Speeding up Python using Cython

Rolf Boomgaarden · Thiemo Gries · Florian Letsch

Universität Hamburg

November 28th, 2013

What is Cython?

- Compiler, compiles Python-like code to C-code
- Code is still executed in the Python runtime environment
- But is compiled to native machine code instead of Python bytecode
- Can result in more speed and easy wrapping of C libraries

Cython workflow

first approach

- Write a helloworld.pyx source file

```
print "Hello␣World"
```

- Run the Cython compiler to generate a C file

```
$ cython helloworld.pyx
```

- Run a C compiler to generate a compiled library

```
$ gcc [...] -o helloworld.so helloworld.c
```

- Run the Python interpreter and import the module

```
>>> import helloworld
Hello World
```

Cython workflow

first approach

- Write a helloworld.pyx source file

```
print "Hello␣World"
```

- Run the Cython compiler to generate a C file

```
$ cython helloworld.pyx
```

- Run a C compiler to generate a compiled library

```
$ gcc [...] -o helloworld.so helloworld.c
```

- Run the Python interpreter and import the module

```
>>> import helloworld
Hello World
```

Cython workflow

second approach

- Write a helloworld.pyx source file
- Write a setup.py file with compile information

```
from distutils.core import setup
from Cython.Build import cythonize

setup(
    ext_modules = cythonize("helloworld.pyx")
)
```

- Let Python compile the file

```
$ python setup.py build_ext --inplace
```

- Run the Python interpreter and import the module

Cython workflow

second approach

- Write a helloworld.pyx source file
- Write a setup.py file with compile information

```
from distutils.core import setup
from Cython.Build import cythonize

setup(
    ext_modules = cythonize("helloworld.pyx")
)
```

- Let Python compile the file

```
$ python setup.py build_ext --inplace
```

- Run the Python interpreter and import the module

Cython workflow

third approach: pyximport

- Write a helloworld.pyx source file
- Use Pyximport

```
>>> import pyximport; pyximport.install()
>>> import helloworld
Hello World
```

Pure Python

```python
import numpy as np

def my_add(a, b):
    (...) # validate parameter

    dtype = a.dtype
    height = a.shape[0]
    width = a.shape[1]

    result = np.zeros((height, width), dtype=dtype)

    for y in range(height):
        for x in range(width):
            result[y,x] = a[y,x] + b[y,x]

    return result
```

Listing 1: add1.py

Pure Python

```python
import numpy as np

def my_add(a, b):
    (...) # validate parameter

    dtype = a.dtype
    height = a.shape[0]
    width = a.shape[1]

    result = np.zeros((height, width), dtype=dtype)

    for y in range(height):
        for x in range(width):
            result[y,x] = a[y,x] + b[y,x]

    return result
```

Listing 1: add1.py

`Time: ∼19 minutes (2048x2048, 100x)`

Python run through Cython

```python
import numpy as np

def my_add(a, b):
    (...) # validate parameter

    dtype = a.dtype
    height = a.shape[0]
    width = a.shape[1]

    result = np.zeros((height, width), dtype=dtype)

    for y in range(height):
        for x in range(width):
            result[y,x] = a[y,x] + b[y,x]

    return result
```

Listing 2: add2.pyx

Python run through Cython

```python
import numpy as np

def my_add(a, b):
    (...) # validate parameter

    dtype = a.dtype
    height = a.shape[0]
    width = a.shape[1]

    result = np.zeros((height, width), dtype=dtype)

    for y in range(height):
        for x in range(width):
            result[y,x] = a[y,x] + b[y,x]

    return result
```

Listing 2: add2.pyx

`Time: ∼16 minutes (2048x2048, 100x)`

Cython: Adding types

```
import numpy as np; cimport numpy as np

DTYPE = np.uint8
ctypedef np.uint8_t DTYPE_t

def my_add(np.ndarray a, np.ndarray b):
    (...) # validate parameter

    cdef int height = a.shape[0]
    cdef int width = a.shape[1]
    cdef np.ndarray result = np.zeros((height, width), dtype=DTYPE)

    cdef int x, y
    for y in range(height):
        for x in range(width):
            result[y,x] = a[y,x] + b[y,x]

    return result
```

Listing 3: add3.pyx

Cython: Adding types

```
import numpy as np; cimport numpy as np

DTYPE = np.uint8
ctypedef np.uint8_t DTYPE_t

def my_add(np.ndarray a, np.ndarray b):
    (...) # validate parameter

    cdef int height = a.shape[0]
    cdef int width = a.shape[1]
    cdef np.ndarray result = np.zeros((height, width), dtype=DTYPE)

    cdef int x, y
    for y in range(height):
        for x in range(width):
            result[y,x] = a[y,x] + b[y,x]

    return result
```

Listing 3: add3.pyx

`Time: ∼16 minutes (2048x2048, 100x)`

Cython: Efficient indexing

```
import numpy as np; cimport numpy as np

DTYPE = np.uint8
ctypedef np.uint8_t DTYPE_t

def my_add(np.ndarray[DTYPE_t,ndim=2] a, np.ndarray[DTYPE_t,ndim=2] b):
    (...) # validate parameter

    cdef int height = a.shape[0]
    cdef int width = a.shape[1]
    cdef np.ndarray[DTYPE_t, ndim=2] result = np.zeros((height, width),
          dtype=DTYPE)

    cdef int x, y
    for y in range(height):
        for x in range(width):
            result[y,x] = a[y,x] + b[y,x]

    return result
```

