HIGH PERFORMANCE PYTHON OFFLOADING TO THE INTEL® XEON PHI™ (CO)PROCESSOR

Dr.-Ing. Michael Klemm
Senior Application Engineer
Software and Services Group

Legal Disclaimer & Optimization Notice

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

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

Python* in HPC

Python has gained a lot of interest throughout the HPC community (and others):

- Jupyter (formely IPython)
- Atomic Simulation Environment
- Numpy / SciPy
- Pandas
- … and many, many more

Be Faster with the Intel® Distribution for Python*

Intel® Distribution for Python*

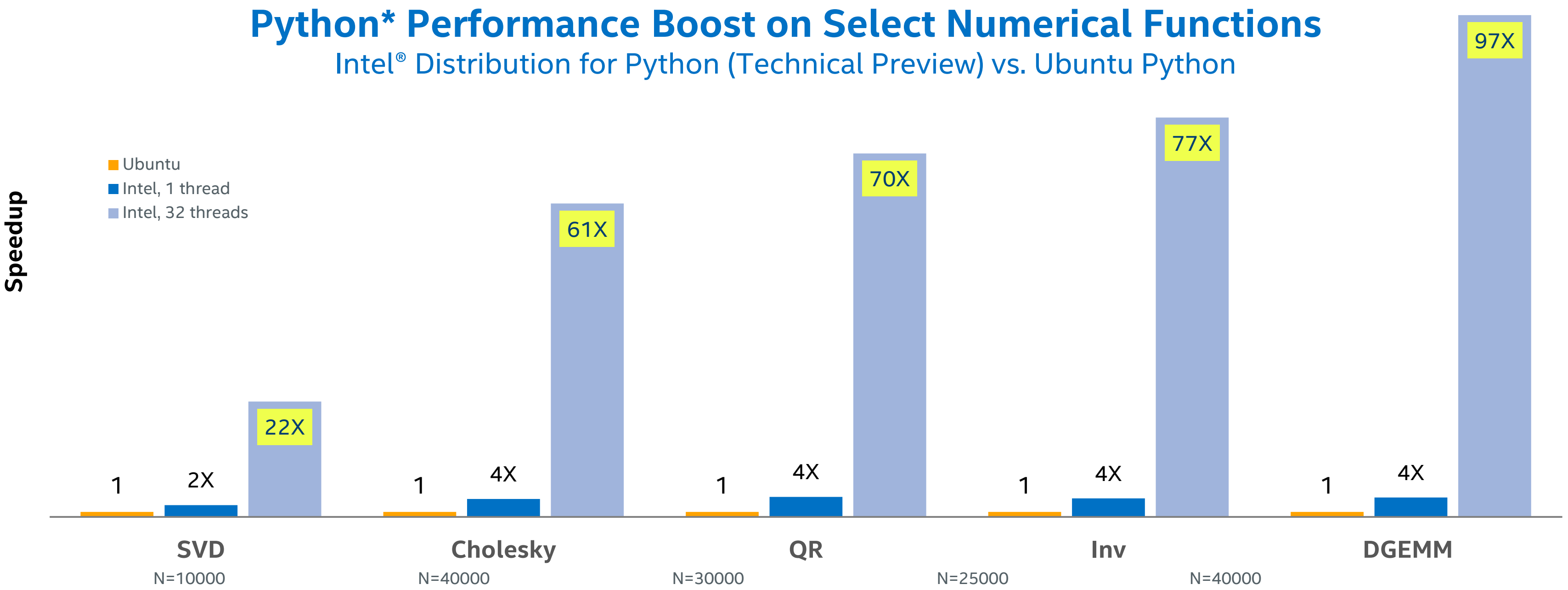
- NumPy/SciPy performance accelerated with Intel® Math Kernel Library
  - ~3x speedups on single thread#
  - Significant performance gains on multiple threads#
    (dgemm showed 97x speed-up)
- Easy installation
- Python 2.7 & 3.5
- Windows & Linux

Coming soon…

- Support for Mac OS

Performance Benchmarks on next slide

Be Faster with the Intel® Distribution for Python* /2



Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests 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. System configuration: Intel® Xeon® CPU E5-2698v3 with 2.30 GHz (2 sockets, 16 cores each, Intel® Hyper-Threading Technology off), 64 GB main memory in 8 DIMMs with 2133 MHz; operating system Ubuntu 14.04 LTS; Intel® Distribution for Python* 2.7.10 Technical Preview 1 (August 3, 2015); Ubuntu* built Python*: Python 2.7.10, NumPy 1.9.2 built with GCC 4.8.4.

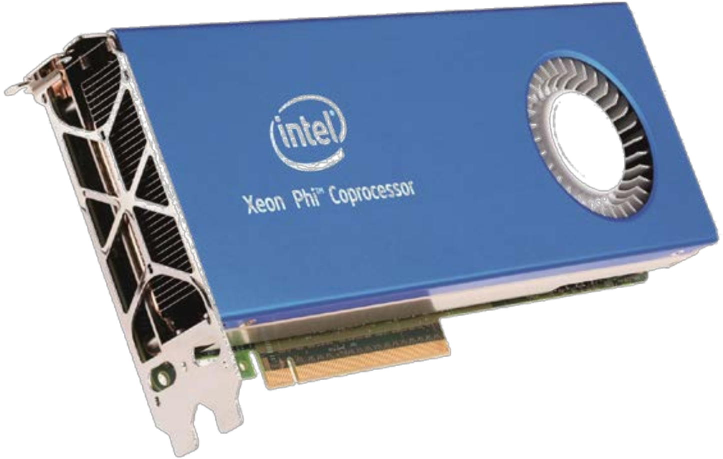
Agenda

- Quick Introduction to the Intel® Xeon Phi™ Coprocessor
- Finding Offload Candidates
- Using the Python Offload Module

