Accelerating Python: Offloading to Intel® Xeon Phi™ (Co)processors

Dr.-Ing. Michael Klemm
Software and Services Group
Intel Corporation

Legal Disclaimer & Optimization Notice

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

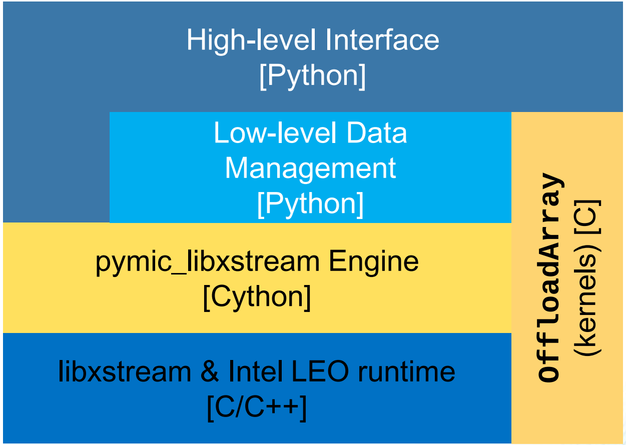
The pyMIC Offload Infrastructure

- 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 facts
  - 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: low-level device interaction
  - Transfer of shared libraries
  - Data transfers, kernel invocation
- C/C++ extension module
  - Low-level device management
  - Interaction with LEO
- 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

```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 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);
}
```

High-level Data Structures

`OffloadDevice`

- Interaction with devices
- Loading of shared libraries

`OffloadStream`

- Invocation of kernel functions
- Buffer management

`OffloadArray`

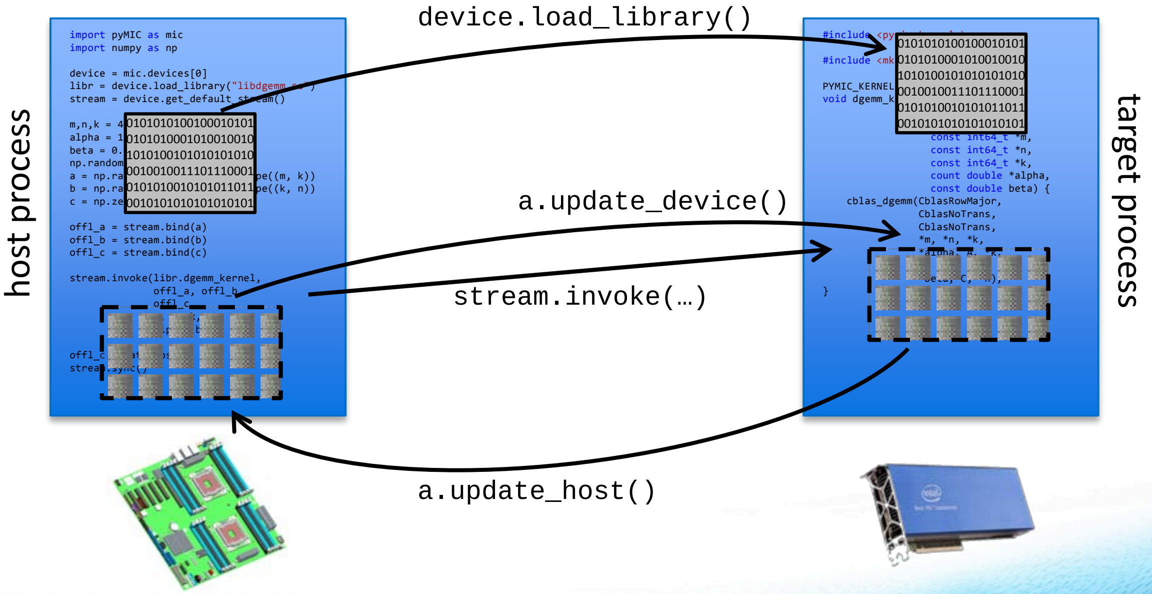
- numpy.ndarray container
- Transfer management
- Simple kernels and operators (fill, +, *)

