Software

RSE CONFERENCE 2018: GETTING MORE PYTHON PERFORMANCE WITH INTEL® OPTIMIZED DISTRIBUTION FOR PYTHON

Plan for the Workshop: What do YOU want to do?

We have a bit under 90 minutes…

I have a bunch of slides, but we can skip through fast.

We have three hands-on examples

- kmeans clustering (`04_kmeans`) shows
  - Intel optimization benefit in a machine learning application
  - Intel® Vtune profiling
- Black-Scholes shows
  - Profiling (timing in Python, using Intel® Vtune)
  - Use of numpy, numba, Cython (general approaches to Python performance optimization)
- SVM shows
  - How to use scikit-learn for handwritten digit recognition
  - Intel performance gains if you compare runs with/without Intel Python

Proposal: short intro, kmeans (until we've had enough), Black-Scholes

MORE PERFORMANCE USING INTEL® DISTRIBUTION FOR PYTHON

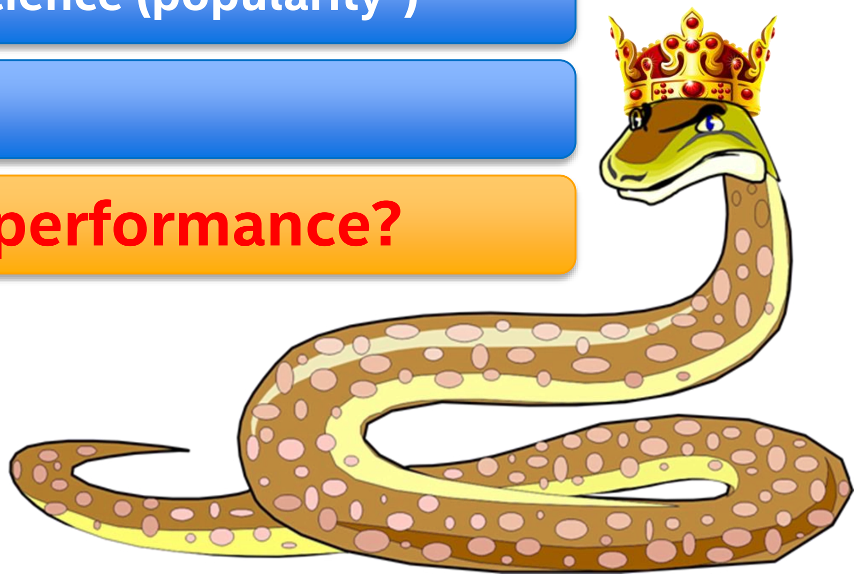
Most Wanted: Python

~7.8 Million out of ~21 Million developers use python (EDC 2016)

#1 language for machine learning/data science (popularity[1])

>100000 python packages on PyPI

But...often slow. Can we get better performance?



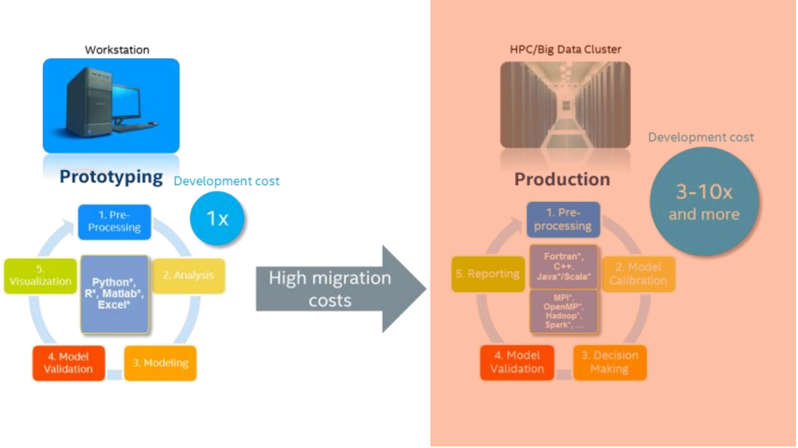
[1] https://www.kdnuggets.com/2017/01/most-popular-language-machine-learning-data-science.html

Faster Python* with Intel® Distribution for Python 2018

High Performance Python Distribution

- Accelerated NumPy, SciPy, scikit-learn well suited for scientific computing, machine learning & data analytics
- Drop-in replacement for existing Python. **No code changes required**
- Highly optimized for latest Intel processors
- **Free for everyone**, or pay to get Priority Support – a direct connection with Intel engineers for technical questions

Get sufficient performance without an extra development cycle



What's Inside Intel® Distribution for Python

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

HIGHER 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/scikit-learn with Intel® MKL[1] & Intel® DAAL[2]	Prebuilt & optimized packages for numerical computing, machine/deep learning, HPC, & data analytics	Compatible & powered by Anaconda*, supports conda & pip
Data analytics, machine learning & deep learning with scikit-learn, pyDAAL	Drop in replacement for existing Python - **No code changes required**	Distribution & individual optimized packages also available at conda & Anaconda.org, YUM/APT, Docker image on DockerHub
Scale with Numba* & Cython*	Jupyter* notebooks, Matplotlib included	Optimizations upstreamed to main Python trunk
Includes optimized mpi4py, works with Dask* & PySpark*	Conda build recipes included in packages	Commercial support through Intel® Parallel Studio XE 2017
Optimized for latest Intel® architectures	**Free download & free for all uses including commercial deployment**	

Intel® Architecture Platforms

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

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

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

Installing Intel® Distribution for Python* 2018

Standalone Installer

Download full installer from
https://software.intel.com/en-us/intel-distribution-for-python

ANACONDA CLOUD

anaconda.org
anaconda.org/intel channel

```
> conda config --add channels intel
> conda install intelpython3_full
> conda install intelpython3_core
```

python™ Package Index

pip
Intel-optimized packages

```
> pip install intel-<pkg-name>
```

Docker Hub
docker

Docker Hub

```
> docker pull intelpython/intelpython3_full
```

YUM/APT
repositories

yum/apt

Access for yum/apt:
https://software.intel.com/en-us/articles/installing-intel-free-libs-and-python

What's New? Intel® Distribution for Python*

What's New in 2018 version

- Updated to latest version of Python 3.6
- Optimized scikit-learn for machine learning speedups
- Conda build recipes for custom infrastructure

What's new in 2019 beta?

