ИСПОЛЬЗОВАНИЕ INTEL® DISTRIBUTION FOR PYTHON*
**INTRODUCTION**

- Python is among the most popular programming languages
  - Especially in research/data science for prototyping
  - But very limited use in production

- Big data real time analytics requires deploying models to High Performance Data Analytics (HPDA) environments
  - Hire a team of Java/C++ programmers ...
  - OR

- Ease access to HPDA for data scientist

- This talk shows how Intel Distribution for Python may ease access to production environment for researcher/data scientist

Complementary to Intel TAP ATK

**Python** is #1 programming language in **hiring demand** followed by **Java** and **C++**.

And the demand is growing
PROBLEM STATEMENT: DATA SCIENTIST IS NOT A PROFESSIONAL PROGRAMMER

Until 2015 customer demand for using Intel tools with scripting & JIT languages was addressed by publishing Knowledge Base articles on Intel Developer Zone

“... I was hoping to take advantage of by building my NumPy and Scipy installations using the MKL ... After spending over 40 hours trying to figure it out myself, I gave up.” – Researcher, Arizona State University*

“Any KB 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.” – Anonymous Data Scientist*
Programming Languages Productivity

**Language Verbosity**

(LOC/Feature)

- Python
- Java
- C++
- C

**Programming Complexity**

(Hours)

- Python
- Java
- C++
- C

*Prechelt*

**Berkholz**

---


**Berkholz**, Programming languages ranked by expressiveness
Problem Statement: Reducing gap between Interpreted and Native Code Requires Professional Programming Skills

Chapter 19. Performance Optimization of Black Scholes Pricing

\[ V_{pu} = S_0 \cdot \text{CDF}(d_1) - e^{-rT} \cdot X \cdot \text{CDF}(d_2) \]

\[ V_{pu} = e^{-rT} \cdot X \cdot \text{CDF}(-d_2) - S_0 \cdot \text{CDF}(-d_1) \]

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

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

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.
Highlights: Intel® Distribution for Python* 2017
Focus on advancing Python performance closer to native speeds

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

- Prebuilt, accelerated Distribution for numerical & scientific computing, data analytics, HPC. Optimized for IA
- Drop in replacement for your existing Python. No code changes required

Drive performance with multiple optimization techniques

- Accelerated NumPy/SciPy/scikit-learn with Intel® Math Kernel Library
- Data analytics with pyDAAL, Enhanced thread scheduling with TBB, Jupyter* notebook interface, Numba, Cython
- Scale easily with optimized mpi4py and Jupyter notebooks

Faster access to latest optimizations for Intel architecture

- Distribution and individual optimized packages available through conda and Anaconda Cloud
- Optimizations upstreamed back to main Python trunk
Near Native Performance Speedups on IA

**Intel® Xeon® Processor**

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® Xeon Phi™ CPU 7210 1.30 GHz, 96 GB of RAM, 6 DIMMS of 16GB@1200MHz

**Intel® Xeon Phi™ Product Family**
Intel® Xeon® Processor

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 Xeon Phi™ CPU 7210 1.30 GHz, 96 GB of RAM, 6 DIMMS of 16GB@1200MHz
TASK: COLLABORATION FILTERING
Real World Example

Recommendations of useful purchases

- Amazon, Netflix, Spotify,... use this all the time
Collaboration Filtering

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

- Reading of items and its ratings
- Item-to-item similarity assessment
- Reading of user's ratings
- Generation of recommendations

Input data was taken from:
- 1,000,000 ratings.
- 6,040 users
- 3,260 movies
Pure Python (items similarity assessment)

- All the work takes \(~338\) minutes
- From that, items similarity assessment takes \(~336\) minutes (i.e. \(~99.6\%\) of all the execution time)
- The picture shows an example of profiling for 20 thousand ratings

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)

Experimental version of the product. Might differ from the final release
Why’s Interpreted Code Unfriendly To Modern HW?

Moore’s law still works and will work for at least next 10 years

We have hit limits in
- Power
- Instruction level parallelism
- Clock speed

But not in
- Transistors (more memory, bigger caches, wider SIMD, specialized HW)
- Number of cores

Flop/Byte continues growing
- 10x worse in last 20 years

Efficient software development means
- Optimizations for data locality & contiguity
- Vectorization
- Threading

Source: J.Cownie, HPC Trends and What They Mean for Me, Imperial College, Oct. 2015
Python + numpy, scipy, sklearn modules

- Now the work takes ~25 seconds
- The most compute-intensive part takes ~6-7% of all the execution time

Configuration Info: - 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)
Generation of user recommendations

- Items similarity assessment – is not the ultimate goal
- The main task – based on user experience, recommend him/her reasonable items
- The picture shows generation of the recommendations for 300000 users

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)
Speedup in Intel® Distribution for Python*

- Python and its modules are assembled by Intel® Compiler
- Numeric computations use Intel® Math Kernel Library. The picture shows that OpenMP* parallelization is used for matrix multiplication

Experimental version of the product. Might differ from the final release
But can we just use standard ThreadPool without Intel stuff?

def process_in_parallel(n, body):
    from multiprocessing.pool import ThreadPool
    global tp_pool, numthreads
    if 'tp_pool' not in globals():
        print "Creating ThreadPool(%s)" % numthreads
        tp_pool = ThreadPool(int(numthreads))
    tp_pool.map(body, xrange(n))
Generation of user recommendations

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)
### Possible Problems with Parallelism

- Applying parallelism only for the innermost loop can be inefficient: scalability
  - Over-synchronized: overheads become visible if there is not enough work inside
  - Over-utilization: distribution to the whole machine can be inefficient
  - Amdahl law: serial regions limit scalability of the whole program
- Applying parallelism on the outermost level only:
  - Under-utilization: it does not scale if there is not enough tasks or/and load imbalance
  - Provokes oversubscription if nested level is threaded independently & unconditionally
- Frameworks can be used from both levels
  - To parallel or not to parallel? That is the question
Intel® Threading Building Blocks (Intel® TBB) is a production C++ library that simplifies threading for performance.

- Lets you manage parallelism, not threads.
- Works with off-the-shelf C++ compilers.
- Proven to be portable to new compilers, operating systems, and architectures.
- Free C++ open source library (commercial licensing is also available)
Intel® Threading Building Blocks (Intel® TBB) + TBB module for Python

- Intel® Threading Building Blocks (Intel® TBB)
  - Free C++ open source library
  - Efficient nested parallelism
  - Can be obtained from

- In order to use experimental version of the Python wrapper:

  $ python -m TBB <your>.py
Generation of user recommendations (dense data)

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)
Generation of user recommendations (sparse data)

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)
Do not believe us on our bare word, try it yourself!

• Intel® Distribution For Python and Intel® VTune™ Amplifier For Python will be available in Parallel Studio XE 2017 Beta in April:
  •
  •
  • Accelerated packages will be available at Intel channel in Conda:
  •
• Intel® Threading Building Blocks (Intel® TBB):
  •
• Take a look at Intel® Developer Zone –:
  • To see other products
  • To find out, in what circumstances the free of charge versions are available
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.

- Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, 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