Optimize Offloads with High-level Containers

- Get a device handle (numbered from 0 to n-1)
- Load native code as a shared-object library
- Use bind to create an offload buffer for host data
- Invoke kernel function and pass actual arguments
- Update host data from the device buffer

```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.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



Buffer Management: Buffer Creation

```python
class OffloadStream:
  def bind(self, array, update_device=True):
    if not isinstance(array, numpy.ndarray):
      raise ValueError("only numpy.ndarray can be associated "
                       "with OffloadArray")

    # detect the order of storage for 'array'
    if array.flags.c_contiguous:
      order = "C"
    elif array.flags.f_contiguous:
      order = "F"
    else:
      raise ValueError("could not detect storage order")

    # construct and return a new OffloadArray
    bound = pymic.OffloadArray(array.shape, array.dtype, order, False,
                               device=self._device, stream=self)
    bound.array = array

    # allocate the buffer on the device (and update data)
    bound._device_ptr = self.allocate_device_memory(bound._nbytes)
    if update_device:
      bound.update_device()

    return bound
```

```python
class OffloadStream:
  def allocate_device_memory(self, nbytes, alignment=64, sticky=False):
    device = self._device_id
  if nbytes <= 0:
    raise ValueError('Cannot allocate negative amount of '
                     'memory: {0}'.format(nbytes))
  device_ptr = _pymic_impl_stream_allocate(device, self._stream_id,
                                           nbytes, alignment)
  return SmartPtr(self, device, device_ptr, sticky)
```

```cpp
unsigned char *buffer_allocate(int device,
                               libxstream_stream *stream,
                               size_t size,
                               size_t alignment) {
  void *memory = NULL;
  libxstream_mem_allocate(device, &memory, size, alignment);
  return reinterpret_cast<unsigned char *>(memory);
}
```

Buffer Management: Data Transfer

```python
class OffloadArray:
  def update_device(self):
    host_ptr = self.array.ctypes.get_data()
    s = self.stream
    s.transfer_host2device(host_ptr,
                           self._device_ptr,
                           self._nbytes)
    return None

  def update_host(self):
    host_ptr = self.array.ctypes.get_data()
    s = self.stream
    s.transfer_device2host(self._device_ptr,
                           host_ptr,
                           self._nbytes)
  return self
```

```c
void buffer_copy_to_target(int device,
                           libxstream_stream *stream,
                           unsigned char *src,
                           unsigned char *dst,
                           size_t size,
                           size_t offset_host,
                           size_t offset_device) {
  unsigned char *src_offs = src + offset_host;
  unsigned char *dst_offs = dst + offset_device;

  libxstream_memcpy_h2d(src_offs, dst_offs,
                        size, stream);
}
```

SOFTWARE AND SERVICES

The Low-level Offload Protocol

host process

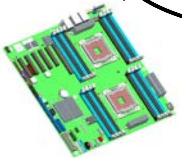
```python
import pyMIC as mic
import numpy as np

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

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.zeros((m, n))

offl_a = stream.bind(a)
offl_b = stream.bind(b)
offl_c = stream.bind(c)

stream.invoke(libr.dgemm_kernel,
              offl_a, offl_b,
              offl_c,
```

stream.allocate_device_memory(…)

stream.transfer_host2device(…)

```c
#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,
                  count double *alpha,
                  const double beta) {
    cblas_dgemm(CblasRowMajor,
                CblasNoTrans,
                CblasNoTrans,
                *m, *n, *k,
}
```

target process



stream.transfer_device2host(…)

stream.deallocate_device_memory(…)

Using the Low-level API

```python
import pymic
import numpy

device = pymic.devices[0]
stream = device.get_default_stream()

a = numpy.arange(0.0, 32.0)
b = numpy.empty_like(a)

