Intel® Distribution for Python*

High Performance Python for Data Analytics and More

Asma Farjallah

Courtesy of Frank Schlimbach

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.

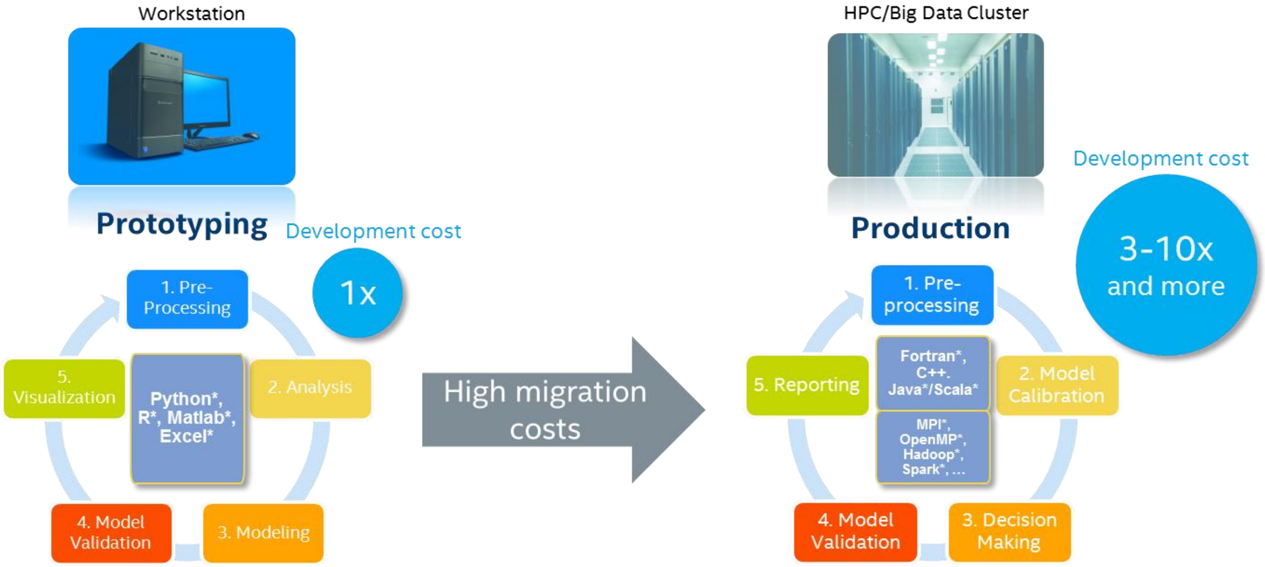
Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase.

Results have been simulated and are provided for informational purposes only. Results were derived using simulations run on an architecture simulator or model. Any difference in system hardware or software design or configuration may affect actual performance.

For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

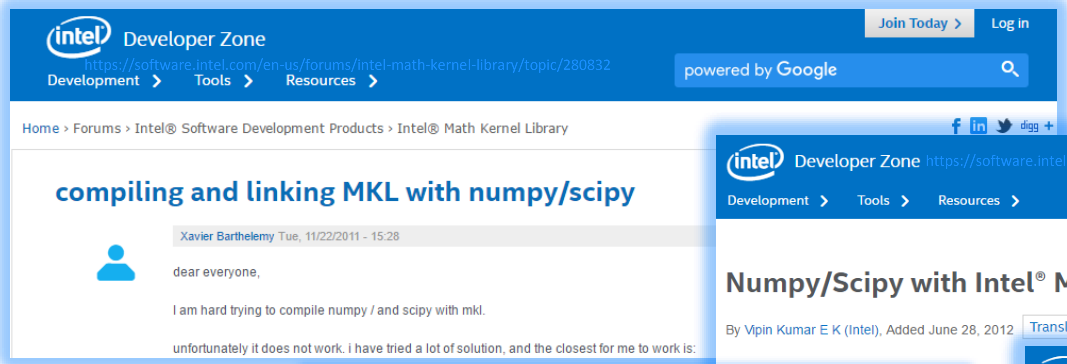
Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

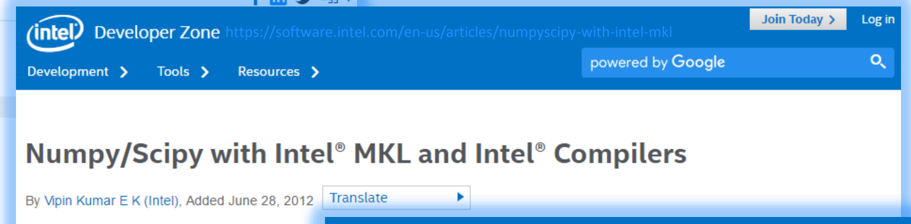
WHAT PROBLEMS WE SOLVE:
1. PRODUCTIVE PERFORMANCE

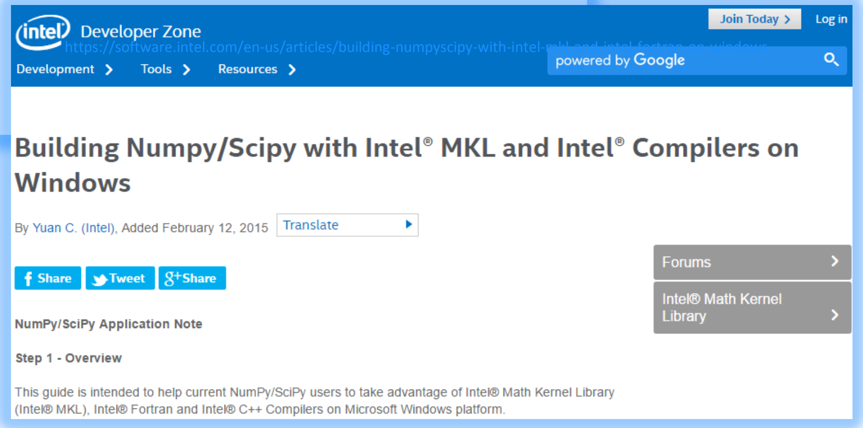


Make Python* usable beyond prototyping environment by scaling out to HPC and Big Data environments

WHAT PROBLEMS WE SOLVE:
2. EASE-OF-USE





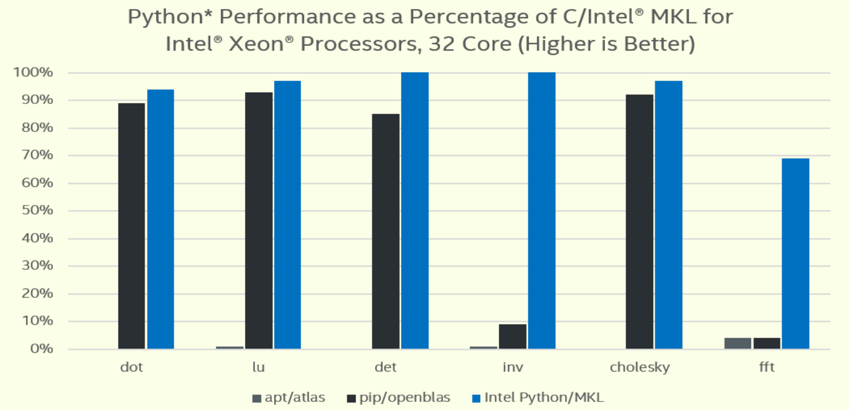


"Any articles I found on your site that related to actually using the MKL for compiling something were overly technical. ***I couldn't figure out what the heck some of the things were doing or talking about****."* – Intel® Parallel Studio 2015 Beta Survey Response

