INTEL® HPC DEVELOPER CONFERENCE
FUEL YOUR INSIGHT
Python Scalability Story
In Production Environments

Sergey Maidanov
Software Engineering Manager for Intel® Distribution for Python*

Stanley Seibert
Director of Community Innovation
Continuum Analytics
Motivation: Why Python

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

And the demand is growing.
What Problems We Solve: Scalable Performance

Make Python usable beyond prototyping environment by scaling out to HPC and Big Data environments.
“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
INTEL® DISTRIBUTION FOR PYTHON* 2017
Advancing Python performance closer to native speeds

<table>
<thead>
<tr>
<th>Easy, out-of-the-box access to high performance Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Prebuilt, optimized for numerical computing, data analytics, HPC</td>
</tr>
<tr>
<td>• Drop in replacement for existing Python. No code changes required</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance with multiple optimization techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Accelerated NumPy/SciPy/Scikit-Learn with Intel® MKL</td>
</tr>
<tr>
<td>• Data analytics with pyDAAL, enhanced thread scheduling with TBB, Jupyter* Notebook interface, Numba, Cython</td>
</tr>
<tr>
<td>• Scale easily with optimized MPI4Py and Jupyter notebooks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Faster access to latest optimizations for Intel architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Distribution and individual optimized packages available through conda and Anaconda Cloud: anaconda.org/intel</td>
</tr>
<tr>
<td>• Optimizations upstreamed back to main Python trunk</td>
</tr>
</tbody>
</table>
Why Yet Another Python Distribution?

Mature AVX2 instructions based product

Intel® Xeon® Processors

![Graph showing 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

![Graph showing 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 Production Environment

- 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*

Intel® Advanced Vector Extensions

Evolution of Multicore

* Roofline Performance Model https://crd.lbl.gov/departments/computer-science/PAR/research/roofline/
Efficiency = Parallelism in 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
Numpy* & Scipy* optimizations with Intel® MKL

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

Linear Algebra
- BLAS
- LAPACK
- ScaLAPACK
- Sparse BLAS
- Sparse Solvers
  - Iterative
  - PARDISO* SMP & Cluster

Fast Fourier Transforms
- 1D and multidimensional FFT

Vector Math
- Trigonometric
- Hyperbolic
- Exponential
- Log
- Power
- Root

Vector RNGs
- Multiple BRNG
- Support methods for independent streams creation
- Support all key probability distributions

Summary Statistics
- Kurtosis
- Variation coefficient
- Order statistics
- Min/max
- Variance-covariance

And More
- Splines
- Interpolation
- Trust Region
- Fast Poisson Solver

INTEL® HPC DEVELOPER CONFERENCE

Functional domain in this color accelerate respective NumPy, SciPy, etc. domain
Scikit-Learn* optimizations with Intel® MKL

**Speedups of Scikit-Learn Benchmarks**

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

<table>
<thead>
<tr>
<th>Application</th>
<th>Intel SKlearn</th>
<th>System Sklearn</th>
<th>Intel PyDAAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCA, 1M Samples, 200 Features</td>
<td>1.11</td>
<td>1</td>
<td>54.13</td>
</tr>
</tbody>
</table>

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

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

PCA, 1M Samples, 200 Features

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

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
Distributed parallelism

Intel® MPI library accelerates Intel® Distribution for Python* (Mpi4py*, Ipyparallel*)

Intel Distribution for Python* also supports

- PySpark* - Python interfaces for Spark*, a fast and general engine for large-scale data processing.
- Dask* - a flexible parallel computing library for analytic computing.

PyDAAL Implicit ALS with Mpi4Py*

Configuration Info: 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.
Composable multi-threading with Intel® TBB