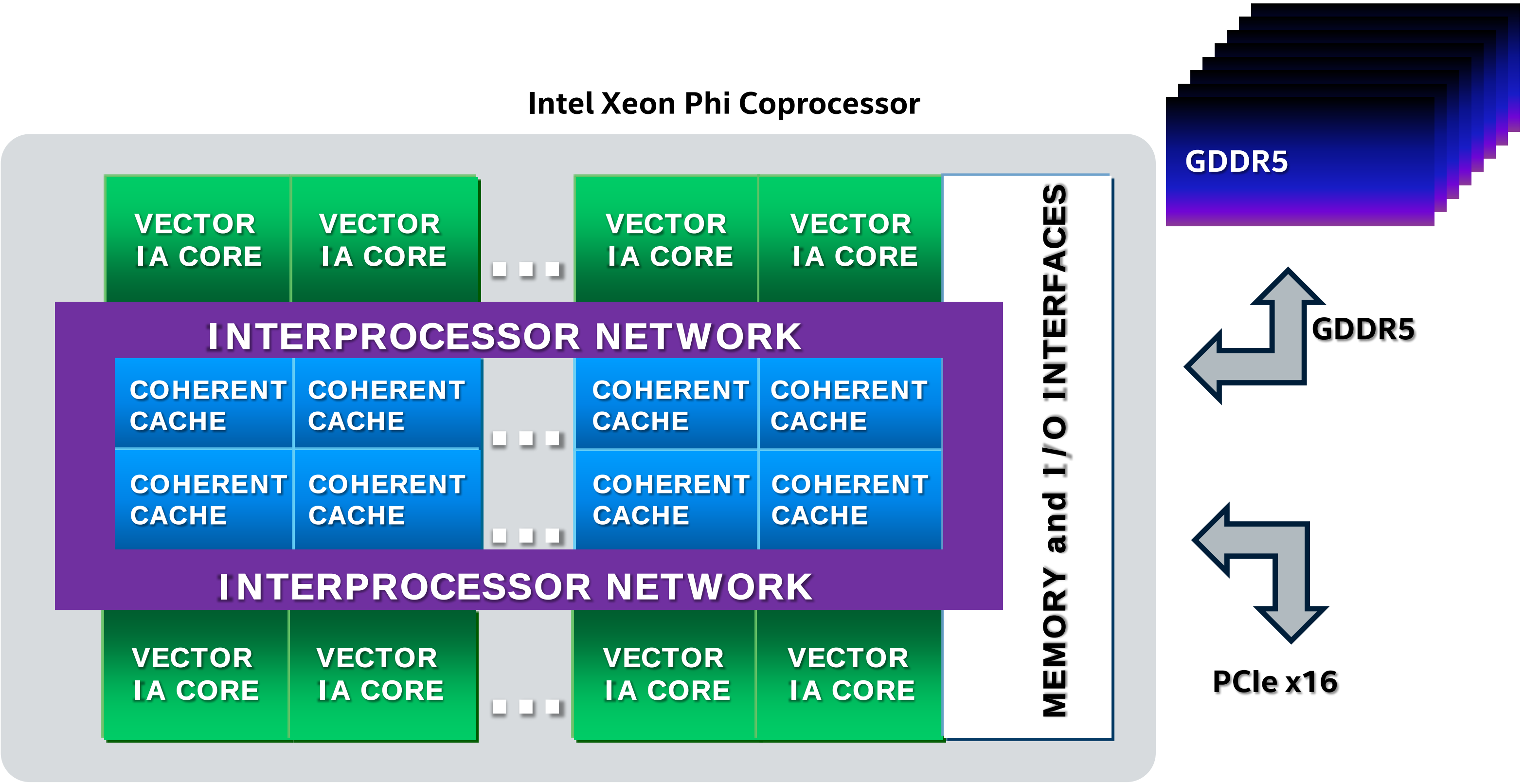
Agenda

- Quick Introduction to the Intel® Xeon Phi™ Coprocessor
- Finding Offload Candidates
- Using the Python Offload Module

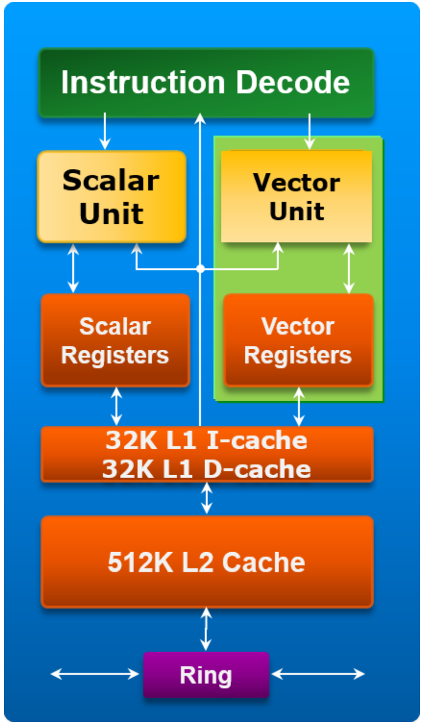
Intel® Xeon Phi™ Coprocessor



Intel® Xeon Phi™ Coprocessor /2



Intel® Xeon Phi™ Coprocessor /3



Up to 61 in-order cores w/ 4 hardware threads

Two pipelines

- Pentium® processor family-based scalar units
- Fully-coherent L1 and L2 caches
- 64-bit addressing

Vector unit

- 512-bit SIMD Instructions
- 32 512-bit wide vector registers (8x DP or 16x SP each)
- Pipelined one-per-clock throughput
- Dual issue with scalar instructions

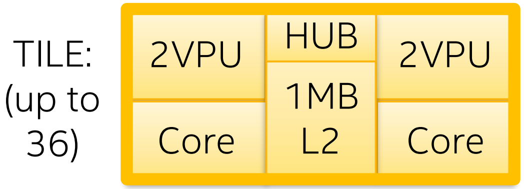
Intel® Xeon Phi™ Processor

ISA
Intel® Xeon® Processor Binary-Compatible (w/Broadwell)

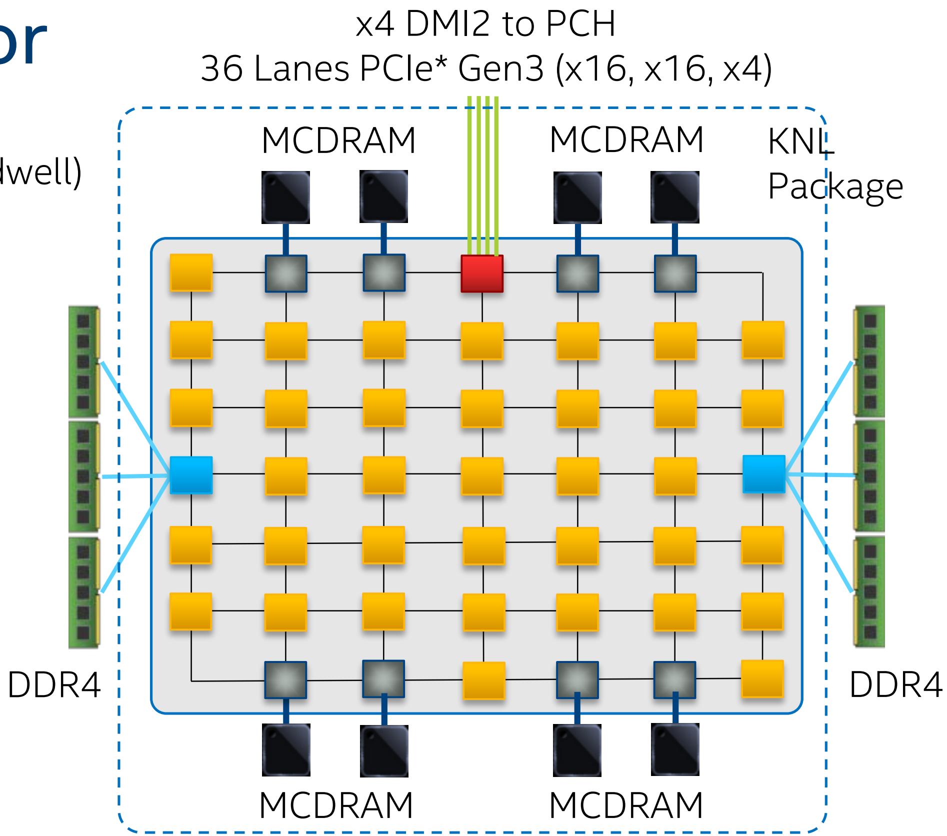
On-package memory
Up to 16GB, ~400 GB/s STREAM

Platform Memory
Up to 384GB (6ch DDR4-2400 MHz, 100 GB/sec)

Fixed Bottlenecks
- ✓ 2D Mesh Architecture
- ✓ Out-of-Order Cores
- ✓ 3x single-thread vs. KNC



Enhanced Intel® Atom™ cores based on Silvermont™ Microarchitecture



Tile | EDC (embedded DRAM controller) | IMC (integrated memory controller) | IIO (integrated I/O controller)