Faster Machine learning with Scikit-learn functions

- Support Vector Machine (SVM) and K-means prediction, accelerated with Intel® DAAL

Built-in access to XGBoost library for Machine Learning

- Access to Distributed Gradient Boosting algorithms

Ease of access installation

- Now integrated into Intel® Parallel Studio XE installer.

Access Intel-optimized Python packages through

ACCELERATE PYTHON* PERFORMANCE

Software.intel.com/en-us/distribution-for-python

ANACONDA CLOUD

python Package Index

Docker Hub

docker

YUM/APT

repositories

OUT-OF-THE-BOX PERFORMANCE WITH ACCELERATED NUMERICAL PACKAGES

NUMPY & SCIPY OPTIMIZATIONS

- BLAS/LAPACK using MKL
- MKL-based FFT functionality
- Vectorized, threaded universal functions
  - Use of Intel® C Compiler, and Intel® Fortran Compiler
  - Aligned memory allocation
- Threaded memory copying

https://github.com/IntelPython/mkl_random

MKL-BASED RANDOM NUMBER GENERATION

- Added numpy subpackage numpy.random_intel that mirrors numpy.random in the scope
- Exposes all basic random number generators provided in Intel® MKL.

```
In [3]: import numpy as np, numpy.random_intel as irnd, numpy.random as vrnd

In [4]: irnd.seed(1234,brng='SFMT19937')

In [5]: %time x1 = irnd.randn(10**6)
        CPU times: user 4 ms, sys: 4 ms, total: 8 ms
        Wall time: 7.73 ms

In [6]: %time x2 = vrnd.randn(10**6)
        CPU times: user 44 ms, sys: 0 ns, total: 44 ms
        Wall time: 44.8 ms
```

Intel random number generator

Default numpy random number generator

MKL-BASED FFT

https://github.com/IntelPython/mkl_fft

Intel® Math Kernel Library provides:

- Full stride support, no extra copying
- In-place & out-of-place transforms
- Native support for ND-transforms
- Support for double and single precision

```
In [1]: import scipy.fftpack, numpy as np

In [2]: size = (23, 45, 34)
        x = np.random.randn(*size) + 1j*np.random.randn(*size)
        x = x.T

In [3]: y = scipy.fftpack.fftn(x,overwrite_x=True)

In [4]: y is x

Out[4]: True
```

OPTIMIZED UNIVERSAL FUNCTION LOOPS

Compiler-generated vectorized instructions

with automatic run-time dispatching for different architectures

Vectorized transcendental functions

using Intel® Compiler's SVML

Threaded evaluation

using Intel® MKL's Vector Math Library.

But Wait…There’s More!

Moving beyond optimized Python*, how efficient is your Python/C/C++ application code?

Are there any hard to find sources of performance loss?

Performance analysis gives the answer!

INTRODUCTION TO INTEL® VTUNE™ FOR PYTHON

Tune Python* + Native Code for Better Performance

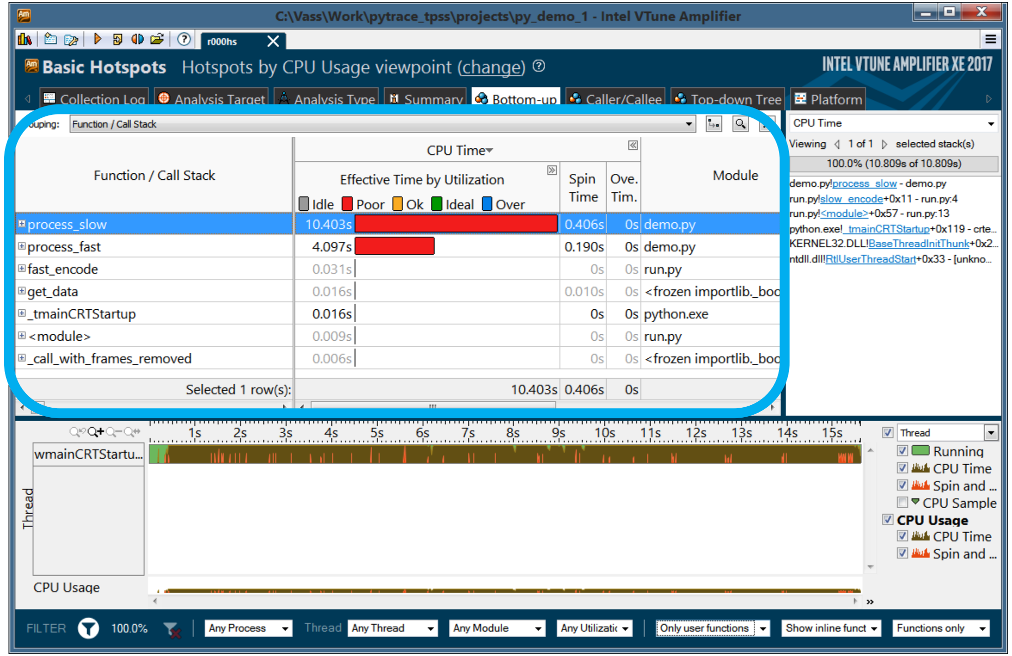
Analyze Performance with Intel® VTune™ Amplifier (available in Intel® Parallel Studio XE)