# determine size of the array in bytes and get pointer
nbytes = a.dtype.itemsize * a.size
ptr_a = a.ctypes.data
ptr_b = b.ctypes.data

# allocate buffer spaces in the target
dev_ptr_1 = stream.allocate_device_memory(nbytes)
dev_ptr_2 = stream.allocate_device_memory(nbytes)
```

```python
# transfer a into the first buffer and shuffle a bit
stream.transfer_host2device(ptr_a, dev_ptr_1,
                            nbytes/2, offset_host=0,
                            offset_device=nbytes/2)
stream.transfer_host2device(ptr_a, dev_ptr_1, nbytes/2,
                            offset_host=nbytes/2,
                            offset_device=0)
# do some more shuffling on the target
for i in xrange(0, 4):
    stream.transfer_device2device(dev_ptr_1, dev_ptr_2,
                                  nbytes/4,
                                  off_dev_src=i*(nbytes/4),
                                  off_dev_dst=(3-i)*(nbytes/4))
# transfer data back into 'b' array and shuffle even more
for i in xrange(0, 4):
    stream.transfer_device2host(dev_ptr_2, ptr_b, nbytes/4,
                                offset_device=i*(nbytes/4),
                                offset_host=(3-i)*(nbytes/4))
stream.sync()
```

Example: Singular Value Decomposition

- Treat picture as 2D matrix
- Decompose matrix:

$$M = U \times \Sigma \times V^T$$

- Ignore some singular values
- Effectively compresses images

Example: Singular Value Decomposition

Host code

```
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, cont'd

```
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
```

Integration with GPAW

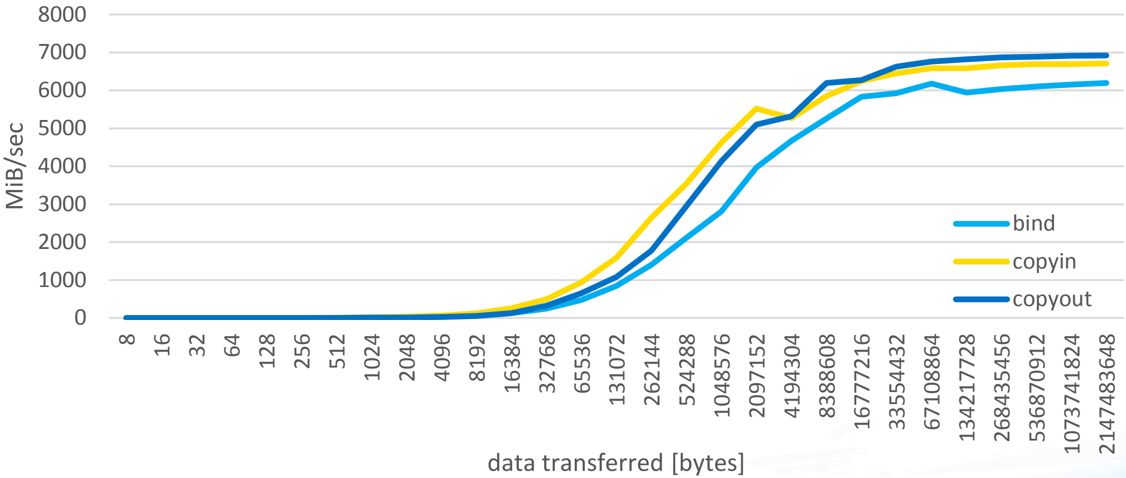
- GPAW is an open source software package for various quantum mechanical simulations at atomic scale
  - http://wiki.fysik.dtu.dk/gpaw
  - Few hundred users all over the world
- Implemented as a combination of Python and C
  - High-level algorithms in Python
  - Computational kernels in C (or in libraries)
  - Massively parallelized (with MPI)
  - Key operation: matrix-matrix multiplication

Integration in GPAW

```
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)
```