Intel® Xeon Phi™ Processor /2

Out-of-order core w/ 4 SMT threads

VPU tightly integrated with core pipeline

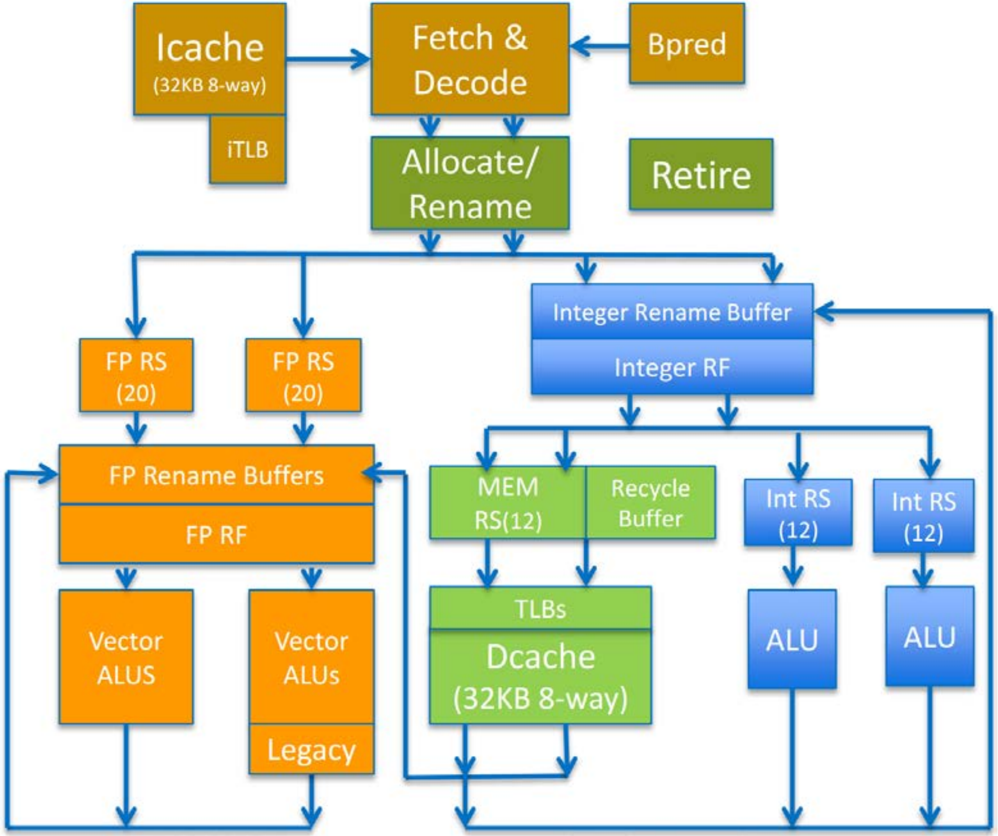
2-wide decode/rename/retire

2x 64B load & 1 64B store port for D$

L1 prefetcher and L2 prefetcher

Fast unaligned and cache-line split support

Fast gather/scatter support



Agenda

- Quick Introduction to the Intel® Xeon Phi™ Coprocessor
- Finding Offload Candidates
- Using the Python Offload Module

Offload Candidates

Offload candidates are regions of code that are amenable for offloading.

- Requirements for offload candidates
  - Compute-intensive code regions (kernels)
  - Highly parallel
  - Compute scaling stronger than data transfer, e.g., compute $O(n^3)$ vs. data size $O(n^2)$

Offload Candidates /2

Offload candidates are regions of code that are amenable for offloading.

- Finding offload candidates
  - Create a benchmark to trigger application code of interest
  - Find hotspots in the application
  - Determine input and output data
  - Determine data sizes transferred

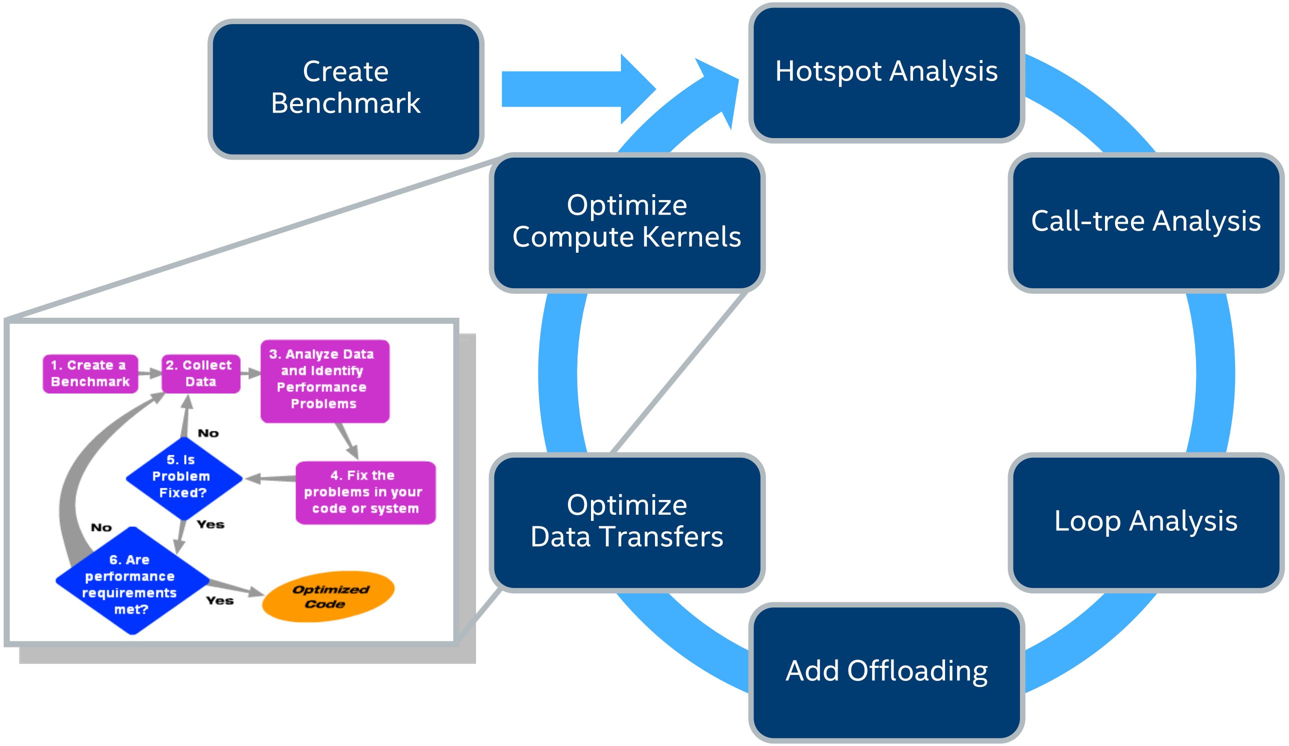
Finding Offload Candidates

Hard way: Stare hard and long enough at the code

Easier way: Augment code with logging/profiling code

Right way: Use an external profiling tool to measure

Offload Analysis Methodology



Intermezzo: Profiling Tools

Event-based
e.g., the built-in Python* cProfile profiling tool

Instrumentation-based
Requires modifications to the target

Statistical
Provide approximate results, but less intrusive

Intel® VTune™ Amplifier XE

Why is your application slow?

