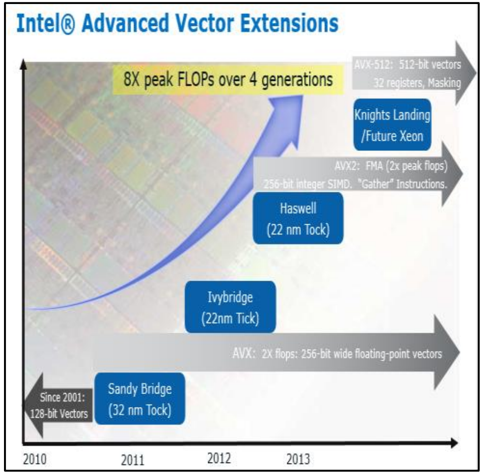
What scalability technically means

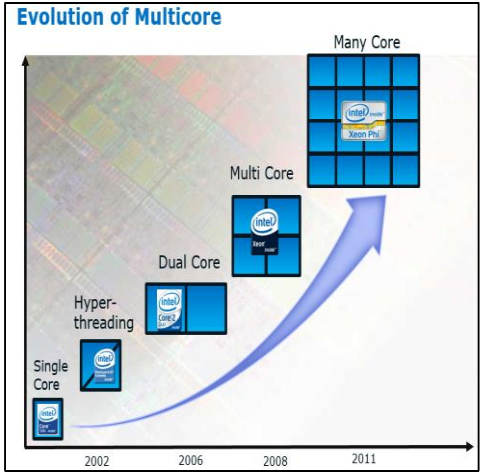
Hardware and software efficiency crucial in production (Perf/Watt, etc.)

Efficiency = Parallelism

- Instruction Level Parallelism with effective memory access patterns
- SIMD
- Multi-threading
- Multi-node







* Roofline Performance Model https://crd.lbl.gov/departments/computer-science/PAR/research/roofline/

Extracting parallelism in Python

- CPython as interpreter inhibits parallelism but…
- … Overall Python tools evolved far toward unlocking parallelism

Efficiency = Parallelism

Packages (numpy*, scipy*, scikit-learn*, etc.) accelerated with MKL, DAAL, IPP	Composable multi-threading with Intel® TBB, OpenMP*, and SMP packages	Multi-node parallelism with mpi4py* accelerated with Intel® MPI*
Language extensions for vectorization & multi-threading (Cython*, Numba*)	Integration with Big Data platforms and Machine Learning frameworks	Mixed language profiling with Intel® VTune™ Amplifier

Near to native Python efficiencies

Domain	Native	Python Efficiency
Linear Algebra (numpy/scipy)	MKL BLAS/LAPACK	91%
FFT (numpy/scipy)	MKL FFT	85%
Arithmetic & Transcendentals (numpy)	MKL VML, ICC SVML	92%
Numba (Black Scholes) - serial	ICC	92%
Numba (Black Scholes) - parallel	ICC	82%
Scikit-learn	DAAL	90%
RNG (numpy)	MKL RNG	90%

`PythonEfficiency=Python/BestNative*100%`. Geomean across representative workloads within domain.

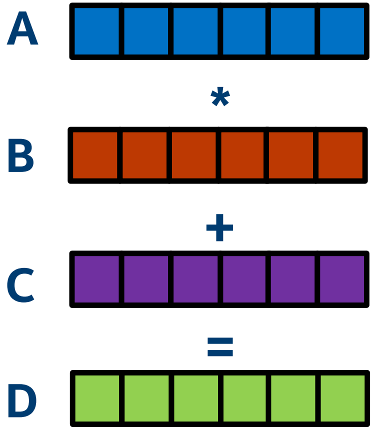
Linear algebra: dot, det, inv, lu; **FFT:** 1D, 2D, 3D (in-place and out-of-place); **Arithmetic & Transcendental:** +, -, *, erf, exp, invsqrt, log10; **Scikit-learn:** cosinedist, corrdist, kmeans (fit, predict), linearregr (fit, predict), ridgeregr (fit, predict), SVM (fit, predict); **RNG:** rand, randn, gamma, beta, randint, poisson, hypergeometric