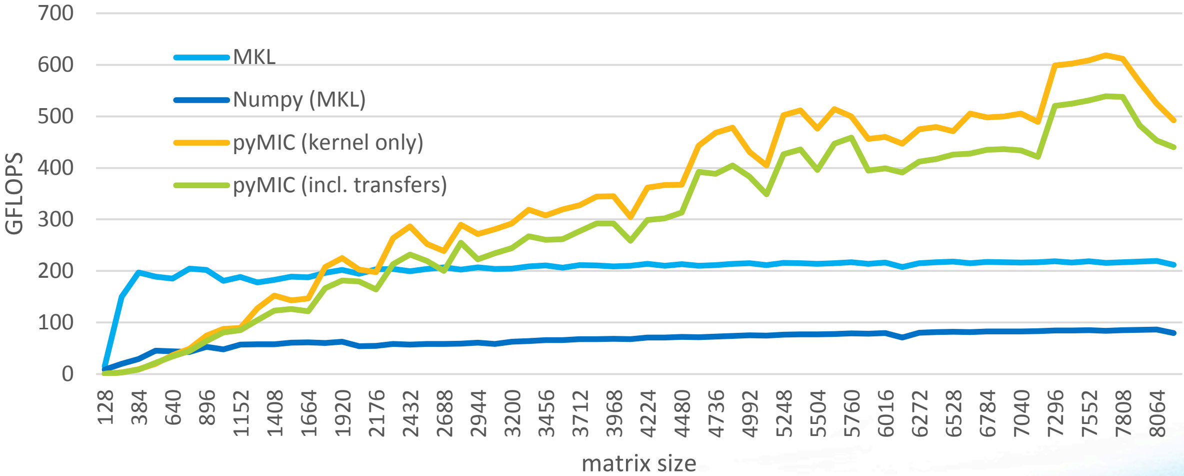
```
import pymic as mic

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
```

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.

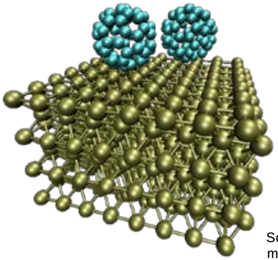
Performance: dgemm

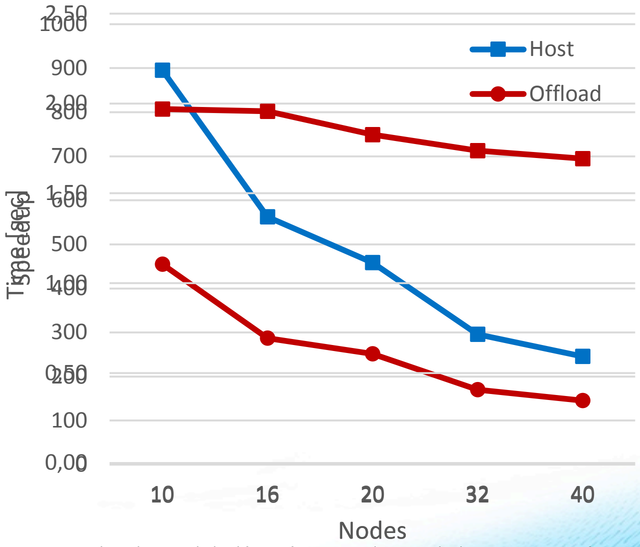


Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests are measured using specific computer systems, components, software, operations, and functions. Any change to any of those factors may cause the results to vary. You 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.

Example Application: GPAW

- Open-source software package for quantum mechanical simulations at atomic scale (http://wiki.fysik.dtu.dk/gpaw)
- Combination of Python & C
  - High-level algorithms in Python
  - Computational kernels in
  - Massively parallelized (with MPI)
  - Key operation: 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 E5-2620v2 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 & Future Work

- pyMIC
  - A slim, easy-to-use offload interface for Python
  - Native kernels on the target devices
  - Almost negligible extra overhead for Python integration

- Future versions will likely bring:
  - Offloading of full {C|P}ython code

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

intel®