Challenge

- Single tool that profiles Python + native mixed code applications
- Detection of inefficient runtime execution

Solution

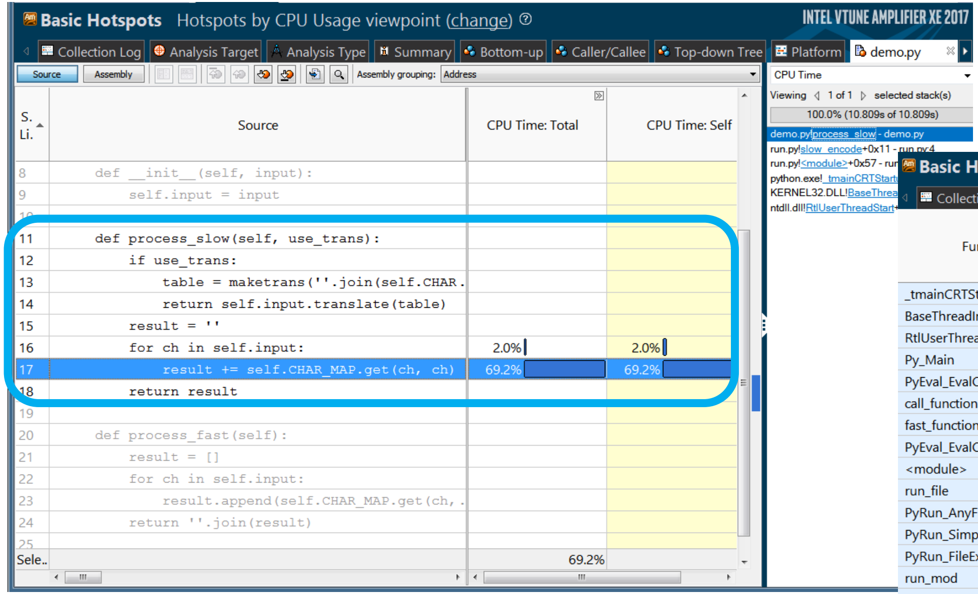
- Auto-detect mixed Python/C/C++ code & extensions
- Accurately identify performance hotspots at line-level
- Low overhead, attach/detach to running application
- Focus your tuning efforts for most impact on performance



Auto detection & performance analysis of Python & native functions

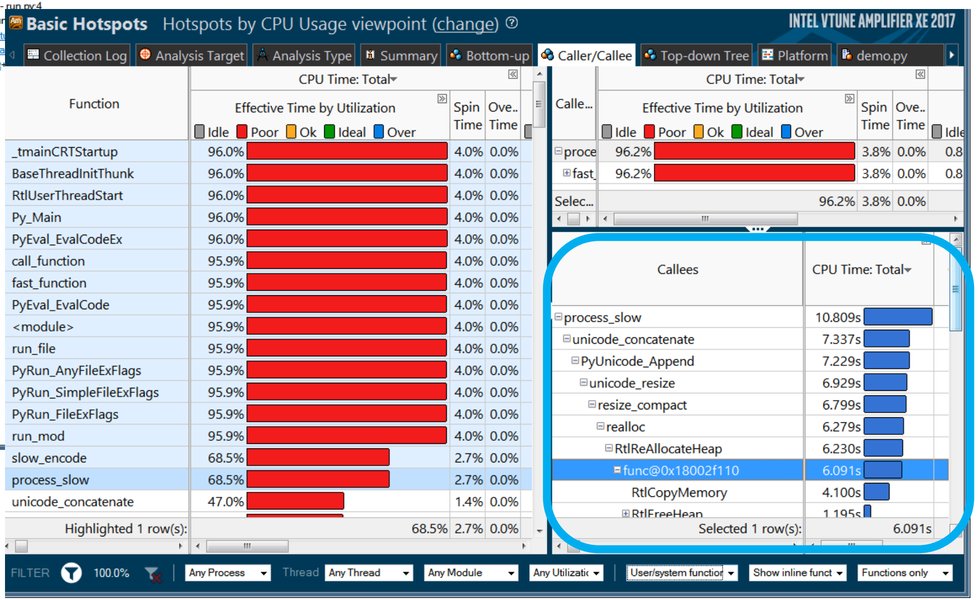
Available in Intel® VTune™ Amplifier & Intel® Parallel Studio XE

Diagnose Problem code quickly & accurately



Identifies exact line of code that is a bottleneck

Details Python* calling into native functions



A two-pronged approach to Higher Python* Performance

Step 1: Use Intel® Distribution for Python

- Leverage optimized native libraries for performance
- Drop-in replacement for your current Python - **no code changes required**
- Optimized for multi-core and latest Intel processor

Step 2: Use **Intel® VTune™ Amplifier** for profiling

- Get detailed summary of entire application execution profile
- Auto-detects & profiles Python/C/C++ mixed code & extensions with low overhead
- Accurately detect hotspots - line level analysis helps you to make smart optimization decisions fast!
- Available in Intel® Parallel Studio XE Professional & Cluster Edition

More Resources

Intel® Distribution for Python

- Product page – overview, features, FAQs…
- Training materials – movies, tech briefs, documentation, evaluation guides…
- Support – forums, secure support…

Intel® VTune Amplifier

- Product page – overview, features, FAQs…
- Training materials – movies, tech briefs, documentation, evaluation guides…
- Reviews
- Support – forums, secure support…

USING CONDA

Conda

Package manager

- install/remove packages
- handles dependences
- also non-python packages (such as native libs)

Environments

- isolate different sets of packages/versions
- creates hard-links when possible
- similar to virtualenv

Conda basics

Getting started

- conda --help
- conda list
- conda search numpy
- conda search numpy -c intel -c conda-forge

Environments

- conda env list
- conda create -n sandbox -c intel python=3.6
- source activate sandbox
- conda list

- Package management
  - conda install numpy
  - conda remove numpy

Jupyter Notebooks

NOTEBOOKS

- Interactive (via web browser)
- Python, markup, and shell (and other)
- Python kernel running
- Cells get executed individually
- State is carried over between cell execution
- Output appears back in the notebook
- Can also be used to run julia codes (ask me about Julia later if you're interested!)