ParallelAccelerator architecture for Numba*

ParallelAccelerator
Extract shared-memory parallelism

Numba*
Compile Python/Numpy

LLVM
Binary code generation




```
D = A * B + C
```

Recognize parallelism

```
parfor i=1:n
    t[i]=A[i]*B[i]
parfor i=1:n
    D[i]=t[i]+C[i]
```

Fuse loops

```
parfor i=1:n
    t[i]=A[i]*B[i]+C[i]
```

https://github.com/numba/numba

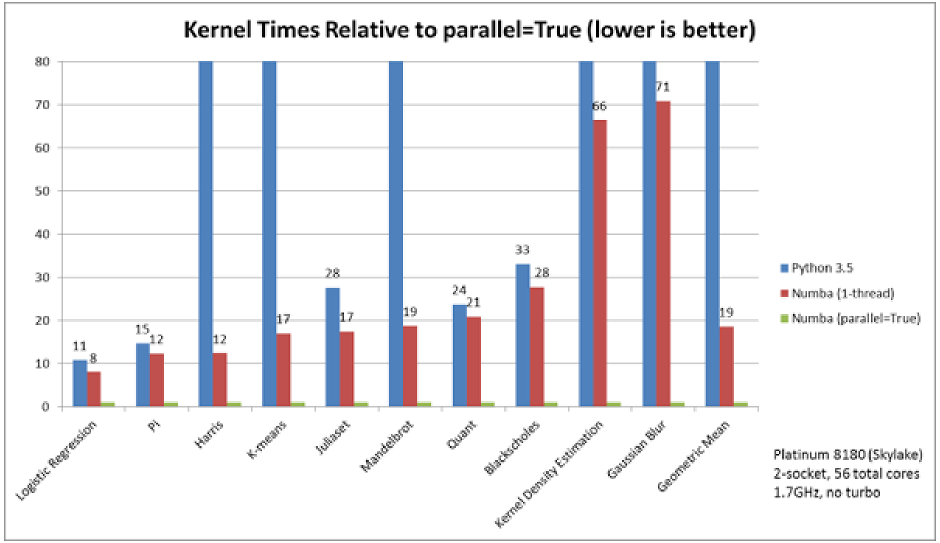
https://github.com/IntelLabs/parallelaccelerator.jl

ParallelAccelerator for Numba - Highlights

```
@numba.jit(nopython=True, parallel=True)
def logistic_regression(Y, X, w, iterations):
  for i in range(iterations):
    w -= np.dot(((1.0 / (1.0 + np.exp(-Y * np.dot(X, w))) - 1.0) * Y), X)
  return w
```

With ParallelAccelerator you can

- Basic math and comparisons
- NumPy ufuncs supported in `nopython` mode
- User-defined ufuncs created with `numba.vectorize`
- Reductions for sum and product
- Array creation `np.ones` and `np.zeros`
- Vector-vector and matrix-vector dot products



ParallelAccelerator
`prange()` and `numba.stencil`

```
@numba.jit(nopython=True, parallel=True)
def normalize(x):
  ret = np.empty_like(x)

  for i in numba.prange(x.shape[0]):
    acc = 0.0
    for j in range(x.shape[1]):
      acc += x[i,j]**2

    norm = np.sqrt(acc)
    for j in range(x.shape[1]):
      ret[i,j] = x[i,j] / norm

  return ret
```

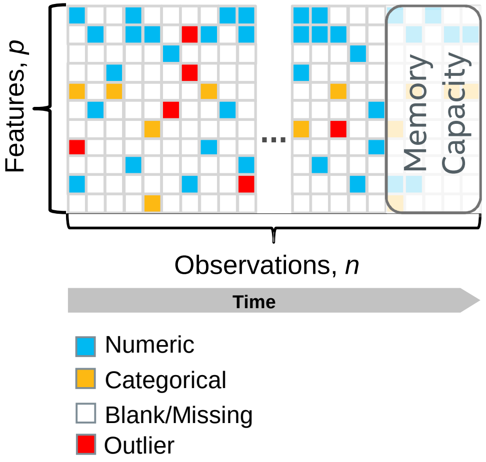
```
N = 10
GAMMA = 2.2

@numba.jit(nopython=True, parallel=True)
def blur(x):
  def stencil_kernel(a):
    acc = 0.0
    for i in range(-N, N+1):
      for j in range(-N, N+1):
        acc += a[i,j]**GAMMA

    avg = acc/((2*N+1)*(2*N+1))
    return np.uint8(avg**(1/GAMMA))

  return numba.stencil(stencil_kernel,
                       neighborhood=((-N,N),(-N,N)))(x)
```