• Amhdal’s law suggests extracting parallelism at all levels
• 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 myapp.py
Composable Parallelism: QR Performance

```
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)
```

```
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**
  - Numpy: 1.00x
  - Dask: 0.61x
  - Intel® MKL, Serial
  - Numpy: 0.22x
  - Dask: 0.89x
  - Intel® MKL, Intel® TBB threading
  - Numpy: 0.47x
  - Dask: 1.46x

*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.
Profiling Python* code with 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.

<table>
<thead>
<tr>
<th>Feature</th>
<th>cProfile</th>
<th>Line_profiler</th>
<th>Intel® VTune™ Amplifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profiling technology</td>
<td>Event</td>
<td>Instrumentation</td>
<td>Sampling, hardware events</td>
</tr>
<tr>
<td>Analysis granularity</td>
<td>Function-level</td>
<td>Line-level</td>
<td>Line-level, call stack, time windows, hardware events</td>
</tr>
<tr>
<td>Intrusiveness</td>
<td>Medium (1.3-5x)</td>
<td>High (4-10x)</td>
<td>Low (1.05-1.3x)</td>
</tr>
<tr>
<td>Mixed language programs</td>
<td>Python</td>
<td>Python</td>
<td>Python, Cython, C++, Fortran</td>
</tr>
</tbody>
</table>
Scaling Python with JIT Compilation

Stanley Seibert
Director of Community Innovation
Continuum Analytics
November 2016
Creating a Compiler For Python

Many valid approaches, but we think these are the most important for data science:

▪ **Cannot replace the standard interpreter**
  – Must be able to continue to use pandas, SciPy, scikit-learn, etc

▪ **Minimize boilerplate**
  – Traditional compiled Python extensions require a lot of infrastructure. Try to stay simple and get out of the way.

▪ **Be flexible about execution model**
  – Not all hardware is a general purpose CPU

▪ **Integrate well with Python’s adaptable ecosystem**
  – Must be able to continue to use pandas, SciPy, scikit-learn, etc
Numba: A JIT Compiler for Python Functions

- An open-source, function-at-a-time compiler library for Python
- Compiler toolbox for different targets and execution models:
  - single-threaded CPU, multi-threaded CPU, GPU
  - regular functions, “universal functions” (array functions), GPU kernels
- Speedup: \(2x\) (compared to basic NumPy code) to \(200x\) (compared to pure Python)
- Combine ease of writing Python with speeds approaching FORTRAN
- Empowers data scientists who make tools for themselves and other data scientists
How does Numba work?

```
@jit
def do_math(a, b):
    ...

>>> do_math(x, y)
```

![Diagram of Numba workflow]

- Python Function (bytecode)
- Bytecode Analysis
- Functions Arguments
- Numba IR
- Type Inference
- Rewrite IR
- Lowering
- LLVM/NVVM JIT
- LLVM IR
- Machine Code
- Execute!
- Cache
## Supported Platforms and Hardware

<table>
<thead>
<tr>
<th>OS</th>
<th>HW</th>
<th>SW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows (7 and later)</td>
<td>32 and 64-bit x86 CPUs</td>
<td>Python 2 and 3</td>
</tr>
<tr>
<td>OS X (10.9 and later)</td>
<td>CUDA &amp; HSA Capable GPUs</td>
<td>NumPy 1.7 through 1.11</td>
</tr>
<tr>
<td>Linux (RHEL 5 and later)</td>
<td>Experimental support for ARM, Xeon Phi,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AMD Fiji GPUs</td>
<td></td>
</tr>
</tbody>
</table>
Basic Example

```python
@jit(nopython=True)
def nan_compact(x):
    out = np.empty_like(x)
    out_index = 0
    for element in x:
        if not np.isnan(element):
            out[out_index] = element
            out_index += 1
    return out[:out_index]
```

```python
In [88]:
a = np.random.uniform(size=10000)
a[a < 0.2] = np.nan
np.testing.assert_equal(nan_compact(a), a[~np.isnan(a)])
```

```python
In [89]:
%timeit a[~np.isnan(a)]
%timeit nan_compact(a)
```

10000 loops, best of 3: 52 µs per loop
100000 loops, best of 3: 19.6 µs per loop
Basic Example

In [87]: @jit(nopython=True)
def nan_compact(x):
    out = np.empty_like(x)
    out_index = 0
    for element in x:
        if not np.isnan(element):
            out[out_index] = element
            out_index += 1
    return out[:out_index]

In [88]: a = np.random.uniform(size=10000)
a[a < 0.2] = np.nan
np.testing.assert_equal(nan_compact(a), a[~np.isnan(a)])

In [89]: %timeit a[~np.isnan(a)]
%timeit nan_compact(a)

10000 loops, best of 3: 52 µs per loop
100000 loops, best of 3: 19.6 µs per loop

2.7x speedup!
Releasing the GIL

In [14]:

```python
SQRW_2PI = np.sqrt(2 * np.pi)
@numba.jit(nogil=True)
def gaussian(x, mu, sigma):
    return np.exp(0.5 * ((x - mu)**2) / sigma) / (sigma * SQRW_2PI)

print(gaussian(0.5, 1.5, 1.0))
```

```
0.657744623479
```

Option to release the GIL

Using Python concurrent.futures

Speedup Ratio

Number of Threads

0. 0.9 1.8 2.6 3.5
0 2 4
Universal Functions (Ufuncs)

Ufuncs are a core concept in NumPy for array-oriented computing.

- A function with scalar inputs is broadcast across the elements of the input arrays:
  - `np.add([1,2,3], 3) == [4, 5, 6]`
  - `np.add([1,2,3], [10, 20, 30]) == [11, 22, 33]`

- Parallelism is present, by construction. Numba will generate loops and can automatically multi-thread if requested.

- Before Numba, creating fast ufuncs required writing C. No longer!
Universal Functions (Ufuncs)

In [13]:
```python
@numba.vectorize
def response(v, gamma):
    if v < 0:
        return 0.0
    elif v < 1:
        return v ** gamma
    else:
        return v
```

Different decorator!

In [14]:
```python
x = np.linspace(-1, 2, 10000)
gamma = 1.7
%timeit np.piecewise(x, [x < 0, x >= 1],[0.0, lambda x: x, lambda x: x**gamma])
%timeit response(x, gamma)
```

1000 loops, best of 3: 244 µs per loop
The slowest run took 411.51 times longer than the fastest. This could mean that an intermedia
te result is being cached.
10000 loops, best of 3: 136 µs per loop

1.8x speedup!
Multi-threaded Ufuncs

In [12]:

```python
SQRT_2 = np.sqrt(2)
@
m numa.vectorize('float64(float64, float64, float64)', target='parallel')
def gaussian_cdf_parallel(x, mu, sigma):
    return 0.5 * (1 + math.erf((x - mu) / (sigma * SQRT_2)))
```

Specify type signature

Select parallel target

Automatically uses all CPU cores!
Distributed Computing
Example: Dask

Dask Client (Haswell) → Dask Scheduler → Dask Worker (Skylake) → Dask Worker (Skylake) → Dask Worker (Knight's Landing)

@jit
def f(x):
  ...

- Serialize with pickle module
- Works with Dask and Spark (and others)
- Automatic recompilation for each target

```
import dask

@jit
def f(x):
    ...
```
Other Numba Features

- Detects CPU model during code generation and instructs LLVM to optimize for that architecture.
- Automatic dispatch to multiple type-specialized implementations of the same function
- Uses LLVM auto-vectorization optimization passes for SIMD code generation
- Supports calls directly to C with CFFI and ctypes
- Optional caching of compiled functions to disk
- Ahead of time compilation to shared libraries
- Extension API allowing 3rd parties to extend the compiler with new data types and functions.
Conclusion

- Numba - Create new high performance functions on-the-fly with pure Python
- Understands NumPy arrays and many NumPy operations
- Supplies several compilation modes and options for multi-threading
- Use with your favorite distributed computing framework

- For more information: http://numba.pydata.org
- Comes with Anaconda: https://www.continuum.io/downloads
Call To Action
Intel is working with community leaders like Continuum Analytics to bring the BEST performance on IA to Python developers

• Start with either Intel’s or Continuum’s distribution
  • Both have Intel performance goodness baked in!
  • You cannot go wrong either way!
• Give Numba* a try and see performance increase
• Try Python* performance profiling with Intel® VTune™ Amplifier!

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

Stan Seibert
stan.seibert@continuum.io
www.intel.com/hpcdevcon

Sergey Maidanov
sergey.Maidanov@intel.com
www.intel.com/hpcdevcon

Intel® Distribution for Python*
Powered by Anaconda®
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 about compiler optimizations, see our Optimization Notice at https://software.intel.com/en-us/articles/optimization-notice#opt-en.

Copyright © 2016, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.