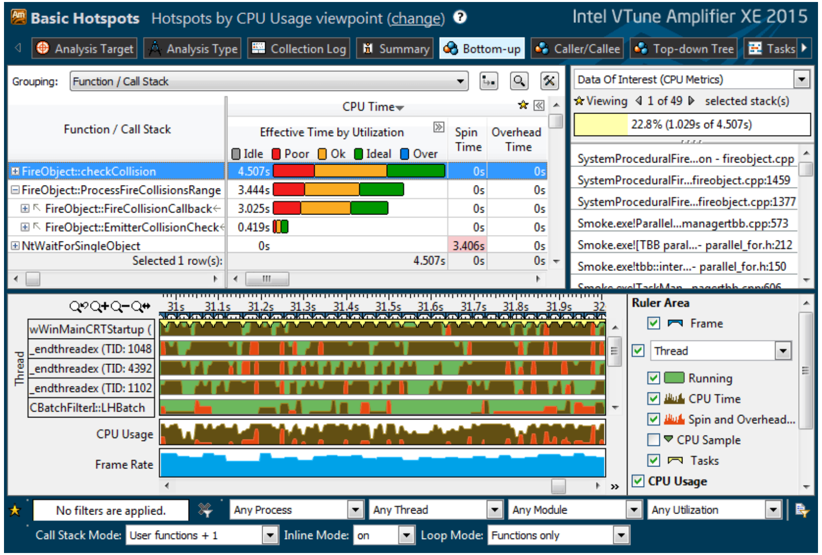
Does its speed scale with more cores?

Tuning without data is just guessing

- Accurate CPU, GPU[1] & threading data
- Powerful analysis & filtering of results
- Easy set-up, no special compiles

> "Last week, Intel® VTune™ Amplifier XE helped us find almost 3x performance improvement. This week it helped us improve the performance another 3x."

Claire Cates
Principal Developer
SAS Institute Inc.



For Windows* and Linux* From $899
(GUI only now available on OS X*)

http://intel.ly/vtune-amplifier-xe

Intel® VTune™ Amplifier XE /2

Get the Data You Need

- Hotspot (Statistical call tree), Call counts (Statistical)
- Thread Profiling – Concurrency and Lock & Waits Analysis
- Cache miss, Bandwidth analysis…[1]
- GPU Offload and OpenCL™ Kernel Tracing on Windows

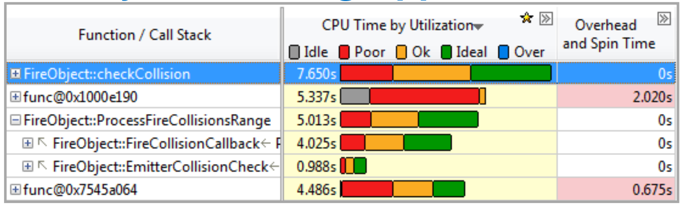
Find Answers Fast

- View Results on the Source / Assembly
- OpenMP Scalability Analysis, Graphical Frame Analysis
- Filter Out Extraneous Data - Organize Data with Viewpoints
- Visualize Thread & Task Activity on the Timeline

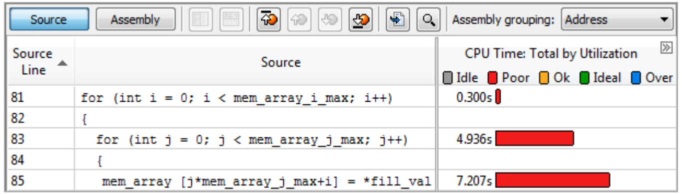
Easy to Use

- No Special Compiles – C, C++, C#, Fortran, Java, ASM
- Visual Studio* Integration or Stand Alone on Windows* or Linux*
- Graphical Interface & Command Line
- Local & Remote Data Collection
- New! Analyze Windows* & Linux* data on OS X* [2]

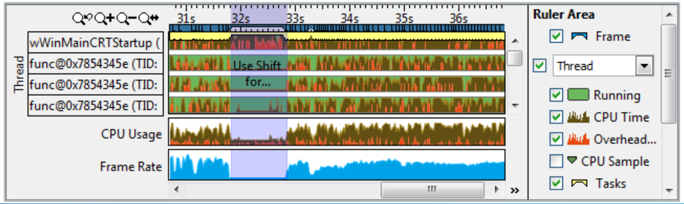
Quickly Find Tuning Opportunities



See Results On The Source Code



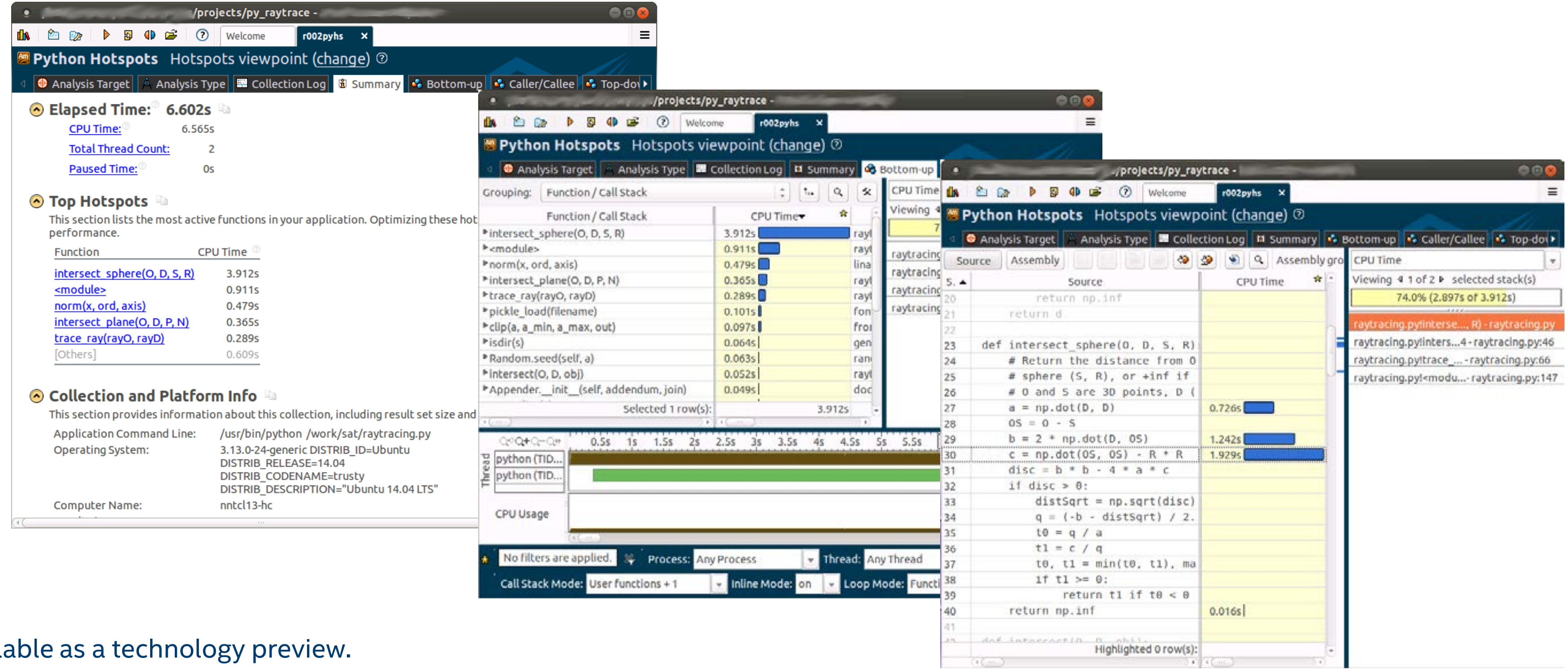
Timeline Visualizes & Filters



[1] Events vary by processor.   [2] No data collection on OS X*

*Other names and brands may be claimed as the property of others.

Profiling Python* Applications with Intel® VTune™ for Python



Available as a technology preview.