Data Management for Big Data

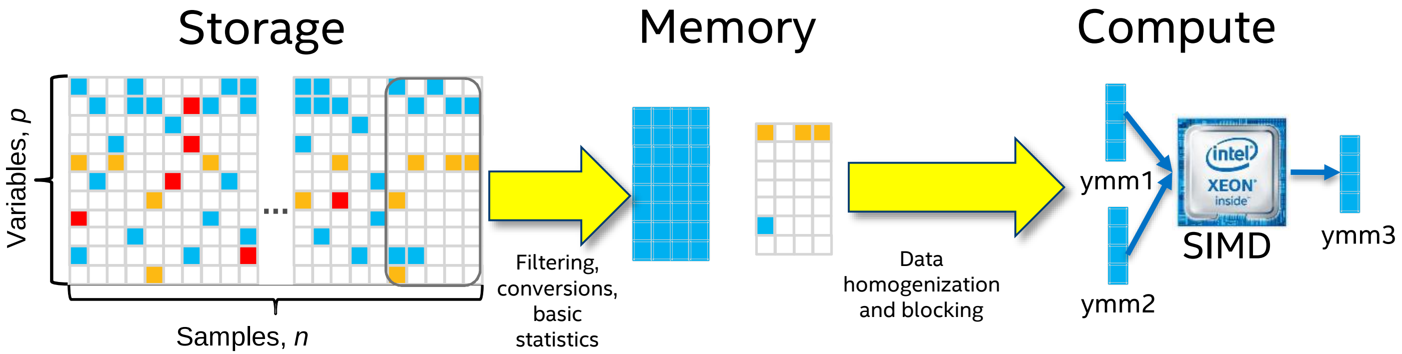


Big Data Attributes	Computational Solution
Distributed across different devices	•Distributed processing with communication-avoiding algorithms
Huge data size not fitting into device memory	•Distributed processing •Online algorithms
Data coming in time	•Data buffering & asynchronous computing •Online algorithms
Non-homogeneous data	•Categorical→Numeric (counters, histograms, etc) •Homogeneous numeric data kernels • Conversions, Indexing, Repacking
Sparse/Missing/Noisy data	•Sparse data algorithms •Recovery methods (bootstrapping, outlier correction)

Bridging Storage & Compute

Optimizing storage ≠ optimizing compute

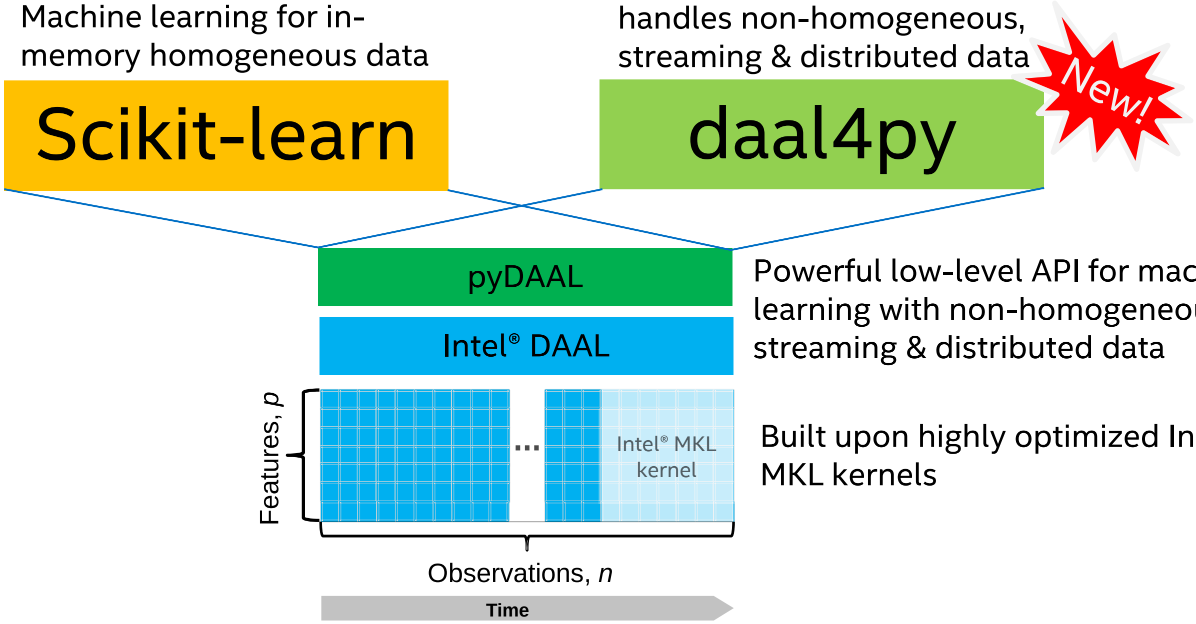
- Storage: efficient non-homogeneous data encoding for smaller footprint and faster retrieval
- Compute: efficient memory layout, homogeneous data, contiguous access
- Easier manageable for traditional HPC, much more challenging for Big Data