WHAT PROBLEMS WE SOLVE:
3. FAST ACCESS TO OPTIMIZATIONS

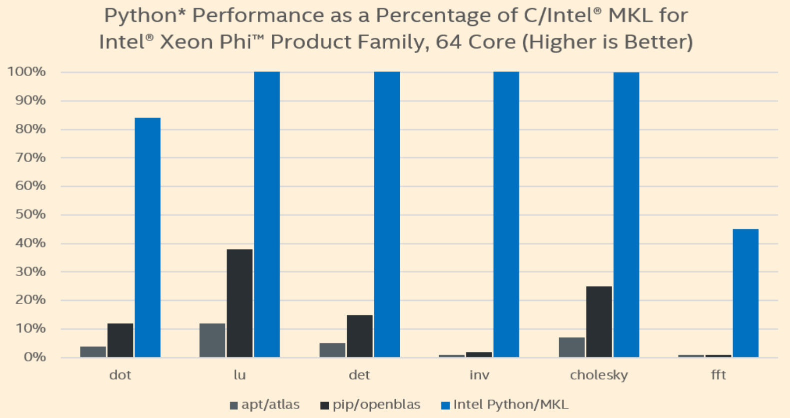
Mature AVX2 instructions based product

Intel® Xeon® Processors



New AVX512 instructions based product

Intel® Xeon Phi™ Product Family



Configuration Info: apt/atlas: installed with apt-get, Ubuntu 16.10, python 3.5.2, numpy* 1.11.0, scipy* 0.17.0; pip/openblas: installed with pip, Ubuntu 16.10, python 3.5.2, numpy 1.11.1, scipy 0.18.0; Intel Python: Intel Distribution for Python 2017;. Hardware: Xeon: Intel Xeon CPU E5-2698 v3 @ 2.30 GHz (2 sockets, 16 cores each, HT=off), 64 GB of RAM, 8 DIMMS of 8GB@2133MHz; Xeon Phi: Intel Intel® Xeon Phi™ CPU 7210 1.30 GHz, 96 GB of RAM, 6 DIMMS of 16GB@1200MHz

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. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

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 .

WHAT'S IN INTEL® DISTRIBUTION FOR PYTHON*?
SCIPY-STACK + SELECTED BIGDATA/ML/HPC PACKAGES

Math/Compute

- Numpy*
- Scipy*
- pyDAAL
- Scikit-learn*
- Numexpr*
- Sympy*
- Mpmath*
- Caffe*
- Theano*

Intel® MKL
Intel® IPP
Intel® DAAL
Intel® Compiler

Parallelism/Performance

- TBB
- Mpi4py*
- Numba*
- Cython*
- Pyzmq*
- Distarray*
- Pandas*
- Pytables*
- H5py*

Intel® TBB
Intel® MPI
Intel® Compiler

Productivity

- Conda*
- Pip*
- Jupyter*
- Notebook*
- Matplotlib*
- Nose*/pytest*/mock*

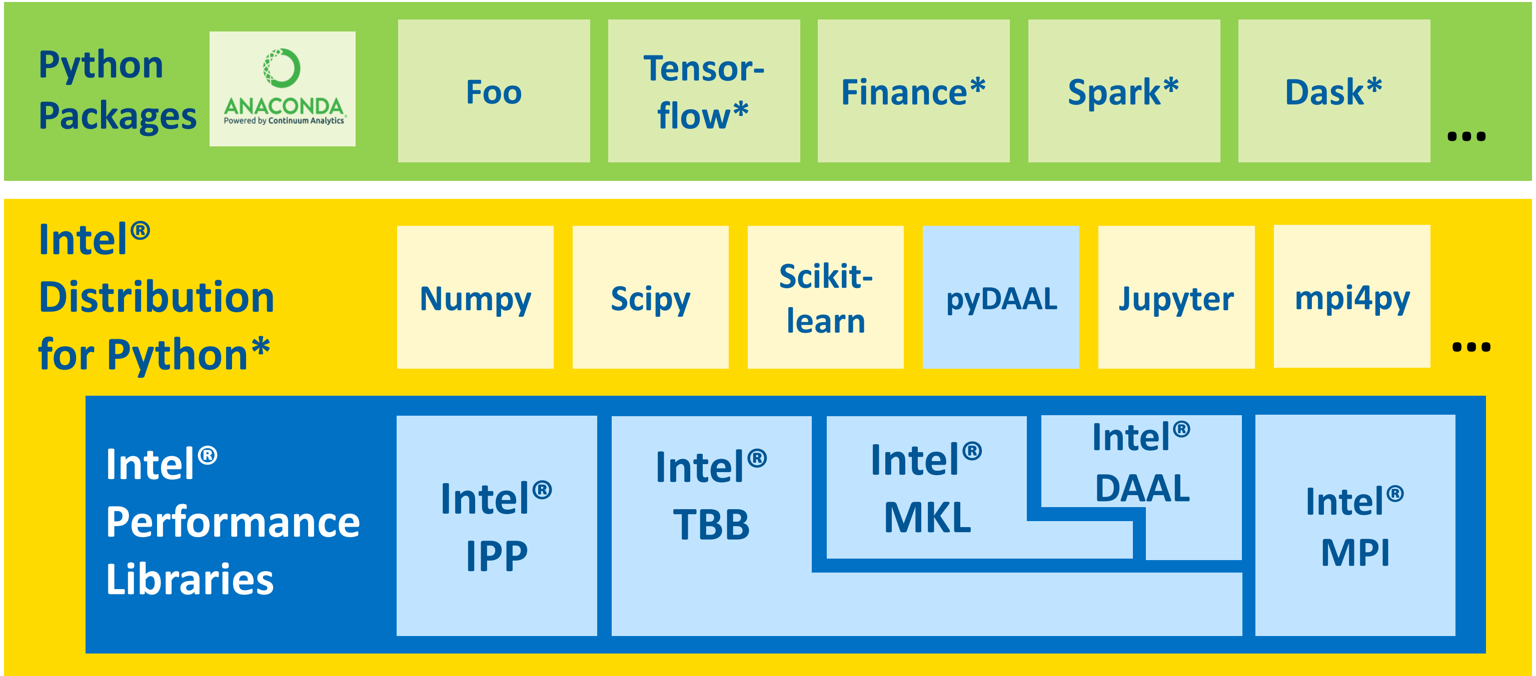
Powered by ANACONDA
Powered by Continuum Analytics

Misc

- Python 2.7/3.5
- Jinja2*
- Pyyaml*
- Tornado*
- Llvmlite*
- Six*
- MarkupSafe*
- Pytz*
- Dateutil*

…

INTEL® DISTRIBUTION FOR PYTHON* LANDSCAPE



INSTALLING INTEL® DISTRIBUTION FOR PYTHON*

Stand-alone installer and on anaconda.org/intel

Linux*

Windows*

MacOS*

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

or

```
> conda config --add channels intel
> conda create -n ip3 intelpython3_full
> source activate ip3
```