Our Example: GPAW
(see https://wiki.fysik.dtu.dk/gpaw/)

GPAW simulates various quantum mechanical effects at atomic scale

- Few hundred users all over the world
- Implemented as a combination of Python and C
  - High-level algorithms in Python
  - Compute kernels in C (or in libraries)
  - Massively parallel (MPI)

Finding Offload Candidates /2

Use Intel® VTune™ Amplifier XE to find hotspots in GPAW

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time
gemm	gpaw-python	281.368s
bmgs_fd_worker	gpaw-python	263.451s
bmgs_interpolate1D2_worker	gpaw-python	121.597s
bmgs_relax	gpaw-python	75.832s
rk	gpaw-python	60.337s
[Others]	N/A*	411.132s

**N/A is applied to non-summable metrics.*

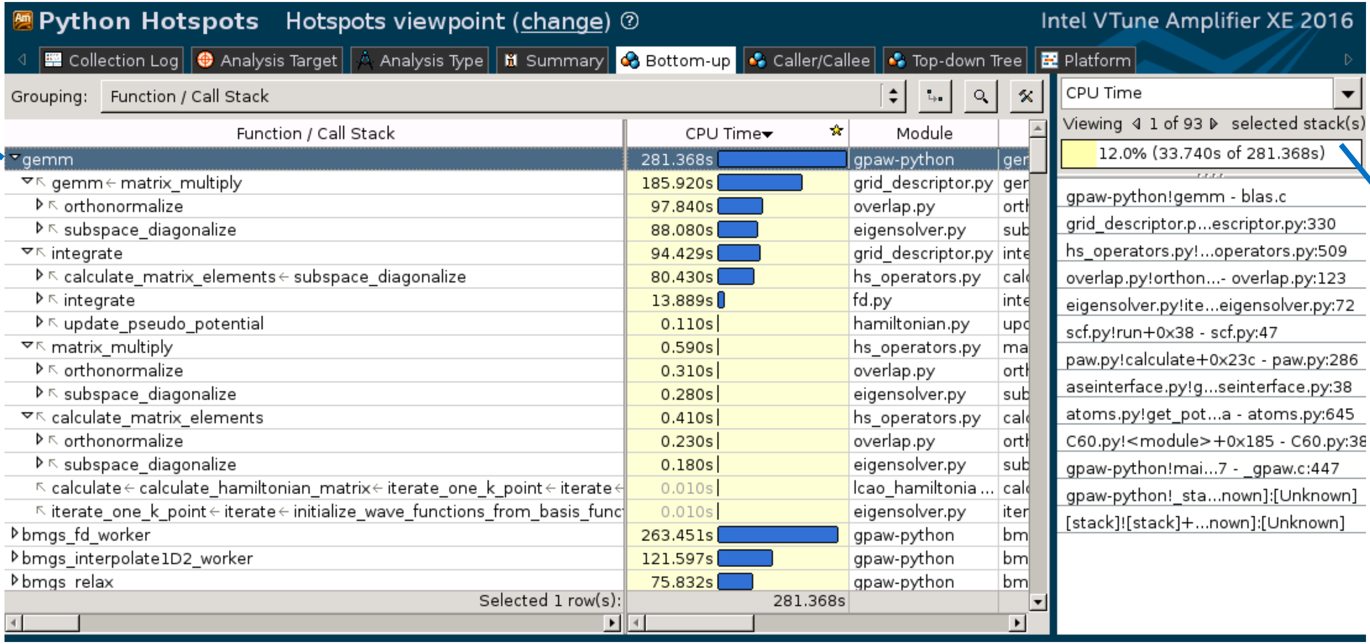
Offload candidate?

Available as a technology preview.

Finding Offload Candidates /3

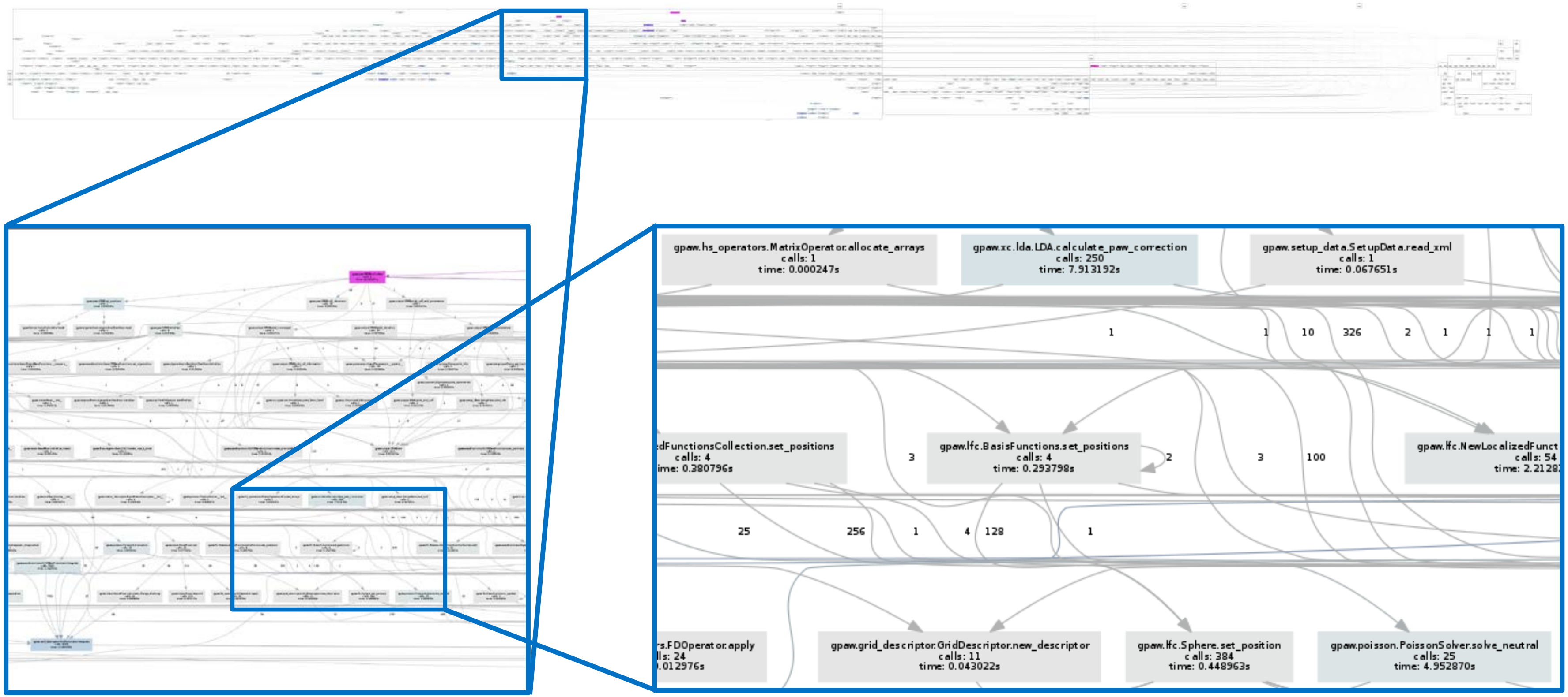
Use Intel® VTune™ Amplifier XE to navigate the call stack

Hotspot



Call stacks

Intermezzo: Why an Interactive Tool is Helpful



Agenda

- Quick Introduction to the Intel® Xeon Phi™ Coprocessor
- Finding Offload Candidates
- Using the Python Offload Module