Scikit-learn, Intel® DAAL, pyDAAL, DAAL4Py

Machine learning for in-memory homogeneous data

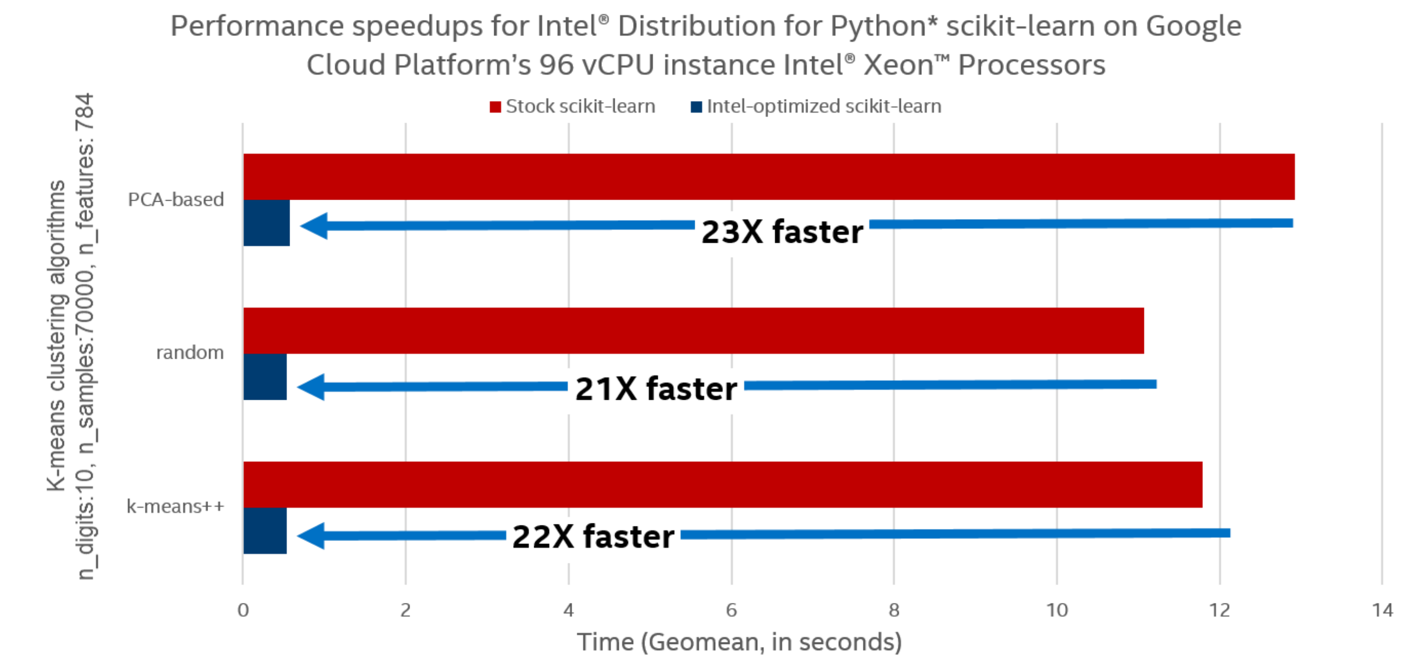
As simple as scikit-learn but handles non-homogeneous, streaming & distributed data



Powerful low-level API for machine learning with non-homogeneous, streaming & distributed data