INTEL® DISTRIBUTION FOR PYTHON* RELEASES

2017 | **2017 U1** | **2017 U2** | **2017 U3** | **2017 U4**

Intel® Parallel Studio XE 2017 libraries

2017
- Scipy-stack
- Random_intel
- Performance
- TBB
- pyDAAL

2017 U1
- Performance
- Usability
- Neural networks (pyDAAL)
- Docker images

2017 U2
- Faster FFT
- Faster Umath
- Faster Memory
- Faster Skit-learn

2017 U3
- MLSL
- Theano
- Caffe
- OpenCV

Continuous on Anaconda.org

Python 3.6

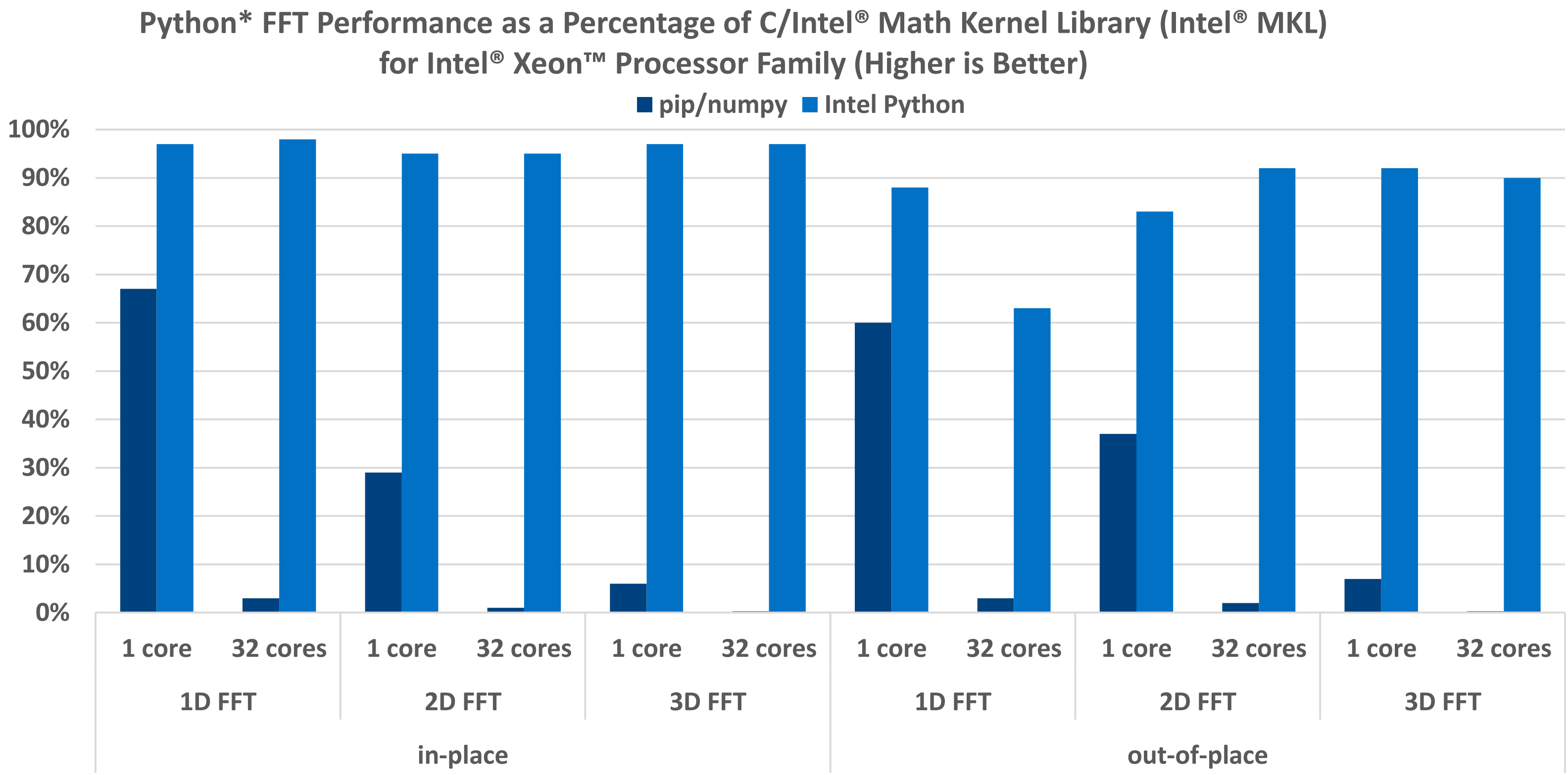
Intel® Parallel Studio XE 2018 (Beta) libraries