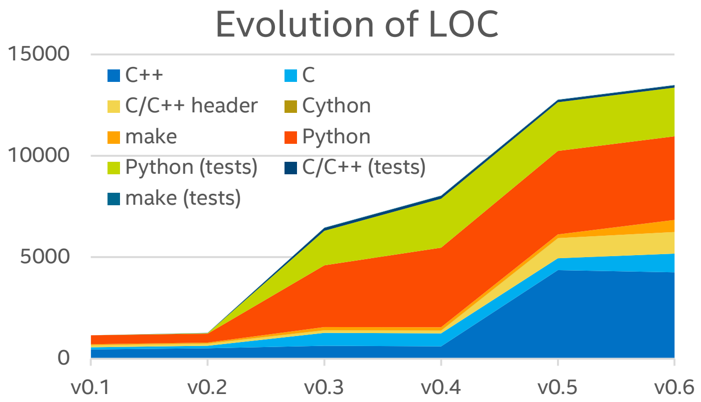
The Python* Offload Infrastructure for the Intel® Many Integrated Core Architecture

Design principles (pyMIC's 4 "K"s)

- Keep usage simple
- Keep the API slim
- Keep the code fast
- Keep control in a programmer's hand

pyMIC trivia

- BSD license
- 3800 lines of C/C++ code;
- 1100 lines of Python code for the main API;
- libxstream and Intel® LEO for interfacing with MPSS



High-Level Overview

LIBXSTREAM & Intel® LEO:

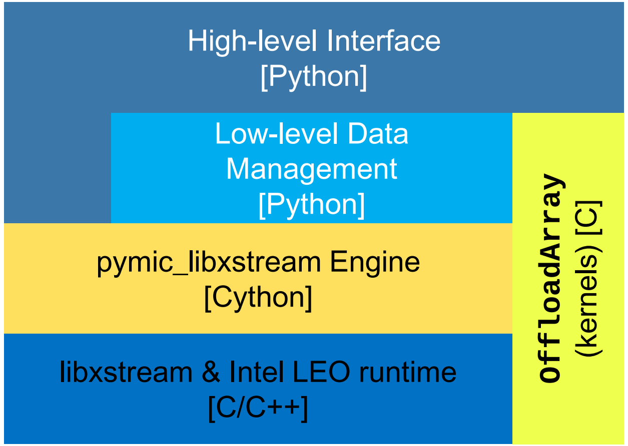
- Transfer of shared libraries
- Data transfers, kernel invocation

Cython extension module:

- Low-level device management
- Interaction with LEO

Levels of abstraction:

- Low-level API with memcpy-like interface, smart device pointers
- High-level API with offload arrays
- Library with internal device kernels



Example dgemm: The Host Side…

```python
import numpy as np




m, n, k = 4096, 4096, 4096
alpha = 1.0
beta = 0.0
np.random.seed(10)
a = np.random.random(m * k).reshape((m, k))
b = np.random.random(k * n).reshape((k, n))
c = np.empty((m, n))

am = np.matrix(a)
bm = np.matrix(b)
cm = np.matrix(c)
cm = alpha * am * bm + beta * cm
```

```python
import pymic as mic
import numpy as np

device = mic.devices[0]
stream = device.get_default_stream()
library = device.load_library("libdgemm.so")

m,n,k = 4096,4096,4096
alpha = 1.0
beta = 0.0
np.random.seed(10)
a = np.random.random(m*k).reshape((m, k))
b = np.random.random(k*n).reshape((k, n))
c = np.empty((m, n))

stream.invoke(library.dgemm_kernel,
              a, b, c,
              m, n, k, alpha, beta)
stream.sync()
```

Example dgemm: The Host Side…

- Get a device handle
- (numbered from 0 to n-1)
- Load native code as a shared-object library

- Invoke kernel function and pass actual arguments
- Copy-in/copy-out semantics for arrays
- Copy-in semantics for scalars

- Synchronize host and coprocessor

```
import pymic as mic
import numpy as np

device = mic.devices[0]
stream = device.get_default_stream()
library = device.load_library("libdgemm.so")

m,n,k = 4096,4096,4096
alpha = 1.0
beta = 0.0
np.random.seed(10)
a = np.random.random(m*k).reshape((m, k))
b = np.random.random(k*n).reshape((k, n))
c = np.empty((m, n))

stream.invoke(library.dgemm_kernel,
              a, b, c,
              m, n, k, alpha, beta)
stream.sync()
```

Example dgemm: The Target Side…

- Arguments are passed as C/C++ types
- All argument passing is done with pointers to actual data

- Invoke (native) dgemm kernel

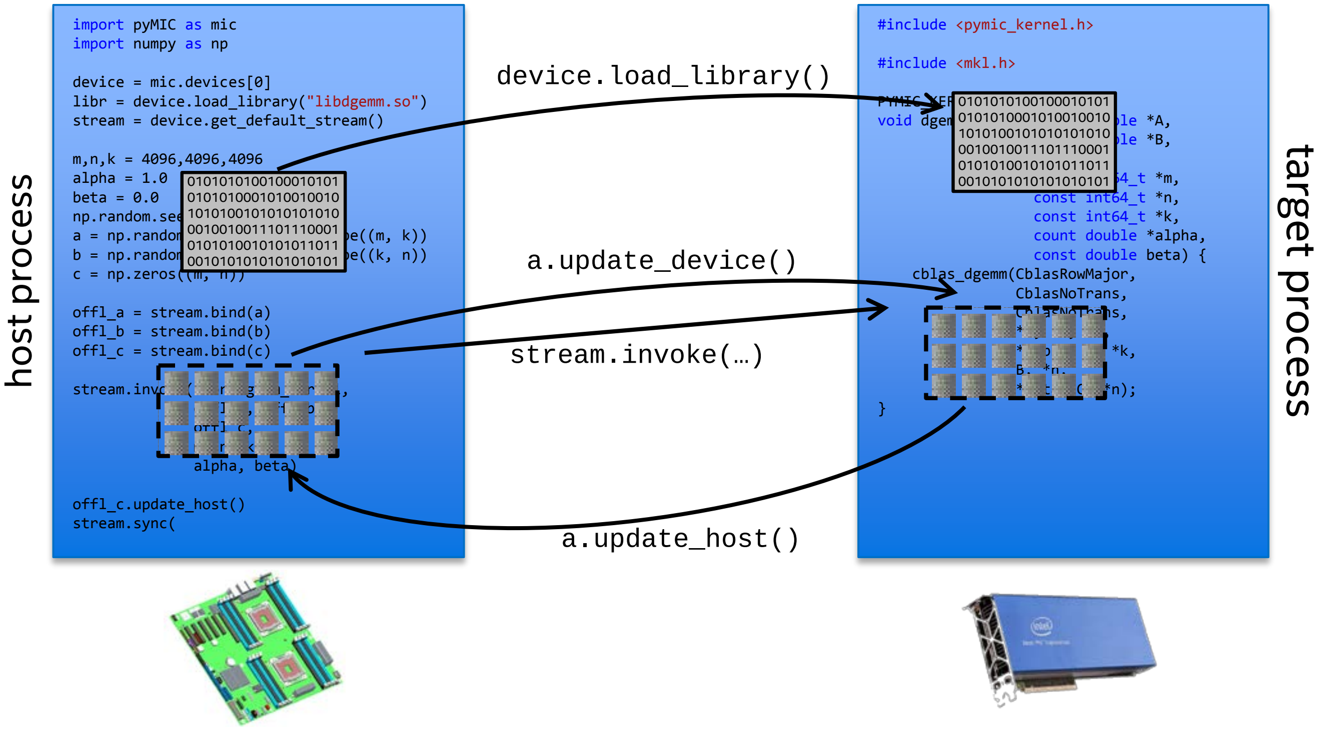
```
#include <pymic_kernel.h>
#include <mkl.h>

PYMIC_KERNEL
void dgemm_kernel(const double *A, const double *B,
                  double *C,
                  const int64_t *m, const int64_t *n,
                  const int64_t *k,
                  const double *alpha,
                  const double *beta) {
    cblas_dgemm(CblasRowMajor,
                CblasNoTrans, CblasNoTrans,
                *m, *n, *k,
                *alpha, A, *k, B, *n,
                *beta, C, *n);
}
```