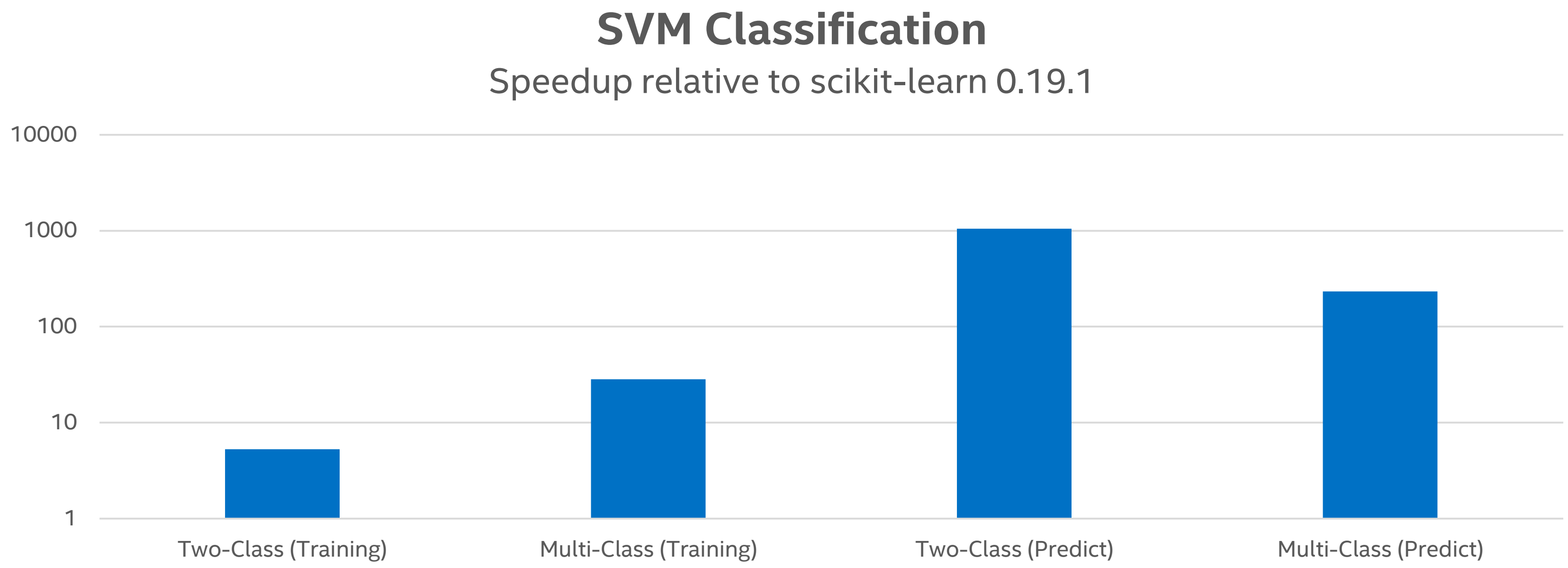
Built upon highly optimized Intel® MKL kernels

Analytics that scales within a node



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

Analytics that scales within a node



Synthetic random data, Linear kernel SVM, 10000 rows, 1000 features, low tolerance=10^{-16}, maxiter==10^{6}. Intel® Distribution for Python* 2018 Update 2, scikit-learn 0.19.1. Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz, 2 sockets, 18 cores/socket, HT:2. RAM: 250GB, Turbo mode and SpeedStep turned off

Distributed computing as simple as Scikit-learn*

Processing in-memory dataset loaded from CSV file

```
import daal4py as d4p
file = "kmeans_dense.csv"
dfin = loadtxt(file), delimiter=',')
centroids = d4p.kmeans_init(10, t_method="plusPlusDense")
result = d4p.kmeans(10).compute(dfin, centroids.compute(dfin))
```

```
python kmeans.py
```

Create numpy array

Parametrize algorithm object

Parametrize and execute in one line

Processing distributed dataset with MPI loaded from multiple CSV file

```
import daal4py as d4p
d4p.daalinit()
files = ["kmeans_dense.csv", …]
dfin = [loadtxt(x, delimiter=',') for x in files]
centroids = d4p.kmeans_init(10, t_method="plusPlusDense", distributed=True)
result = d4p.kmeans(10, distributed=True).compute(dfin, centroids.compute(dfin))
```

```
mpirun -n 4 -genv DIST_CNC=MPI python ./kmeans.py
```

