FUEL YOUR INSIGHT
Scaling Out Python* To HPC and Big Data

Sergey Maidanov
Software Engineering Manager for Intel® Distribution for Python*
What Problems We Solve: Scalable Performance

Make Python usable beyond prototyping environment by scaling out to HPC and Big Data environments.
What Problems We Solve: 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
Why Yet Another Python Distribution?

Mature AVX2 instructions based product

Intel® Xeon® Processors

Python* Performance as a Percentage of C/Intel® MKL for Intel® Xeon® Processors, 32 Core (Higher is Better)

New AVX512 instructions based product

Intel® Xeon Phi™ Product Family

Python* Performance as a Percentage of C/Intel® MKL for Intel® Xeon Phi™ Product Family, 64 Core (Higher is Better)

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

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.
Scaling To HPC & Big Data Environments

- 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/]
Efficiency = Parallelism and Python?

• CPython as interpreter inhibits parallelism but...
• ... Overall Python tools evolved far toward unlocking parallelism

- Native extensions
  - *numpy*, *scipy*, *scikit-learn* accelerated with
    - *Intel® MKL*, *Intel® DAAL*, *Intel® IPP*
  - Composable multi-threading with *Intel® TBB* and *Dask* *
  - 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
  - *(pySpark*, *Theano*, *TensorFlow*, etc.)
- Mixed language profiling with *Intel® VTune™ Amplifier*
Composable Multi-Threaded
With Intel® TBB

• Amhdal’s law suggests extracting parallelism at all levels
• If software components do not coordinate on threads use it may lead to oversubscription
• Intel TBB dynamically balances HW thread loads and effectively manages oversubscription
• Intel engineers extended Cpython* and Numba* thread pools with support of Intel® TBB

>python -m TBB myapp.py
Composable Multi-Threading Example: Batch QR Performance

```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)
```

Speedup relative to Default Numpy*

Intel® MKL, OpenMP* threading
Intel® MKL, Serial
Intel® MKL, Intel® TBB threading

Over-subscription

App-level parallelism only

Intel® MKL
Intel® MKL

Dask
Dask

Numpy
Numpy

0.0x
0.2x
0.4x
0.6x
0.8x
1.0x
1.2x
1.4x

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.

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.
NumPy* and SciPy* Continuous Improvement

- MKL-level performance for dense linear algebra and FFT
- NumPy random, umath & NumExpr* exploit SIMD and multi-threading out-of-the-box
- Better memory allocation and copy in NumPy

Black Scholes Formula

NumPy* umath scalability

Does not scale beyond 16K options 😞

```
def black_scholes (mopt, 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 = 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 = P * d1 - Se * d2
    put = call - P + Se

    return call, put
```
NumExpr* Scalability: Black-Scholes Formula

Black-Scholes Formula with NumExpr*

```
1 import base_bs_erf
2 import numexpr as ne
3
4 def black_scholes ( nopt, price, strike, t, rate, vol ):
5     nr = -rate
6     sigSigTwo = vol * vol * 2
7     
8     P = price
9     S = strike
10    T = t
11
12     call = ne.evaluate("P * (0.5 + 0.5 * erf((log(P / S) - T * nr) / (0.25 * T * sigSigTwo) * 1/sqrt(T * sigSigTwo))) - S * exp(T * nr)" + "((0.5 + 0.5 * erf((log(P / S) - T * nr - 0.25 * T * sigSigTwo) * 1/sqrt(T * sigSigTwo))))")
13     put = ne.evaluate("call - P + S * exp(T * nr) ")
14
15     return call, put
```

What’s a magic?
Cython* Scalability: Black-Scholes Formula

Black-Scholes Formula: NumExpr* vs. Cython*

75% of native C
**Skt-Learn*** Optimizations With Intel® MKL... And Intel® DAAL

Intel® Distribution for Python* ships Intel® Data Analytics Acceleration Library with Python interfaces, a.k.a. pyDAAL

---

**Speedups of Scikit-Learn Benchmarks**

Intel® Distribution for Python* 2017 Update 1 vs. system Python & NumPy/Scikit-Learn

---

**Effect of Intel MKL optimizations for NumPy* and SciPy***

---

**Potential Speedup of Scikit-learn* due to PyDAAL**

PCA, 1M Samples, 200 Features

---

**Effect of DAAL optimizations for Scikit-Learn***

---

System info: 32x Intel® Xeon® CPU E5-2698 v3 @ 2.30GHz, disabled HT, 64GB RAM; Intel® Distribution for Python* 2017 Gold; Intel® MKL 2017.0.0; Ubuntu 14.04.4 LTS; Numpy 1.11.1; scikit-learn 0.17.1.