Preparing hands-on sessions

```
% To enable Vtune profiling (password: workshops)
sudo bash
echo 0 > /proc/sys/kernel/yama/ptrace_scope
exit
cd ~/Cownie
source conda.sh
conda deactivate
% To use pre-installed code (easier...)
conda activate intel_env
% Or, to update packages if you have internet access
conda create -n <your_env_name> -c intel python=3.6 \
notebook numpy mpi4py matplotlib numexpr numba cython
conda activate <your_env_name>
source /opt/intel/vtune_amplifier/amplxe-vars.sh
which amplxe-cl && which amplxe-gui && which icc
cd material
which python
jupyter notebook
```

Hands-on files

Initial versions of the codes are at the top level in files with obvious names

- These may be incomplete, and require that you fill in some code

Suggested solutions are in … the `solutions` sub-directory

Hands-On: Kmeans

Cluster Analysis

Problems

A news provider wants to group the news with similar headlines in the same section

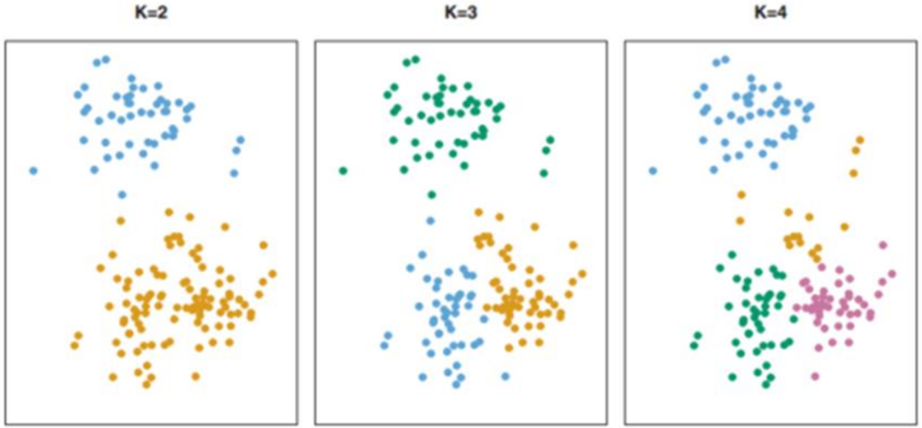
Humans with similar genetic pattern are grouped together to identify correlation with a specific disease

Solution: K-Means

Pick ***k*** centroids

Repeat until converge:

- Assign data points to the closest centroid
- Re-calculate centroids as the mean of all points in the current cluster
- Re-assign data points to the closest centroid



Source: Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2014). *An Introduction to Statistical Learning*. Springer

Hands-On Example: Kmeans (1)

Use kmeans clustering to choose an optimal small number of colours to compress an image

Use `04_kmeans.ipynb` and walk through the code.

- To see original performance, disable Intel optimizations by finding this and executing it before running the code.

```
from sklearn.daal4sklearn import dispatcher
# TODO: Disable Intel-optimizations (DAAL) and repeat fit and predict and get timings
# call disable() on dispatcher before calling KMeans
dispatcher.disable()
```

- Don't worry about the deprecated function used for reading the image.

Hands-On Example: Kmeans (2)

- You could profile this version, or just change this to enable(), as below, run that cell and then run the fit again.

```
from sklearn.daal4sklearn import dispatcher
# TODO: Disable Intel-optimizations (DAAL) and repeat fit and predict and
get timings
# call disable() on dispatcher before calling KMeans
dispatcher.enable()
```

Kmeans conclusions

With no code changes at all Intel® Distribution for Python can give >100x speedup on this machine-learning code (precise gain will depend on the machine you are running on, of course, you'd get more on a many core server…)

We started to use Intel® Vtune Amplifier

- Saw that it can show us parallelism
- Saw that it can show us Python code up the call tree from the expensive operations

HANDS-ON PYTHON PERFORMANCE TOOLS
PERFORMANCE TOOLS APPLIED TO BLACK SCHOLES

Hands-On Example : Black-Scholes What is Black-Scholes?

A Financial Services Industry benchmark (sorry ☺)

It is used for option pricing

Luckily we don't need to understand the details of the theory

From our point of view

- It is a relatively small piece of scientific Python
- It uses transcendental functions (`log, exp, erf, sqrt`)
- It's small enough to play with but big enough to show some general principals

Black Scholes – Naïve

Naïve

Plain obvious python

Kernel works on vectors and scalars

Output 2 vectors

Use timeit, cprofile and line profiler

Using parallelism implicitly

Similar syntax
- Vector based (no loop)
- List -> np.array
- Import from numpy

Numpy

Mathematical building blocks

Universal functions (ufuncs) operate on vectors

Under the hood accelerates computation (e.g. with Intel® MKL)

NumExpr

Hands-On Numpy

Black Scholes option pricing: `01_numpy_blackscholes.ipynb`

Numpy

Naïve '[]' are lists, not arrays

Numpy provides fast array implementation

- contiguous memory
- written in C

Numpy also provides optimized operations on arrays

- vector/array/scalar operations allow calling vectorized libraries
- Umath dispatches to right vector/array/scalar implementation