Optimizing Data Transfers

- Use bind to create an array buffer for host data
- Invoke kernel function and pass actual arguments
- Use offload array instead of NumPy arrays
- No data implicit transfers for offload arrays
- Update host data from the device buffer

```
import pymic as mic
import numpy as np

device = mic.devices[0]
stream = device.get_default_stream()
library = device.load_library("libdgemm.so")

m,n,k = 4096,4096,4096
alpha,beta = 1.0,0.0
np.random.seed(10)
a = np.random.random(m*k).reshape((m, k))
b = np.random.random(k*n).reshape((k, n))
c = np.zeros((m, n))

offl_a = stream.bind(a)
offl_b = stream.bind(b)
offl_c = stream.bind(c)
stream.invoke(library.dgemm_kernel,
              offl_a, offl_b, offl_c,
              m, n, k, alpha, beta)
offl_c.update_host()
stream.sync()
```

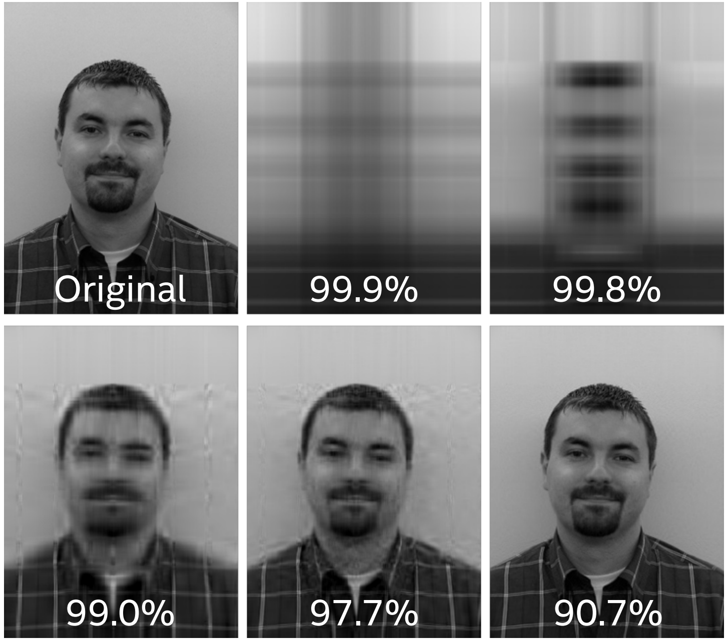
The High-level Offload Protocol



Example: Singular Value Decomposition (SVD)

Picture compression

- Treat picture as 2D matrix
- Decompose matrix:
  $$M = U \cdot \Sigma \cdot V^T$$
- Ignore some singular values, e.g.,
  - Values close to 0 or less than ε
  - Restrict dimensionality of Σ
- Effectively compresses images



Example: Singular Value Decomposition

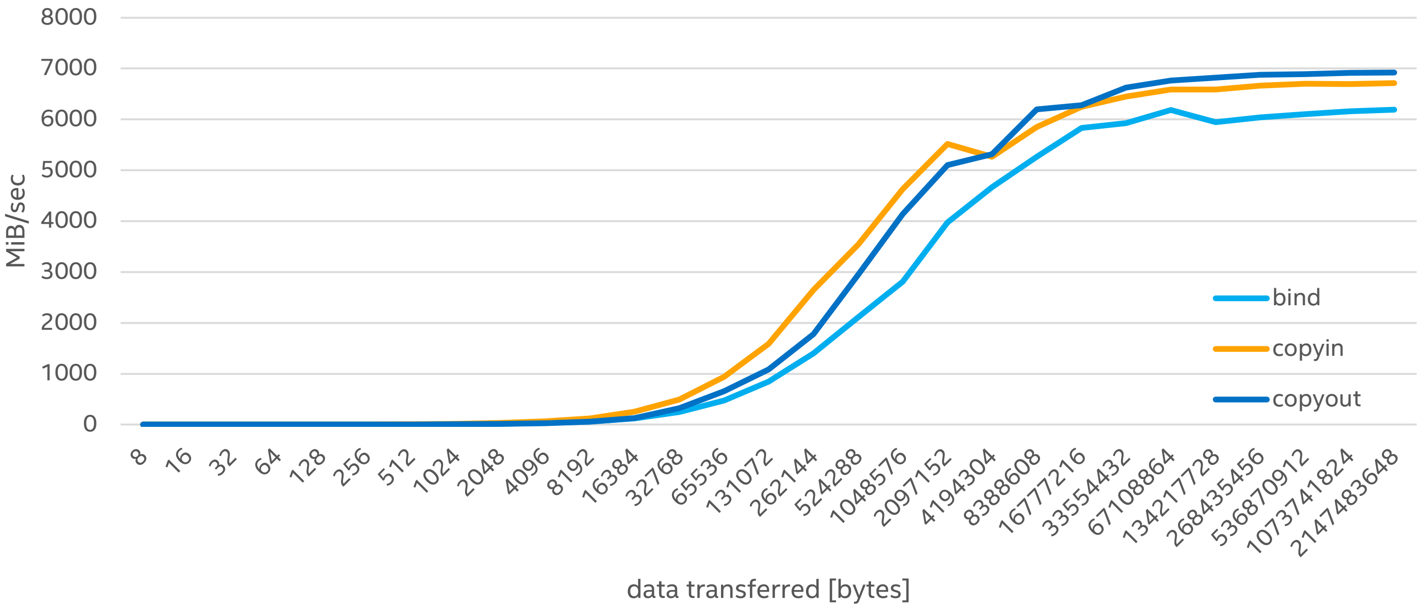
Host code

```python
import numpy as np
import pymic as mic
from PIL import Image

def compute_svd(image):
    mtx = np.asarray(image.getdata(band=0),
                     float)
    mtx.shape = (image.size[1], image.size[0])
    mtx = np.matrix(mtx)
    return np.linalg.svd(mtx)

def reconstruct_image(U, sigma, V):
    reconstructed = U * sigma * V
    image = Image.fromarray(reconstructed)
    return image
```