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.
Ideas Behind Intel® DAAL: Heterogeneous Analytics

- Data is different, data analytics pipeline is the same
- Data transfer between devices is costly, protocols are different
  - Need data analysis proximity to Data Source
  - Need data analysis proximity to Client
  - Data Source device ≠ Client device
  - Requires abstraction from communication protocols
### Ideas Behind Intel® DAAL: Effective Data Management, Streaming and Distributed Processing

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

#### Chart

- **Features, \( p \)**
  - Numeric
  - Categorical
  - Blank/Missing
  - Outlier

- **Observations, \( n \)**
  - Time

- **Memory Capacity**

- **Data Types**
  - Categorical
  - Blank/missing
  - Non-homogeneous
  - Sparse/missing/noisy
Ideas Behind Intel® DAAL: 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
Ideas Behind Intel® DAAL: Languages & Platforms

DAAL has multiple programming language bindings

- C++ – ultimate performance for real-time analytics with DAAL
- Java*/Scala* – easy integration with Big Data platforms (Hadoop*, Spark*, etc)
- Python* – advanced analytics for data scientist
Ex 1. Compute Basic Statistics While Reading Dataset

Ex 2. Offline vs. Online

- $\text{cov}(X, Y) = E((X - E(X))(Y - E(Y)))$
- Algorithm: Maximum Likelihood Estimator
- Dataset: CSV file $n=1\text{M}, p=100$

Better memory management in pyDAAL allows bigger datasets even for offline processing
- Ex.: Scikit-learn fails to process $n=50\text{M}, p=1000$ while pyDAAL successfully completes work

Online processing removes dataset size limitation

System Info: Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, 504GB, 2x24 cores, HT=on, OS RH7.2 x86_64, Intel Distribution for Python 2017 Update 1 (Python 3.5)

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.
Ex 3: Read vs. Compute

- Algorithm: SVM Classification with RBF kernel
- Training dataset: CSV file (PCA-preprocessed MNIST, 40 principal components) \(n=42000, p=40\)
- Testing dataset: CSV file (PCA-preprocessed MNIST, 40 principal components) \(n=28000, p=40\)

System Info:
Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, 504GB, 2x24 cores, HT=on, OS RH7.2 x86_64, Intel Distribution for Python 2017 Update 1 (Python 3.5)

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.

60% faster CVS file read, 2.2x faster training in pyDAAL. Read and compute balanced.
Distributed Parallelism

- Intel® MPI* accelerates Intel® Distribution for Python (mpi4py*, ipyparallel*)
- Intel Distribution for Python also supports
  - PySpark* - Python* interfaces for Spark*, an engine for large-scale data processing
  - Dask* - flexible parallel computing library for numerical computing

Configuration Info: Hardware (each node): Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz, 2x18 cores, HT is ON, RAM 128GB; Versions: Oracle Linux Server 6.6, Intel® DAAL 2017 Gold, Intel® MPI 5.1.3; Interconnect: 1 GB Ethernet.

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.
Summary And Call To Action

• Intel created the Python* distribution for out-of-the-box performance and scalability on Intel® Architecture
  – With minimum to no code modification Python aims to scale
• Multiple technologies applied to unlock parallelism at all levels
  – Numerical libraries, libraries for parallelism, Python code compilation/JITing, profiling
  – Enhancing mature Python packages and bringing new technologies, e.g. pyDAAL, TBB
• With multiple choices available Python developer needs to be conscious what will scale best
  – Intel® VTune™ Amplifier helps making conscious decisions

Intel Distribution for Python is free!
Commercial support included for Intel® Parallel Studio XE customers!
Easy to install with Anaconda* https://anaconda.org/intel/