- Intel-optimized Numpy is accelerated with Intel® Math Kernel Library
- Also accelerates FFT, RNG, Umath, …

Using parallelism under the hood

Numpy

NumExpr

Evaluates strings of numeric expressions

Internally translates to numpy ufuncs

Hands-On numexpr

Black Scholes option pricing: 02_numexpr_blackscholes.ipynb

Compiling Python

Numba

Just-In-Time (LLVM-based, produces parallel code)

Decorators annotate functions

JIT API

Cython

Hands-On numba

Black Scholes option pricing: 03_numba_blackscholes.ipynb

Compiling Python - Numba

Decorators

Defining instantiations

GIL, nopython

API and target

Hands-On: Support Vector Machine classifier

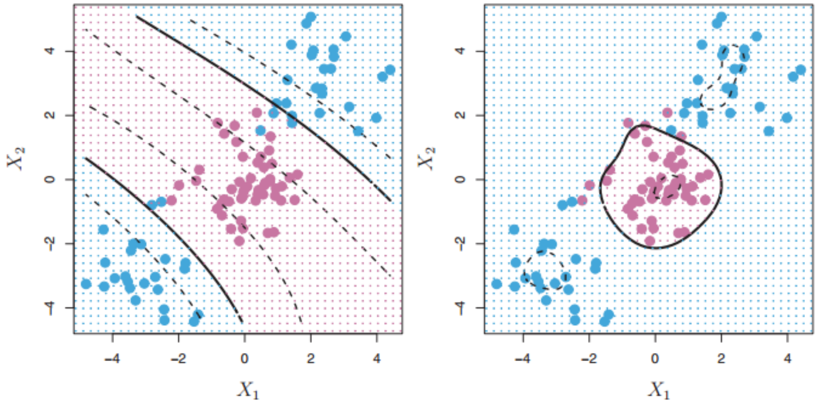
Classification

Problems

- An emailing service provider wants to build a spam filter for the customers
- A postal service wants to implement handwritten address interpretation

Solution: Support Vector Machine (SVM)

- Works well for non-linear decision boundary
- Two kernel functions are provided:
  - Linear kernel
  - Gaussian kernel (RBF)
- Multi-class classifier
  - One-vs-One



Source: Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2014). *An Introduction to Statistical Learning.* Springer

Hands-On: SVM

`05_svm.ipynb`

Performs digit recognition for handwritten numerals

Conclusions and learning points are similar to Kmeans

Key Points from the Workshop

Intel® Distribution for Python

- Is **FREE** for everyone and any use (commercial included)
  - If you want guaranteed support you can pay
- Can be installed via `conda`, `pip`, … so it should be easy to add to your environment
- Can hugely improve the performance of *some* unmodified Python code (especially popular areas like ML/DL)
  - Uses threads
  - Uses optimised, vectorized, parallelized, maths libraries

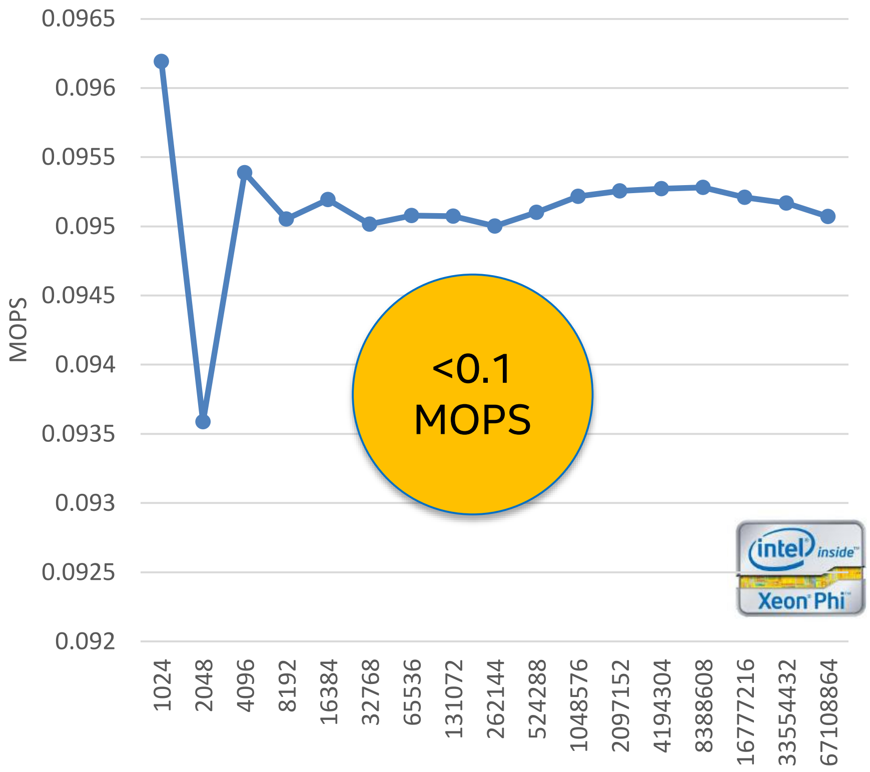
Intel® Vtune™ Amplifier

- Is **FREE** for educators and students
- Can profile mixed Python and compiled code (so you can attribute time in native code back to the Python which called it)