Host code, continued

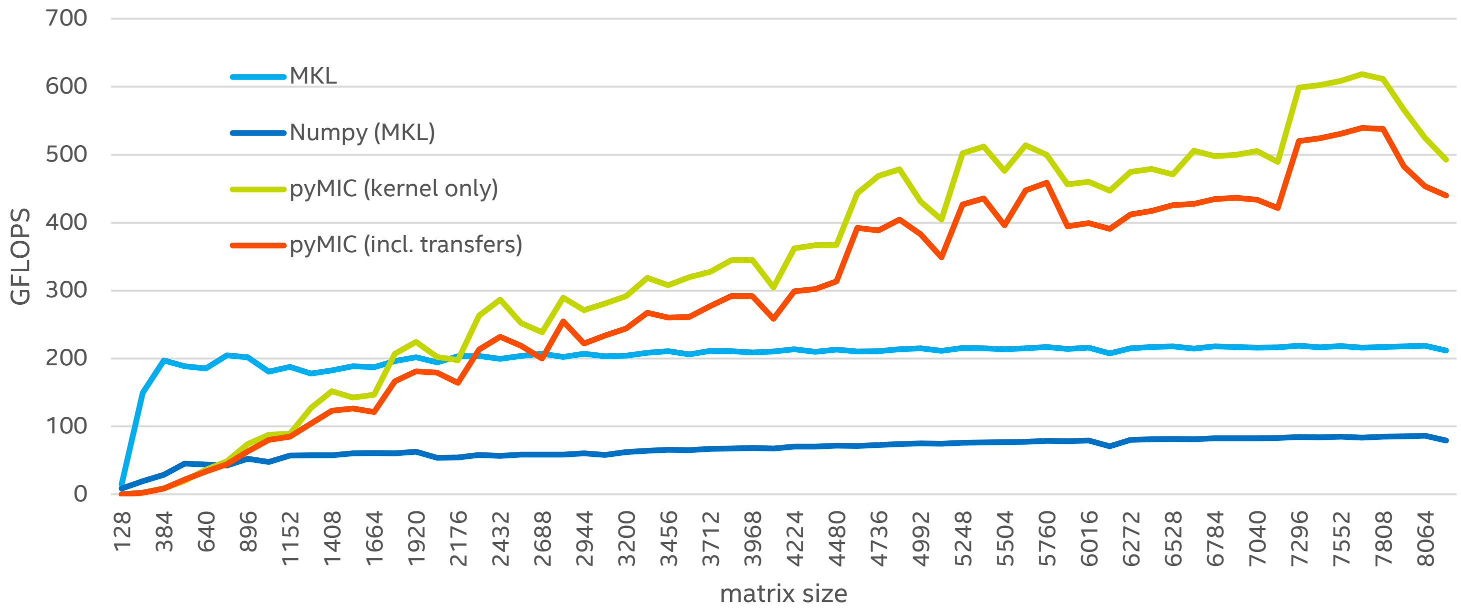
```python
def reconstruct_image_dgemm(U, sigma, V):
    offl_tmp    = stream.empty((U.shape[0], U.shape[1]),
                               dtype=float, update_host=False)
    offl_res    = stream.empty((U.shape[0], V.shape[1]),
                               dtype=float, update_host=False)
    offl_U, offl_sigma = stream.bind(U), stream.bind(sigma)
    offl_V      = stream.bind(V)
    alpha, beta = 1.0, 0.0
    m, k, n = U.shape[0], U.shape[1], sigma.shape[1]
    stream.invoke_kernel(library.dgemm_kernel,
                         offl_U, offl_sigma, offl_tmp,
                         m, n, k, alpha, beta)
    m, k, n = offl_tmp.shape[0], offl_tmp.shape[1], V.shape[1]
    stream.invoke_kernel(library.dgemm_kernel,
                         offl_tmp, offl_V, offl_res,
                         m, n, k, alpha, beta)
    offl_res.update_host()
    stream.sync()
    image = Image.fromarray(offl_res.array)
    return image
```

Performance: Bandwidth of Data Transfers



Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests 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. System configuration: Intel S2600GZ server with two Intel Xeon E5-2697v2 12-core processors at 2.7 GHz (64 GB DDR3 with 1867 MHz), Red Hat Enterprise Linux 6.5 (kernel version 2.6.32-358.6.2) and Intel C600 IOH, one Intel Xeon Phi 7120P coprocessor (C0 stepping, GDDR5 with 3.6 GT/sec, driver v3.3-1, flash image/micro OS 2.1.02.0390), and Intel Composer XE 14.0.3.174. For more complete information visit http://www.intel.com/performance.

Performance: dgemm



Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests 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. System configuration: Intel S2600GZ server with two Intel Xeon E5-2697v2 12-core processors at 2.7 GHz (64 GB DDR3 with 1867 MHz), Red Hat Enterprise Linux 6.5 (kernel version 2.6.32-358.6.2) and Intel C600 IOH, one Intel Xeon Phi 7120P coprocessor (C0 stepping, GDDR5 with 3.6 GT/sec, driver v3.3-1, flash image/micro OS 2.1.02.0390), and Intel Composer XE 14.0.3.174. For more complete information visit http://www.intel.com/performance.

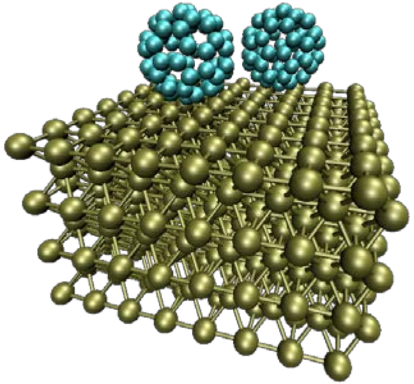
Offloading in GPAW

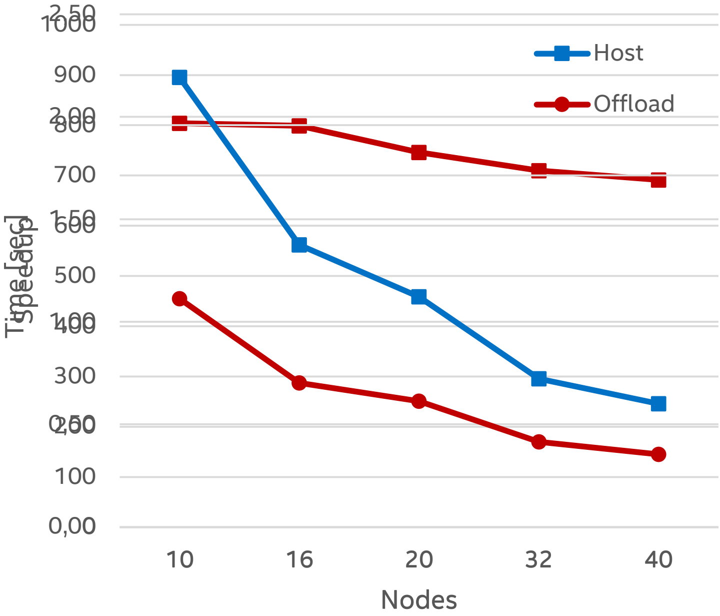
```python
from gpaw.grid_descriptor
   import GridDescriptor

gpts = (64, 64, 64)
nbands = 512
cell = (8.23, 8.23, 8.23)
gd = GridDescriptor(gpts, cell)

psit_nG = gd.zeros(nbands, mic=True)
vt_G = gd.zeros(mic=True)
# Initialize psit_nG and vt_G
htpsit_nG = gd.zeros(nbands, mic=True)

for n in range(nbands):
    htpsit_nG[n] = vt_G * psit_nG[n]

H_nn = gd.integrate(psit_nG, htpsit_ng)
```

```python
import pymic as mic
device = mic.devices[0]
stream = device.get_default_stream()
...
    def zeros(self, n=(), dtype=float,
              mic=False):
        array = self._new_array(n, dtype)
        if mic:
            return stream.bind(array)
        else:
            return array
```

Offload Performance for GPAW

Benchmark “C60 Pb100”





Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests 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. System configuration: Intel S2600GZ server with two Intel Xeon E5-2620v2 6-core processors at 2.1 GHz (64 GB DDR3 with 1600 MHz), Red Had Enterprise Linux 6.5 (kernel version 2.6.32-358.6.2) and two Intel Xeon Phi 7120P coprocessor (C0 stepping, GDDR5 with 3.6 GT/sec, MPSS v3.3.30726), and Intel Composer XE 14.0.3.174. For more complete information visit http://www.intel.com/performance.

Summary

Use Intel® VTune Amplifier XE for Python to find hotspots in the applications

- Optimization targets
- Offload candidates for Intel® Xeon Phi™ Coprocessors

Roadmap for the Python offload module

- Offloading of Cython code
- Events for synchronizing offload streams
- Support for the next-gen Intel® Xeon Phi™ Processor (aka Offload over Fabric)

Download pyMIC at https://github.com/01org/pyMIC
Mailinglist at https://lists.01.org/mailman/listinfo/pymic