Listing 4: add4.pyx

Cython: Efficient indexing

```
import numpy as np; cimport numpy as np

DTYPE = np.uint8
ctypedef np.uint8_t DTYPE_t

def my_add(np.ndarray[DTYPE_t,ndim=2] a, np.ndarray[DTYPE_t,ndim=2] b):
    (...) # validate parameter

    cdef int height = a.shape[0]
    cdef int width = a.shape[1]
    cdef np.ndarray[DTYPE_t, ndim=2] result = np.zeros((height, width),
          dtype=DTYPE)

    cdef int x, y
    for y in range(height):
        for x in range(width):
            result[y,x] = a[y,x] + b[y,x]

    return result
```

Listing 4: add4.pyx

```
Time: 1.249 seconds (2048x2048, 100x)
```

Cython: Don't check boundaries

```
import numpy as np; cimport numpy as np; cimport cython

DTYPE = np.uint8
ctypedef np.uint8_t DTYPE_t

@cython.boundscheck(False)
def my_add(np.ndarray[DTYPE_t,ndim=2] a, np.ndarray[DTYPE_t,ndim=2] b):
    (...) # validate parameter

    cdef int height = a.shape[0]
    cdef int width = a.shape[1]
    cdef np.ndarray[DTYPE_t, ndim=2] result = np.zeros((height, width),
          dtype=DTYPE)

    cdef int x, y
    for y in range(height):
        for x in range(width):
            result[y,x] = a[y,x] + b[y,x]

    return result
```

Listing 5: add5.pyx

Cython: Don't check boundaries

```
import numpy as np; cimport numpy as np; cimport cython

DTYPE = np.uint8
ctypedef np.uint8_t DTYPE_t

@cython.boundscheck(False)
def my_add(np.ndarray[DTYPE_t,ndim=2] a, np.ndarray[DTYPE_t,ndim=2] b):
    (...) # validate parameter

    cdef int height = a.shape[0]
    cdef int width = a.shape[1]
    cdef np.ndarray[DTYPE_t, ndim=2] result = np.zeros((height, width),
          dtype=DTYPE)

    cdef int x, y
    for y in range(height):
        for x in range(width):
            result[y,x] = a[y,x] + b[y,x]

    return result
```

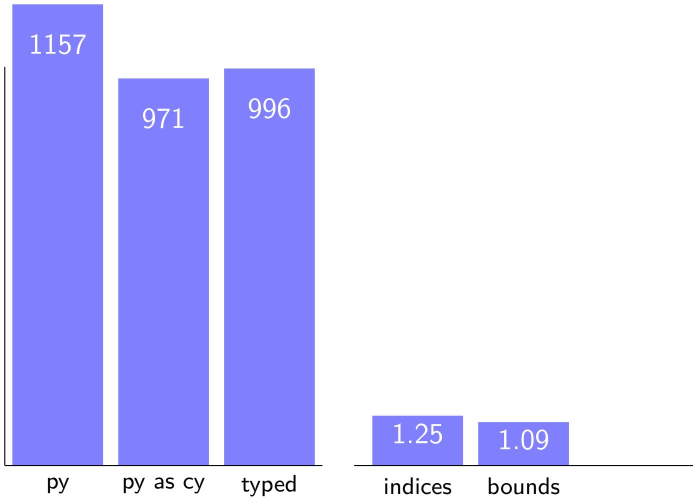
Listing 5: add5.pyx

`Time: 1.086 seconds (2048x2048, 100x)`

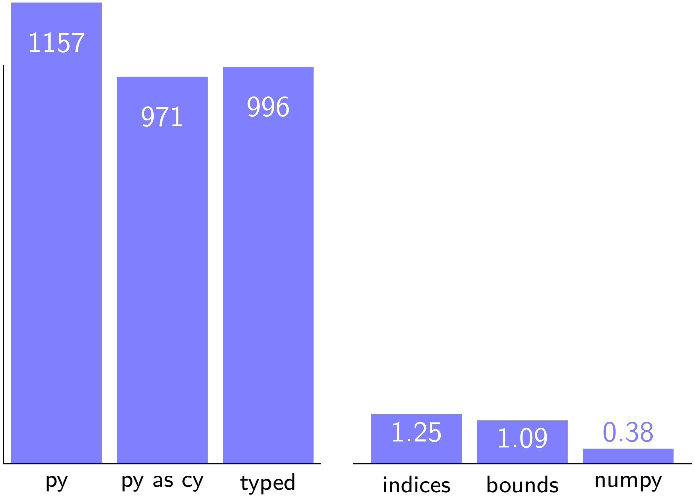
Obtaining speedup

- Static typing
- Access data buffer directly at C level
- Cython gimmicks

Conclusion



Conclusion


1157
971
996
1.25
1.09
0.38
py
py as cy
typed
indices
bounds
numpy

Now what?

1. http://docs.cython.org (tutorial style plus reference)
2. `with nogil:` release GIL
3. `cython.parallel`, parallelism with OpenMP support
4. PyPy support (considered mostly usable since Cython 0.17)
5. actual speedup compared to CPython; but compared to PyPy etc. ?

The end

EOF
(FIXME: better joke)