THE END

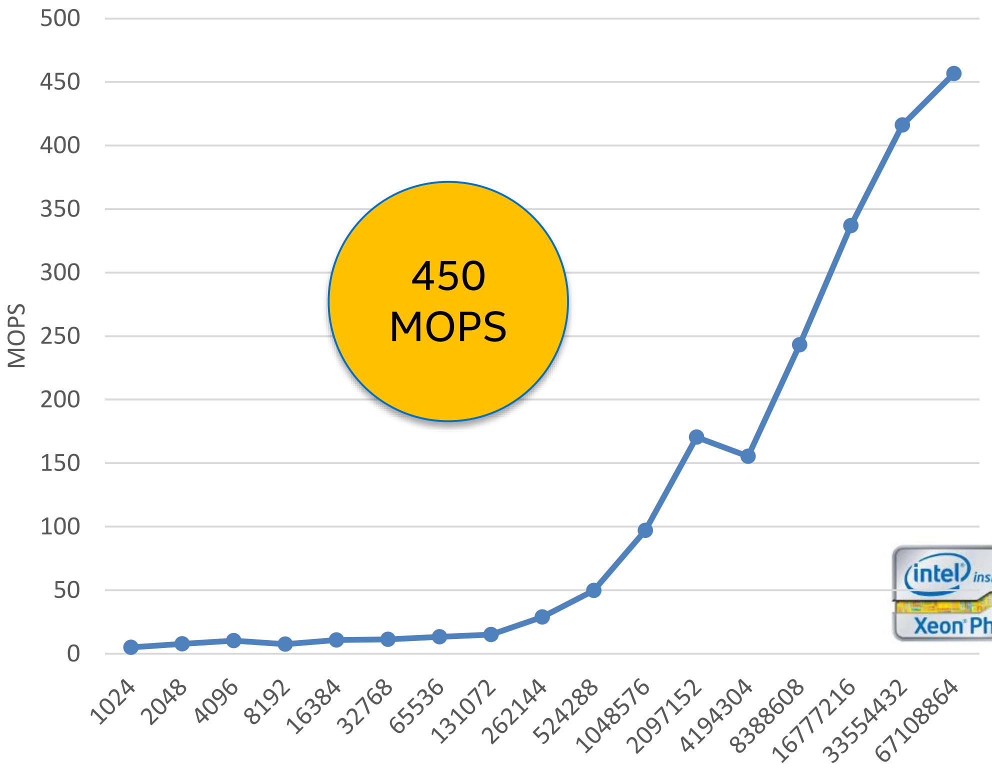
BACKUP

Variant 1: Plain Python



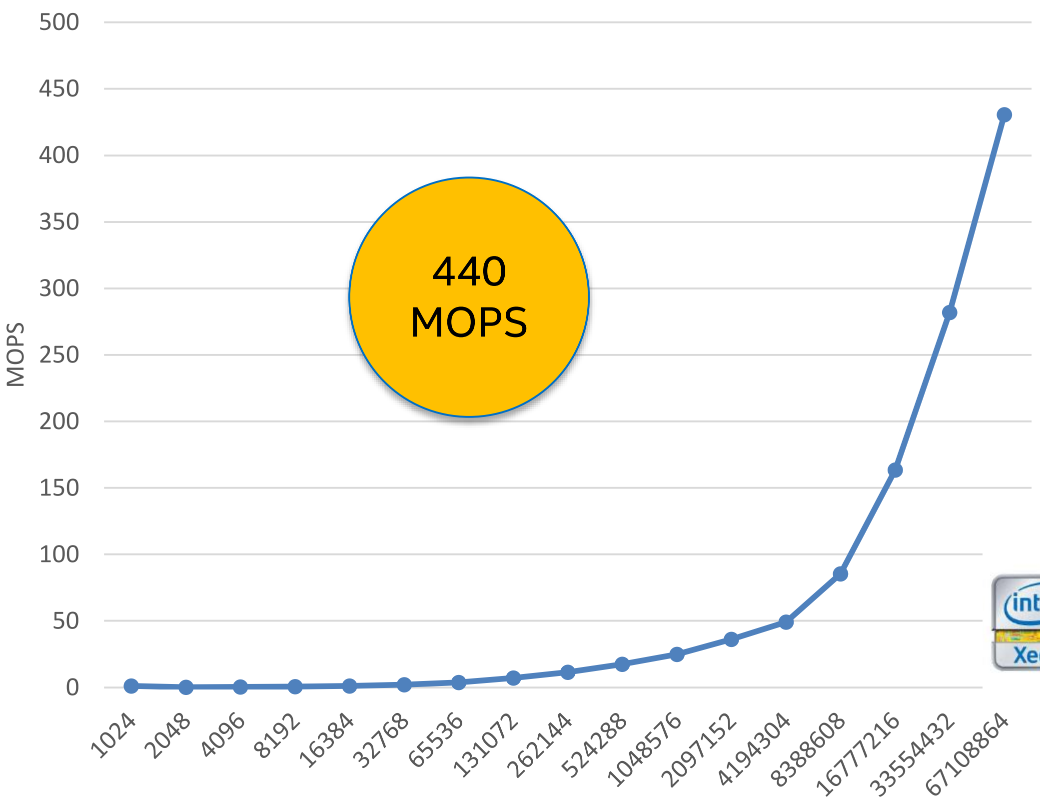

```python
def black_scholes ( nopt, price, strike, t, rate, vol, call, put ):
    mr = -rate
    sig_sig_two = vol * vol * 2

    for i in range(nopt):
        P = float( price [i] )
        S = strike [i]
        T = t [i]

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

        z = T * sig_sig_two
        c = 0.25 * z
        y = 1/sqrt(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 [i] = P * d1 - Se * d2
        put [i] = call [i] - P + Se
```

Variant 2: NumPy* arrays and Umath functions




```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
```