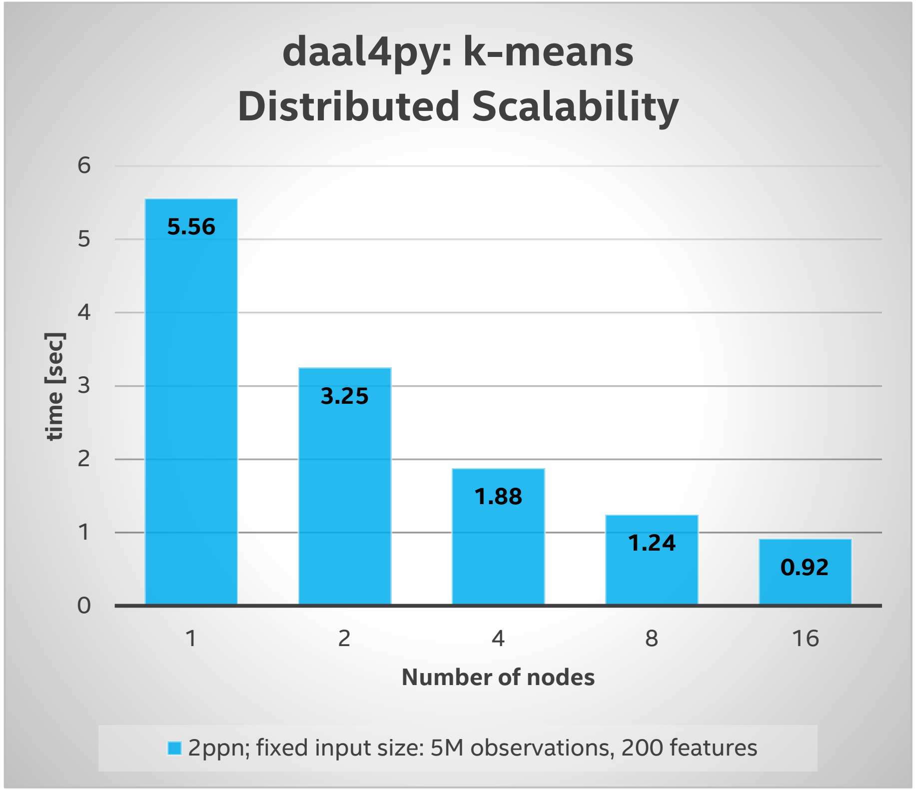
Initialize

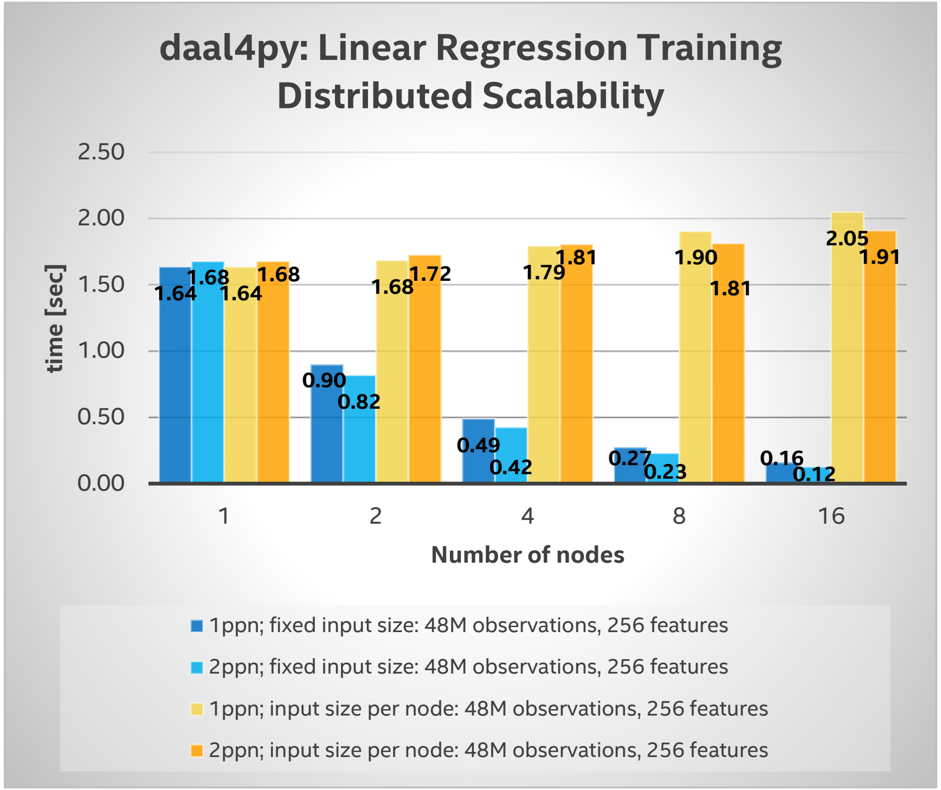
Multiple input arrays/files

Request distributed execution

Try it out `conda install -c intel/label/test dal4py`

Multi-node scaling with DAAL4PY





Configuration Info: 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, Oracle Linux Server release 7.4; Intel® Distribution for Python 2018 Update 1, DAAL4PY (Tech Preview)

Legal Disclaimer & Optimization Notice

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

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information 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

BACKUP

How we enable ecosystem

Day 1 of the new CPU launch

Time

Intel® Distribution for Python*

yum apt

Working with Python vendors

Python Vendor 1

Python Vendor 2

Python Vendor 3

Anaconda Cloud*

Intel conda packages with build recipes & optimization patches

Working with community to upstream

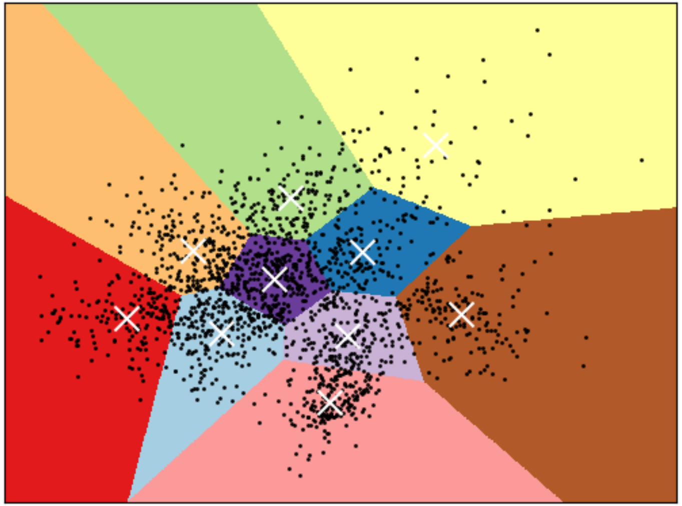
GitHub

PRs with optimization patches

Wheels for Intel runtimes and development packages (MKL, DAAL, TBB, etc.)

python™ Package Index

Clustering MNIST images



http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html#sphx-glr-auto-examples-cluster-plot-kmeans-digits-py

Based on public scikit-learn demo

- Modified variant relies on Intel® Data Analytics Acceleration Library (pyDAAL)

Problem being solved:

- Unsupervised learning