2018 Beta | **2018 BetaU** | **2018**

Sep 2016 | Oct 2016 | Feb 2017 | Apr 2017 | May 2017 | Jun 2017 | Sep 2017

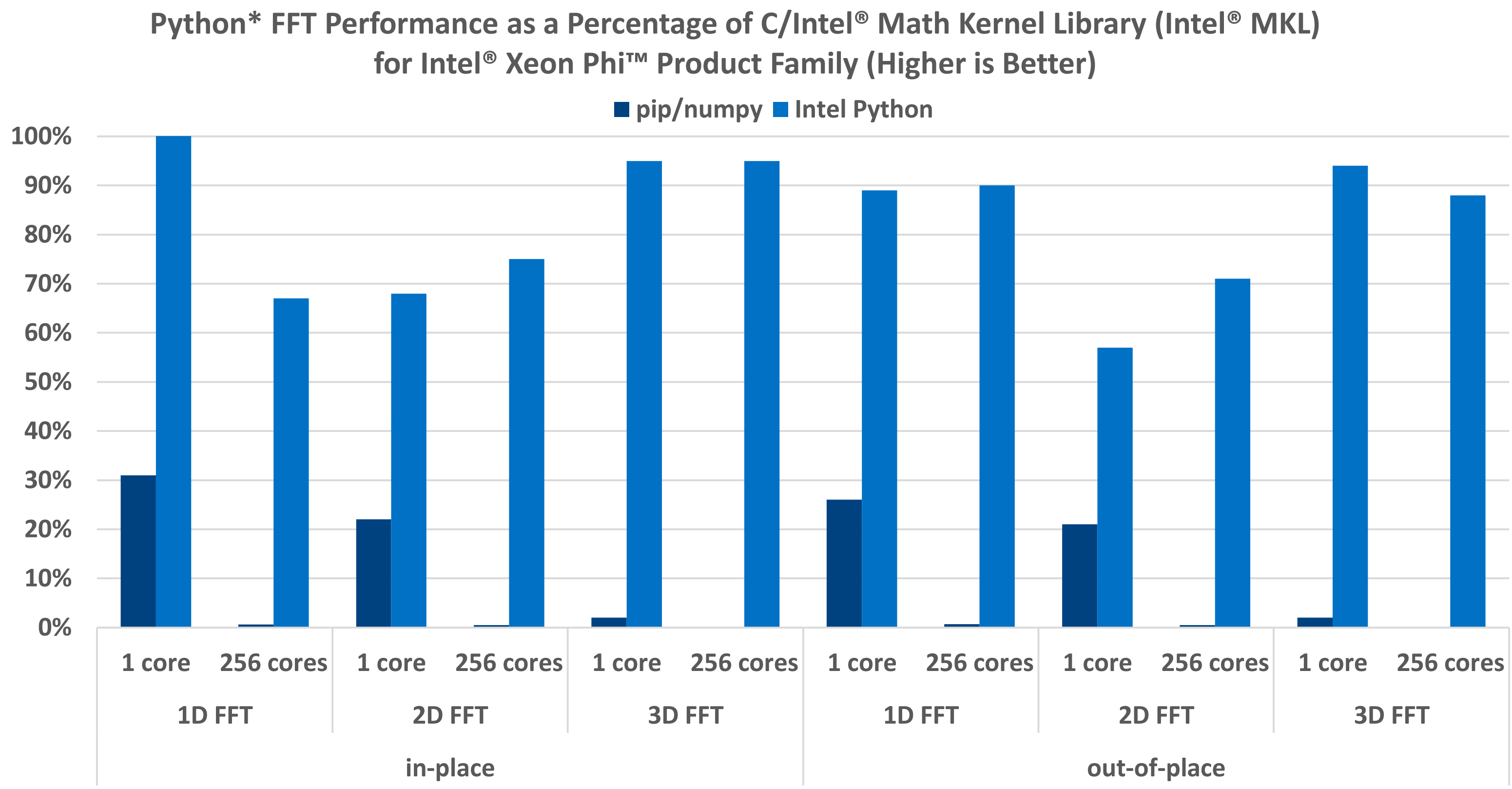
OUT-OF-THE-BOX PERFORMANCE WITH ACCELERATED NUMERICAL PACKAGES

WIDESPREAD OPTIMIZATIONS IN NUMPY & SCIPY FFT



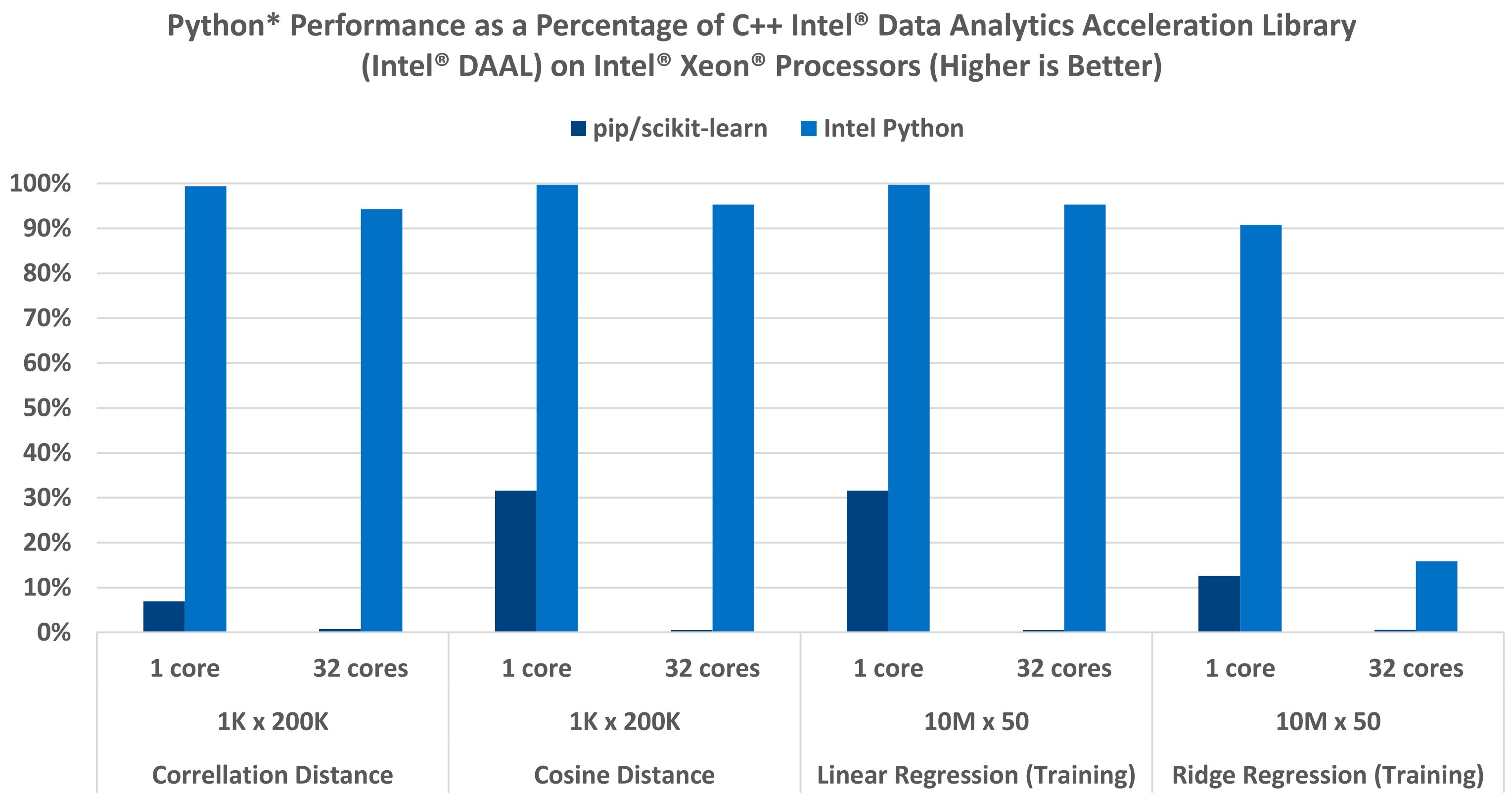
Configuration:
Software:
Pip*/NumPy*: Installed with Pip, Ubuntu*, Python* 3.5.2, NumPy=1.12.1, scikit-learn*=0.18.1, Intel® Distribution for Python 2017, Update 2
Hardware:
Intel® Xeon® E5-2698 v3 processor @ 2.30 GHz (2 sockets, 16 cores each, 1 thread per core), 64GB of DRAM; Intel® Xeon Phi™ processor 7210 @ 1.30 GHz (1 socket, 64 cores, 4 threads per core), DRAM 32 GB, MCDRAM (Flat mode enabled) 16GB

WIDESPREAD OPTIMIZATIONS IN NUMPY & SCIPY FFT



Configuration:
Software:
Pip*/NumPy*: Installed with Pip, Ubuntu*, Python* 3.5.2, NumPy=1.12.1, scikit-learn*=0.18.1, Intel® Distribution for Python 2017, Update 2
Hardware:
Intel® Xeon® E5-2698 v3 processor @ 2.30 GHz (2 sockets, 16 cores each, 1 thread per core), 64GB of DRAM; Intel® Xeon Phi™ processor 7210 @ 1.30 GHz (1 socket, 64 cores, 4 threads per core), DRAM 32 GB, MCDRAM (Flat mode enabled) 16GB