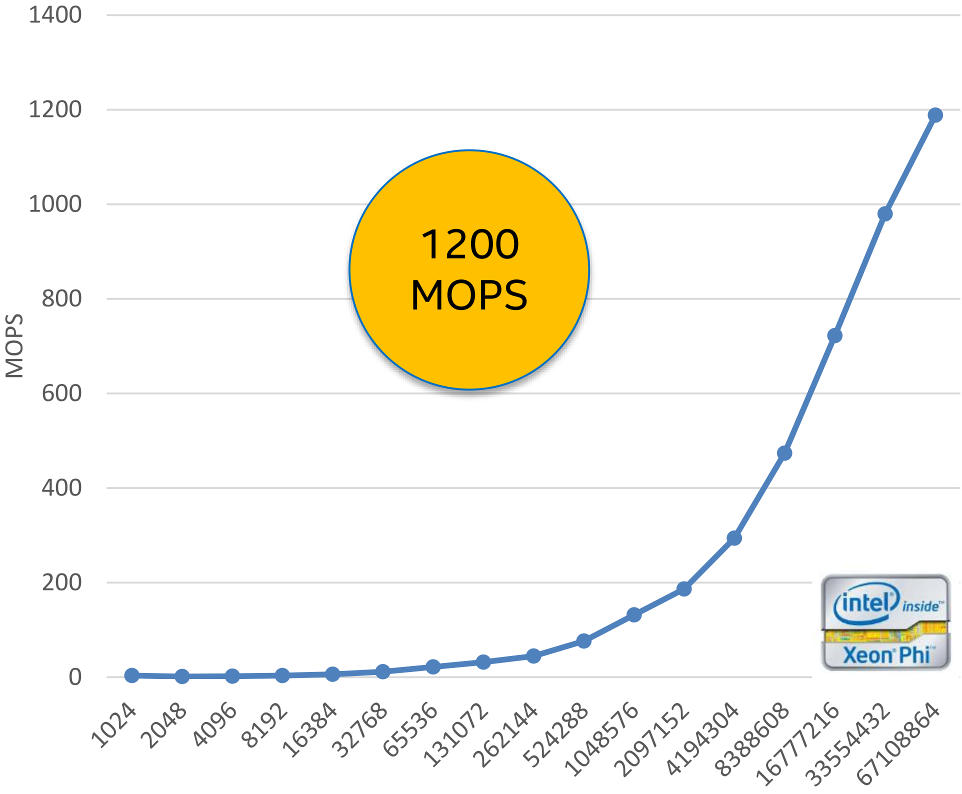
Variant 3: NumExpr* (proxy for Umath implementation)




```python
import numexpr as ne

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

        P = price
        S = strike
        T = t

        a = ne.evaluate("log(P / S) ")
        b = ne.evaluate("T * mr ")

        z = ne.evaluate("T * sig_sig_two ")
        c = ne.evaluate("0.25 * z ")
        y = ne.evaluate("1/sqrt(z) ")

        w1 = ne.evaluate("(a - b + c) * y ")
        w2 = ne.evaluate("(a - b - c) * y ")

        d1 = ne.evaluate("0.5 + 0.5 * erf(w1) ")
        d2 = ne.evaluate("0.5 + 0.5 * erf(w2) ")

        Se = ne.evaluate("exp(b) * S ")

        call = ne.evaluate("P * d1 - Se * d2 ")
        put = ne.evaluate("call - P + Se ")

        return call, put

ne.set_num_threads(ne.detect_number_of_cores())
base_bs_erf.run("Numexpr", black_scholes)
```