- Clusterization of 70,000 MNIST images of hand-written decimal digits
- Image 28x28 pixels forms a tuple of 784 pixel values (features) that form 784-dimensional feature space
- Algorithm partitions 70,000 points into 10 clusters
- Visualization illustrates 2D projection of the original feature-space points

Benchmark: Black Scholes Formula

Problem: Evaluate fair European call- and put-option price, V_{call} and V_{put}, for underlying stock

Model Parameters:

- S_0 – present underlying stock price
- X – strike price
- σ - stock volatility
- r – risk-free rate
- T - maturity

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

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

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

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

```python
def black_scholes ( nopt, price, strike, t, rate, vol ):
    mr = -rate
    sig_sig_two = vol * vol * 2

    P = price
    S = strike
    T = t

    a = log(P / S)
    b = T * mr

    z = T * sig_sig_two
    c = 0.25 * z
    y = invsqrt(z)

    w1 = (a - b + c) * y
    w2 = (a - b - c) * y

    d1 = 0.5 + 0.5 * erf(w1)
    d2 = 0.5 + 0.5 * erf(w2)

    Se = exp(b) * S

    call = P * d1 - Se * d2
    put = call - P + Se

    return call, put
```

In practice one needs to evaluate many (*nopt*) options for different parameters

Good performance benchmark for stressing VPU and memory

Software