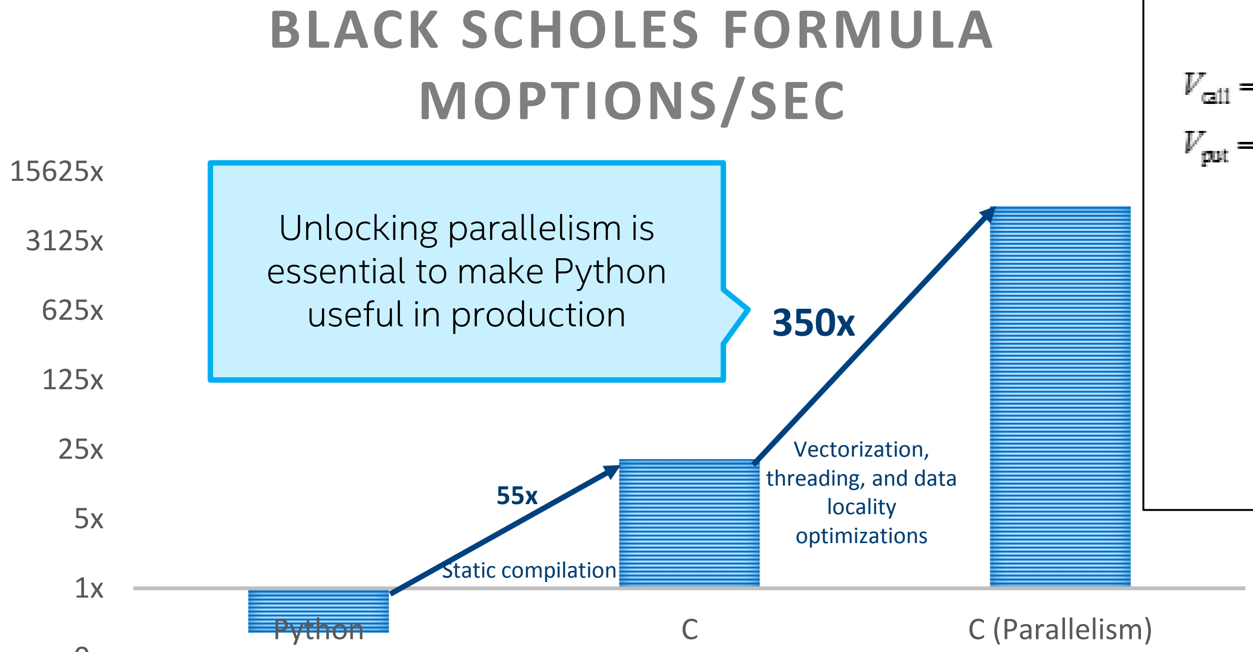
INTEL® DAAL IN SCIKIT-LEARN*



Configuration:
Software:
Pip*/NumPy*: Installed with Pip, Ubuntu*, Python* 3.5.2, NumPy=1.12.1, scikit-learn*=0.18.1, Intel® Distribution for Python 2017, Update 2
Hardware:
Intel® Xeon® E5-2698 v3 processor @ 2.30 GHz (2 sockets, 16 cores each, 1 thread per core), 64GB of DRAM; Intel® Xeon Phi™ processor 7210 @ 1.30 GHz (1 socket, 64 cores, 4 threads per core), DRAM 32 GB, MCDRAM (Flat mode enabled) 16GB

PARALLELISM

WHY PARALLELISM MATTERS



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

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

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

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

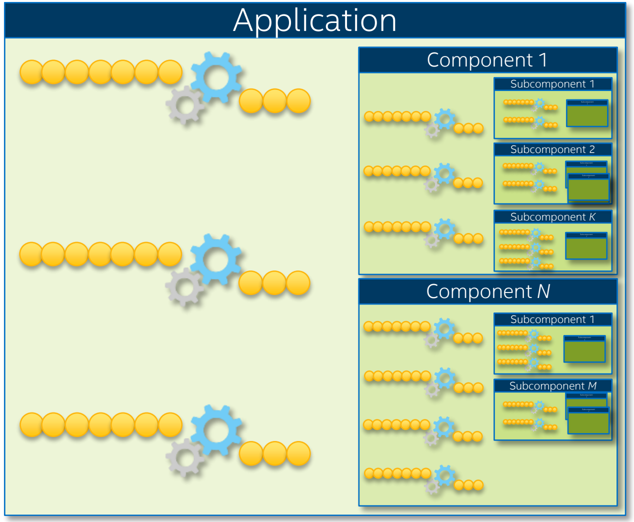
Configuration info: - Versions: Intel® Distribution for Python 2.7.10 Technical Preview 1 (Aug 03, 2015), icc 15.0; Hardware: Intel® Xeon® CPU E5-2698 v3 @ 2.30GHz (2 sockets, 16 cores each, HT=OFF), 64 GB of RAM, 8 DIMMS of 8GB@2133MHz; Operating System: Ubuntu 14.04 LTS.

SHARED-MEMORY PARALLELISM

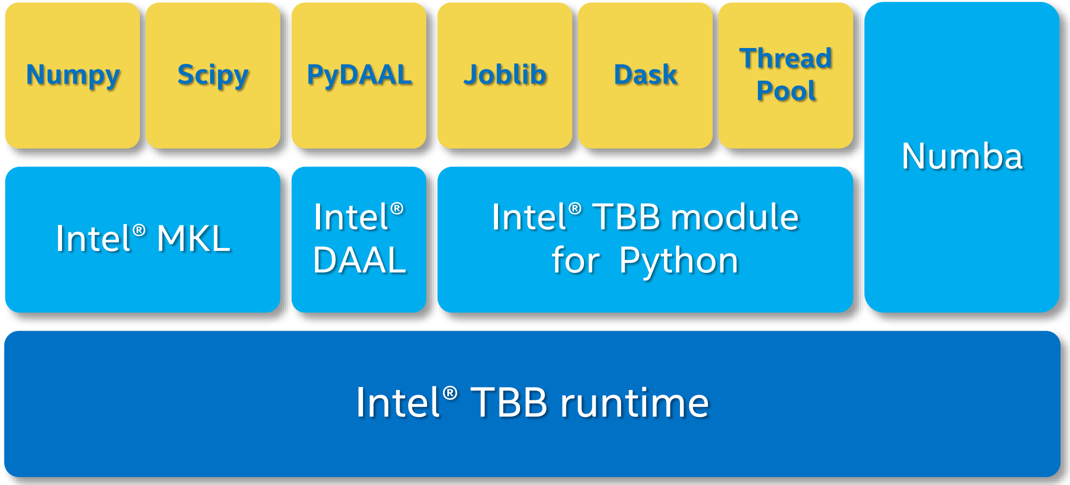
- E.g. thread-pool
- But what about the infamous global interpreter lock?
  - Can be released when calling out to C
  - Native parts can be parallel as long as they do not execute Python
    - ➢ not such a big issue with native computations
  - Limited efficiency by Amdahl's law

INTEL® TBB: PARALLELISM ORCHESTRATION

- Software components are built from smaller ones
- If each component is threaded there can be too much!
- Intel TBB dynamically balances thread loads and effectively manages oversubscription



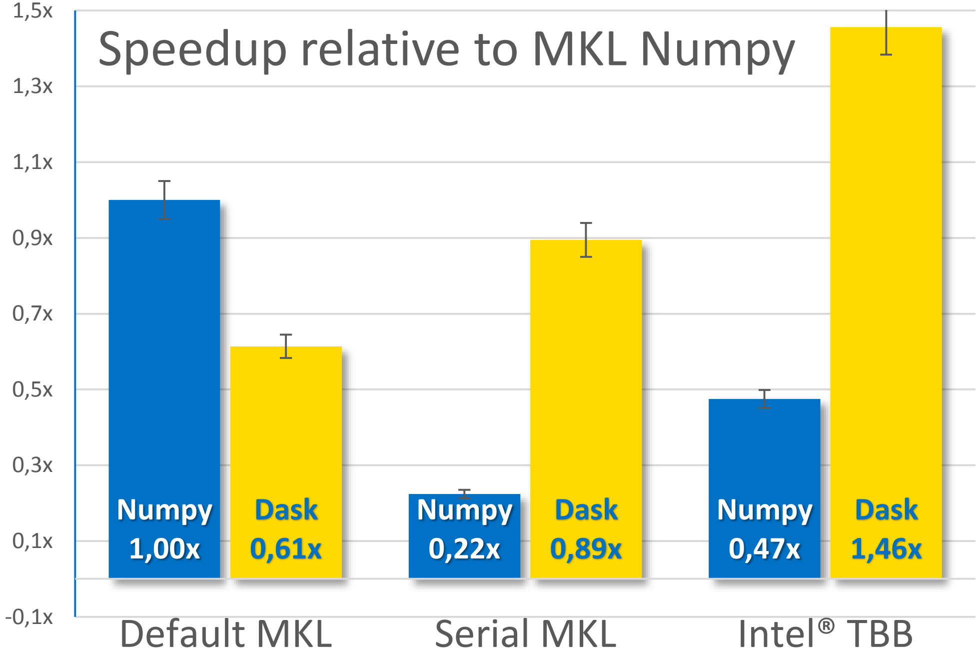

```
> python -m TBB application.py
```



EXAMPLE: NESTED PARAELLISM IN QR

```python
import time, numpy as np
x = np.random.random((100000, 2000))
t0 = time.time()
q, r = np.linalg.qr(x)
test = np.allclose(x, q.dot(r))
assert(test)
print(time.time() - t0)
```