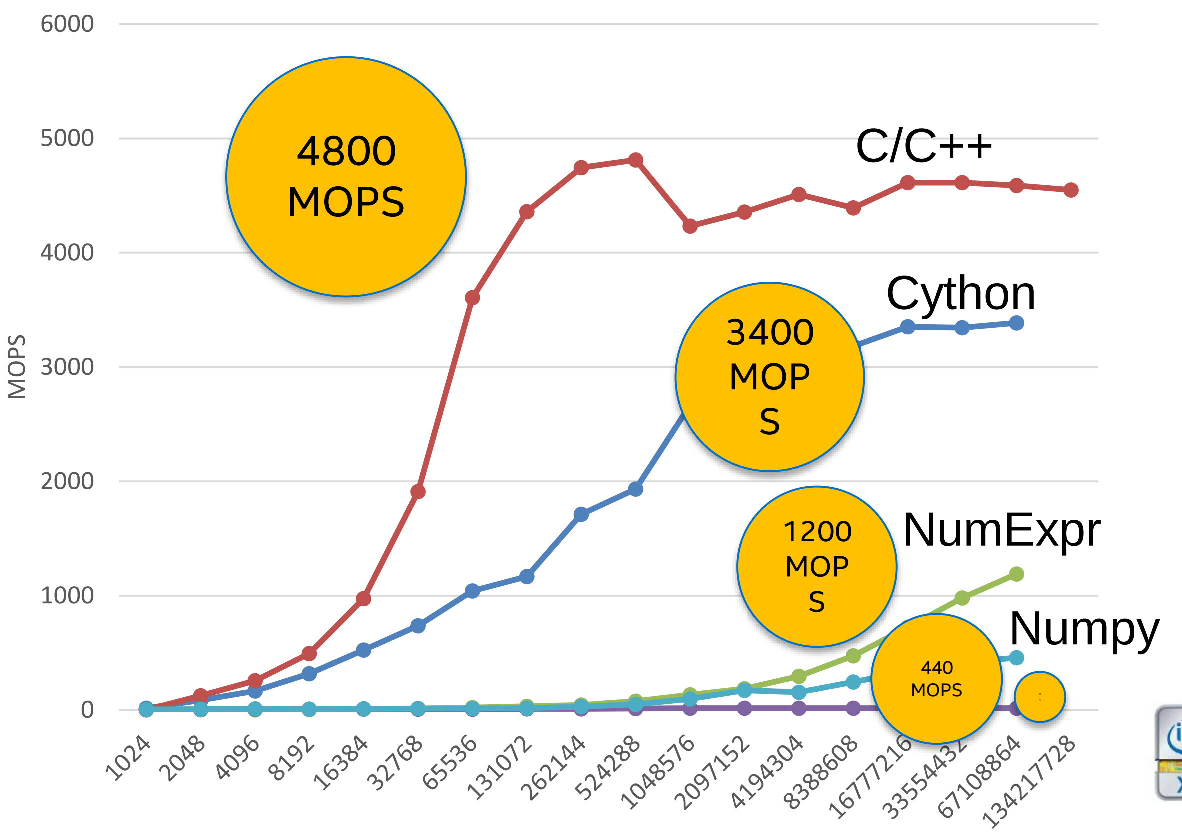
Optimization Notice

Variant 4: NumExpr* (most performant)




```python
import base_bs_erf
import numexpr as ne

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

    P = price
    S = strike
    T = t

    call = ne.evaluate("P * (0.5 + 0.5 * erf((log(P / S) - T * mr +" +
    "0.25 * T * sig_sig_two) * 1/sqrt(T * sig_sig_two))) - S * exp(T * mr)*" +
    "(0.5 + 0.5 * erf((log(P / S) - T * mr - 0.25 * T * sig_sig_two) *" +
    "1/sqrt(T * sig_sig_two))) ")
    put = ne.evaluate("call - P + S * exp(T * mr) ")

    return call, put
```

Variant 5: Native C/C++ vs. Python variants



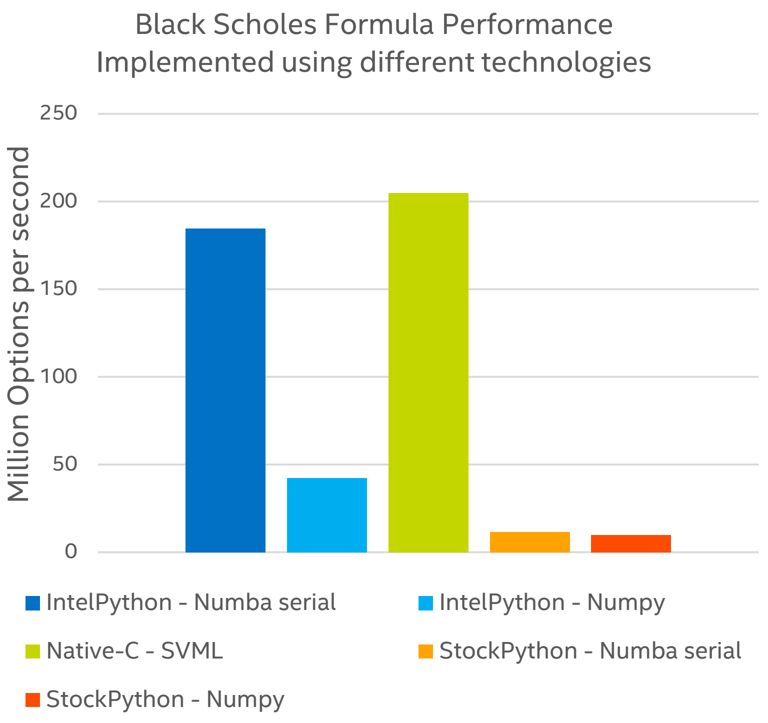
Optimizing NumPy, SciPy, NumExpr to scale

Data Analytics pipelines do not always fully match Machine Learning library functions

- Need to implement custom data transformations
- Need to provide custom optimization functions/kernels
- … And these are performance hotspots sometimes

Pure Python implementation kills performance but there are better alternatives within Python

- NumPy – array programming
- Cython – compiles Python code into native executable
- Numba – JIT compiler to accelerate performance hotspots



Configuration Info: Intel® Xeon® Gold 6140 CPU @ 2.30GHz (2 sockets, 18 cores per socket, 1 thread per core - HT is off), 256GB DDR4. Stock Python: CentOS Linux release 7.3.1611 (Core), python 3.6.4, numpy 1.14.0, numba 0.36.2, llvmlite 0.21.0. Intel® Distribution for Python 2018 Update 2 packages: python 3.6.3 intel_8, numpy 1.14.0 intel_3, numba 0.36.2 intel_2, llvmlite 0.21.1 (intel), mkl 2018.0.2 intel_1, openmp 2018.0.1 intel_0 intel®

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.
-
-

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

LEGAL NOTICES & DISCLAIMERS

CONFIGURATION INFORMATION

Hardware:

Intel® Core™ i7-7567U CPU @ 3.50GHz (1 socket, 2 cores per socket, 2 threads per core), 32GB DDR4 @ 2133MHz
Intel® Xeon® CPU E5-2699 v4 @ 2.20GHz (2 sockets, 22 cores per socket, 1 thread per core - HT is off), 256GB DDR4 @ 2400MHz
Intel® Xeon Phi™ CPU 7250 @ 1.40GHz (1 socket, 68 cores per socket, 4 threads per core), 192GB DDR4 @ 1200MHz, 16GB MCDRAM @ 7200MHz in cache mode

Software:

Stock: CentOS Linux release 7.3.1611 (Core), python 3.6.2, pip 9.0.1, numpy 1.13.1, scipy 0.19.1, scikit-learn 0.19.0
Intel® Distribution for Python 2018 Gold packages: mkl 2018.0.0 intel_4, daal 2018.0.0.20170814, numpy 1.13.1 py36_intel_15, openmp 2018.0.0 intel_7, scipy 0.19.1 np113py36_intel_11, scikit-learn 0.18.2 np113py36_intel_3