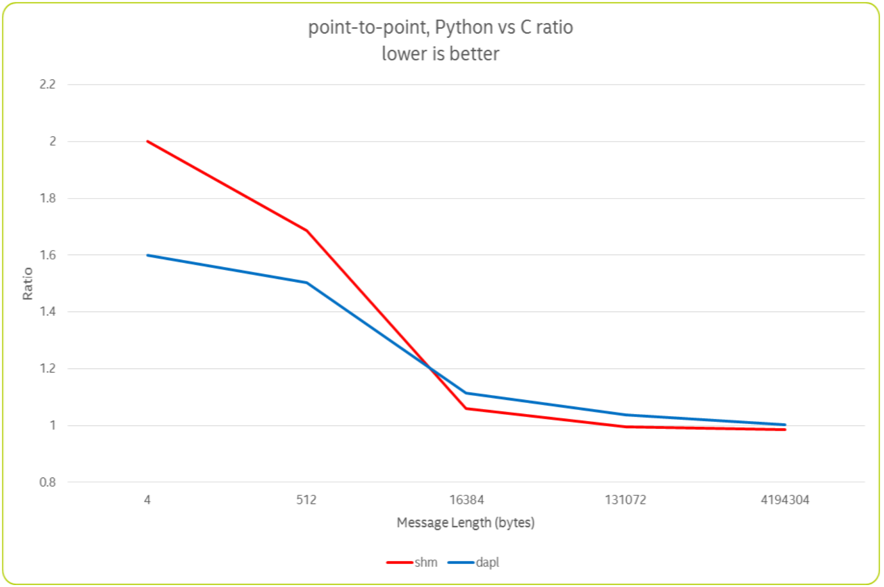
```python
import time, dask, dask.array as da
x = da.random.random((100000, 2000),
                     chunks=(10000, 2000))
t0 = time.time()
q, r = da.linalg.qr(x)
test = da.all(da.isclose(x, q.dot(r)))
assert(test.compute()) # threaded
print(time.time() - t0)
```



System info: 32x Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz, disabled HT, 64GB RAM; Intel(R) MKL 2017.0 Beta Update 1 Intel(R) 64 architecture, Intel(R) AVX2; Intel(R)TBB 4.4.4; Ubuntu 14.04.4 LTS; Dask 0.10.0; Numpy 1.11.0.

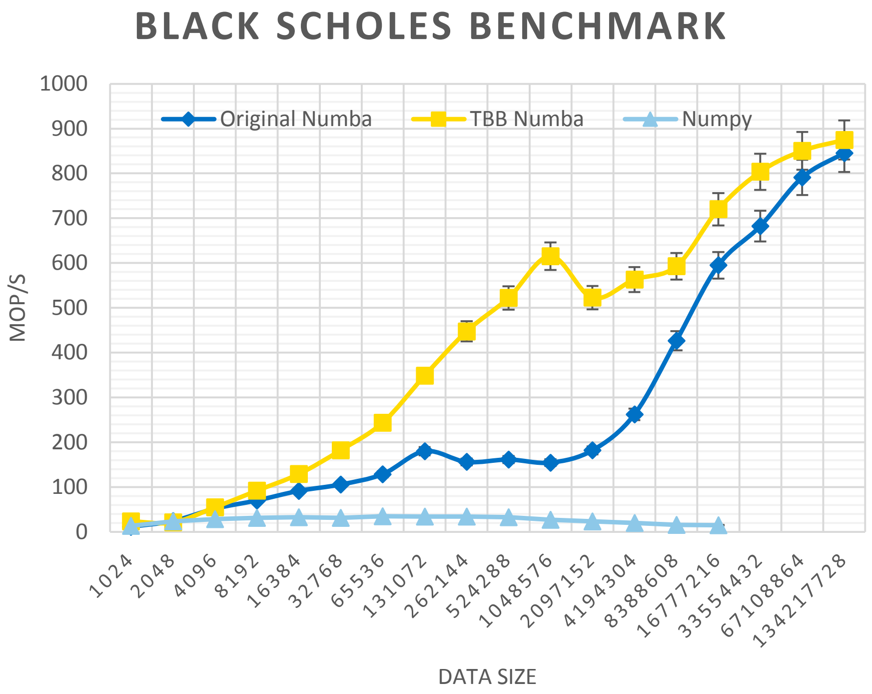
DISTRIBUTED PARALLELISM

- Intel® MPI library
  - mpi4py
  - ipyparallel
- We also support:
  - PySpark -- Python interfaces for Spark - a fast and general engine for large-scale data processing.
  - Dask -- *a flexible parallel computing library for analytic computing.*



NUMBA: JIT COMPILER FOR PYTHON

- Decorators annotate functions
- Just-in-time-compiles to native machine instructions
  - LLVM-based
  - Can parallelize and vectorize loops
- JIT API
- Math-heavy python can get close to native (C/C++) performance
  - Pure Python!



Configuration Info: - Versions: Intel(R) Distribution for Python 2.7.11 2017, Beta (Mar 04, 2016), MKL version 11.3.2 for Intel Distribution for Python 2017, Beta, Fedora* built Python*: Python 2.7.10 (default, Sep 8 2015), NumPy 1.9.2, SciPy 0.14.1, multiprocessing 0.70a1 built with gcc 5.1.1; Hardware: 96 CPUs (HT ON), 4 sockets (12 cores/socket), 1 NUMA node, Intel(R) Xeon(R) E5-4657L v2@2.40GHz, RAM 64GB, Operating System: Fedora release 23 (Twenty Three)

CYTHON: COMPILABLE PYTHON

- Optimizing source-to-source compiler
  - Python + extensions (Cython, based on Pyrex)
    - Types, GIL, parallel, etc.
  - Can parallelize loops
  - Works with any C compiler
    - Intel compiler can SIMD'ize
- Used to create a Python module which then can be imported/used

PROFILING WITH INTEL® VTUNE™ AMPLIFIER

INTEL® VTUNE™ AMPLIFIER

- Right tool for high performance application profiling at all levels
  - Function-level and line-level hotspot analysis, down to disassembly
  - Call stack analysis
  - Low overhead
  - Mixed-language, multi-threaded application analysis
  - Advanced hardware event analysis for native codes (Cython, C++, Fortran) for cache misses, branch misprediction, etc.

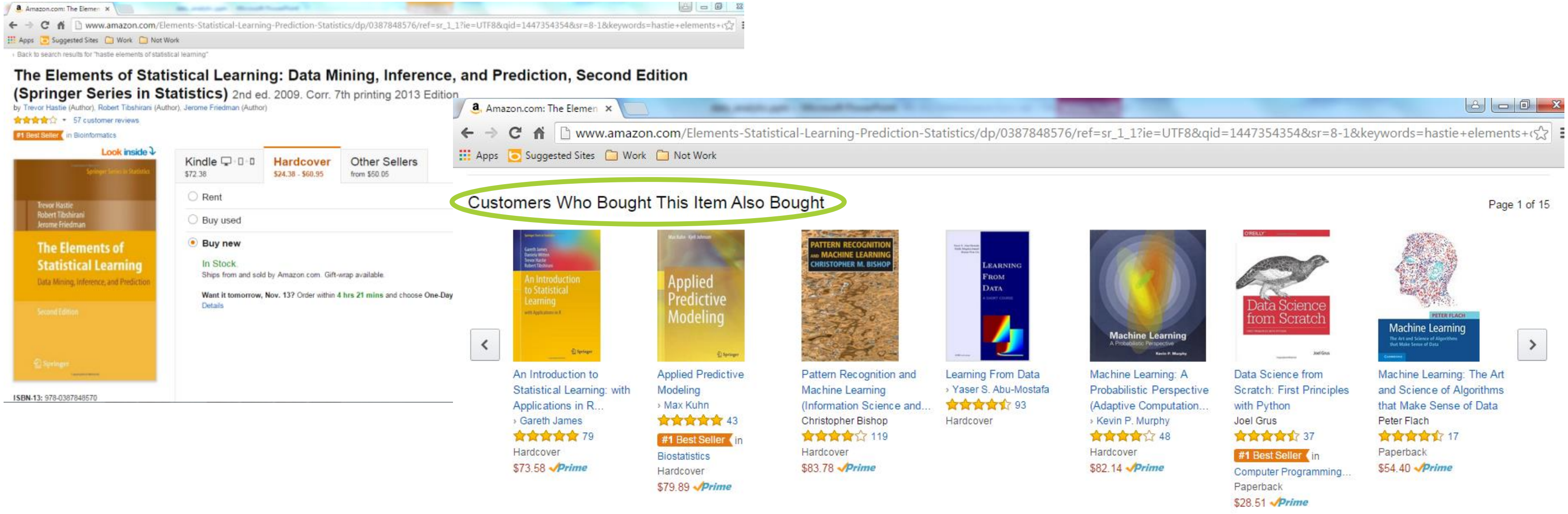
Feature	cProfile	Line_profiler	Intel® VTune™ Amplifier
Profiling technology	Event	Instrumentation	Sampling, hardware events
Analysis granularity	Function-level	Line-level	Line-level, call stack, time windows, hardware events
Intrusiveness	Medium (1.3-5x)	High (4-10x)	Low (1.05-1.3x)
Mixed language programs	Python	Python	Python, Cython, C++, Fortran

COLLABORATIVE FILTERING CASE STUDY

REAL WORLD EXAMPLE

Recommendations of useful purchases

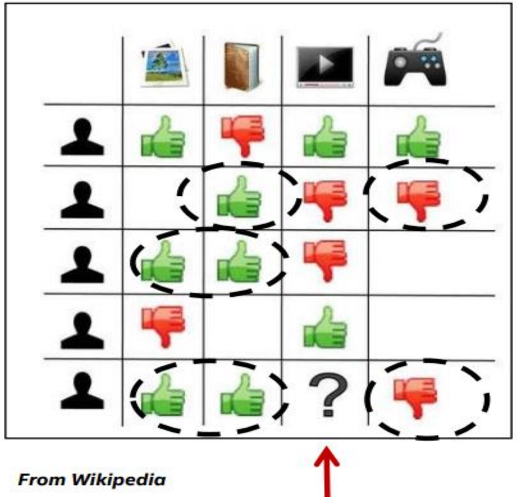
- Amazon, Netflix, Spotify,… use this all the time



COLLABORATIVE FILTERING

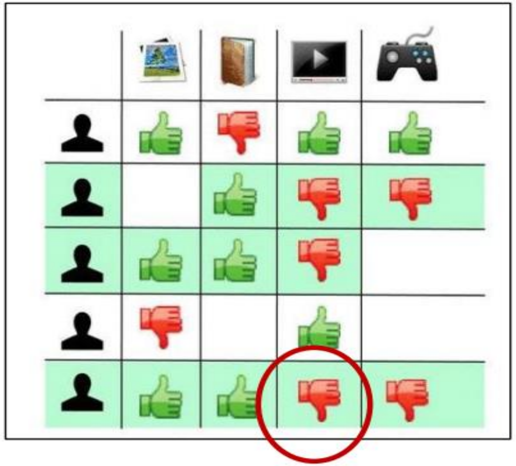
- Processes users' past behavior, their activities and ratings
- Predicts, what user might want to buy depending on his/her preferences

Collaborative Filtering



From Wikipedia

Similarities in users preferences (in Green) are used to predict ratings



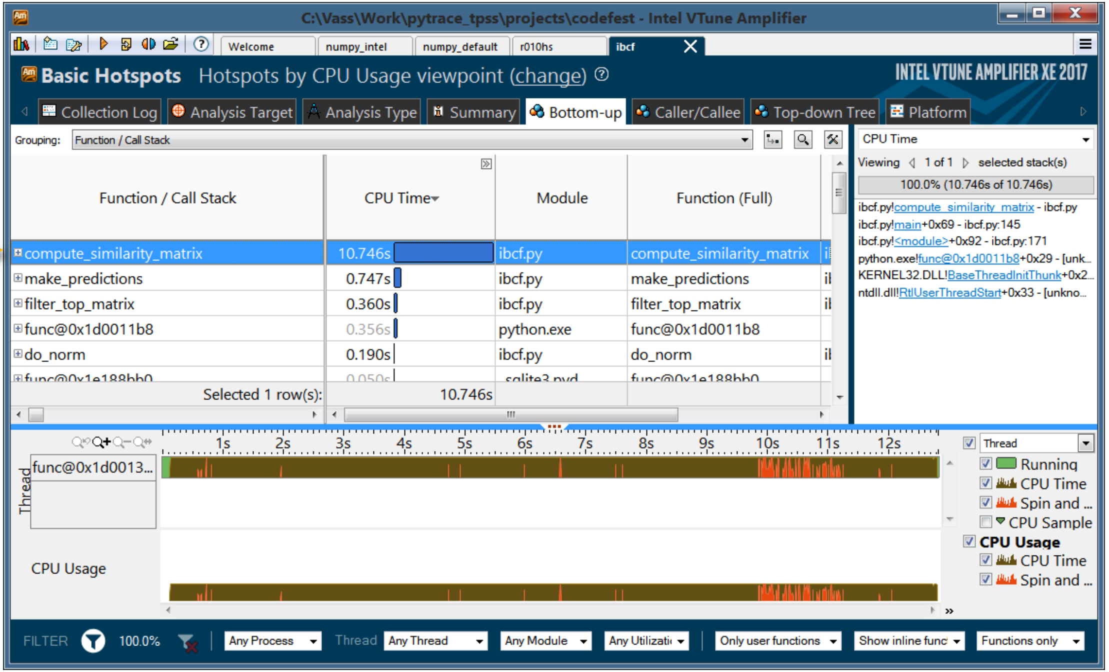
COLLABORATIVE FILTERING: TRAINING AND RECOMMENDATION

- Training
  - Reading of items and its ratings
  - Estimating item-to-item similarity
- Recommendation
  - Reading of user's ratings
  - Generating recommendations

Input data from http://grouplens.org/:
1 000 000 ratings, 6040 users, 3260 movies

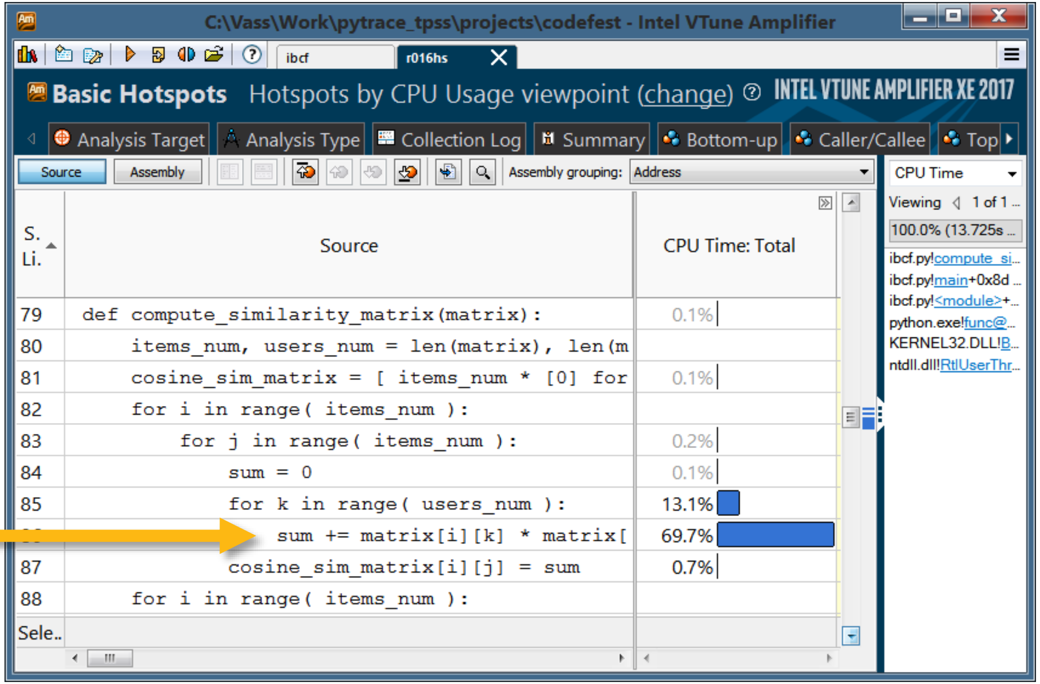
TRAINING: PROFILING PURE PYTHON

Items similarity assessment (similarity matrix computation) is the main hotspot



Configuration Info: - Versions: Red Hat Enterprise Linux* built Python*: Python 2.7.5 (default, Feb 11 2014), NumPy 1.7.1, SciPy 0.12.1, multiprocessing 0.70a1 built with gcc 4.8.2; Hardware: 24 CPUs (HT ON), 2 Sockets (6 cores/socket), 2 NUMA nodes, Intel(R) Xeon(R) X5680@3.33GHz, RAM 24GB, Operating System: Red Hat Enterprise Linux Server release 7.0 (Maipo)

TRAINING: PROFILING PURE

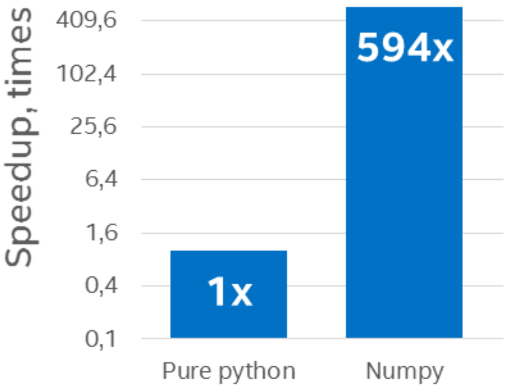


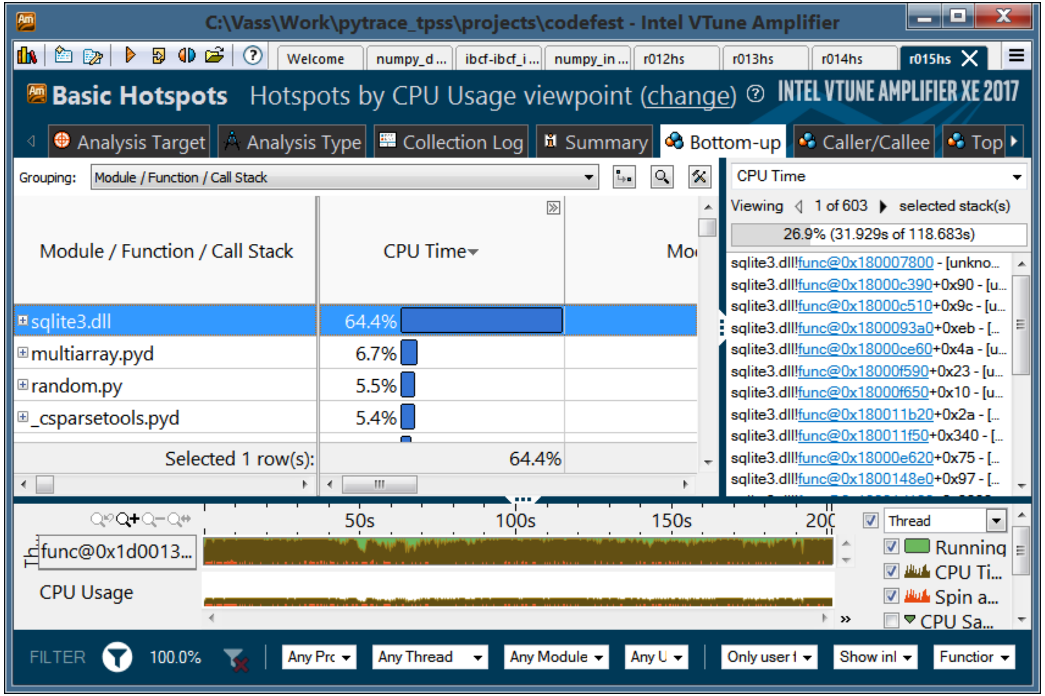
This loop is major bottleneck. Use appropriate technologies (NumPy/SciPy/Scikit-Learn or Cython/Numba) to accelerate

Configuration Info: - Versions: Red Hat Enterprise Linux* built Python*: Python 2.7.5 (default, Feb 11 2014), NumPy 1.7.1, SciPy 0.12.1, multiprocessing 0.70a1 built with gcc 4.8.2; Hardware: 24 CPUs (HT ON), 2 Sockets (6 cores/socket), 2 NUMA nodes, Intel(R) Xeon(R) X5680@3.33GHz, RAM 24GB, Operating System: Red Hat Enterprise Linux Server release 7.0 (Maipo)

TRAINING: PYTHON + NUMPY (MKL)

- Much faster!
- The most compute-intensive part takes ~5% of all the execution time





Configuration info: 96 CPUs (HT ON), 4 Sockets (12 cores/socket), 1 NUMA nodes, Intel(R) Xeon(R) E5-4657L v2@2.40GHz, RAM 64GB, Operating System: Fedora release 23 (Twenty Three)

INTEL® DISTRIBUTION FOR PYTHON*

Full scipy-stack

- numpy, scipy, matplotlib, ipython/jupyter, sympy, pandas

Plus Selected HPC/Big-Data packages

- pyDAAL, scikit-learn, mpi4py, tbb…

Python* 2.7 and 3.5

Windows*, Linux*, MacOS* (all 64bit)

Commercial support via Intel® Parallel Studio

Software