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

```
1 print "Hello World"
```

- Run the Cython compiler to generate a C file

```
1 $ cython helloworld.pyx
```

- Run a C compiler to generate a compiled library

```
1 $ gcc [...] -o helloworld.so helloworld.c
```

- Run the Python interpreter and import the module

```
1 >>> import helloworld
2 Hello World
```

Cython workflow

first approach

- Write a helloworld.pyx source file

```
1 print "Hello World"
```

- Run the Cython compiler to generate a C file

```
1 $ cython helloworld.pyx
```

- Run a C compiler to generate a compiled library

```
1 $ gcc [...] -o helloworld.so helloworld.c
```

- Run the Python interpreter and import the module

```
1 >>> import helloworld
2 Hello World
```

Cython workflow

second approach

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

```
1 from distutils.core import setup
2 from Cython.Build import cythonize
3
4 setup(
5     ext_modules = cythonize("helloworld.pyx")
6 )
```

- Let Python compile the file

```
1 $ 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

```
1 from distutils.core import setup
2 from Cython.Build import cythonize
3
4 setup(
5     ext_modules = cythonize("helloworld.pyx")
6 )
```

- Let Python compile the file

```
1 $ 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
```


KOGS
WINTER
of CODE
FIRST
2014
YEAR

Pure Python

```
1  import numpy as np
2
3  def my_add(a, b):
4      (...) # validate parameter
5
6      dtype = a.dtype
7      height = a.shape[0]
8      width = a.shape[1]
9
10     result = np.zeros((height, width), dtype=dtype)
11
12     for y in range(height):
13         for x in range(width):
14             result[y,x] = a[y,x] + b[y,x]
15
16     return result
```

Listing 1: add1.py

Pure Python

```
1  import numpy as np
2
3  def my_add(a, b):
4      (...) # validate parameter
5
6      dtype = a.dtype
7      height = a.shape[0]
8      width = a.shape[1]
9
10     result = np.zeros((height, width), dtype=dtype)
11
12     for y in range(height):
13         for x in range(width):
14             result[y,x] = a[y,x] + b[y,x]
15
16     return result
```

Listing 1: add1.py

Time: ∼19 minutes (2048x2048, 100x)

Python run through Cython

```
1  import numpy as np
2
3  def my_add(a, b):
4      (...) # validate parameter
5
6      dtype = a.dtype
7      height = a.shape[0]
8      width = a.shape[1]
9
10     result = np.zeros((height, width), dtype=dtype)
11
12     for y in range(height):
13         for x in range(width):
14             result[y,x] = a[y,x] + b[y,x]
15
16     return result
```

Listing 2: add2.pyx

Python run through Cython

```
1  import numpy as np
2
3  def my_add(a, b):
4      (...) # validate parameter
5
6      dtype = a.dtype
7      height = a.shape[0]
8      width = a.shape[1]
9
10     result = np.zeros((height, width), dtype=dtype)
11
12     for y in range(height):
13         for x in range(width):
14             result[y,x] = a[y,x] + b[y,x]
15
16     return result
```

Listing 2: add2.pyx

Time: ∼16 minutes (2048x2048, 100x)

Cython: Adding types

```
1  import numpy as np
2  cimport numpy as np
3
4  DTYPE = np.uint8
5  ctypedef np.uint8_t DTYPE_t
6
7  def my_add(np.ndarray a, np.ndarray b):
8      (...) # validate parameter
9
10     cdef int height = a.shape[0]
11     cdef int width = a.shape[1]
12     cdef np.ndarray result = np.zeros((height, width), dtype=DTYPE)
13
14     cdef int x, y
15     for y in range(height):
16         for x in range(width):
17             result[y,x] = a[y,x] + b[y,x]
18
19     return result
```

Listing 3: add3.pyx

Cython: Adding types

```
1  import numpy as np
2  cimport numpy as np
3
4  DTYPE = np.uint8
5  ctypedef np.uint8_t DTYPE_t
6
7  def my_add(np.ndarray a, np.ndarray b):
8      (...) # validate parameter
9
10     cdef int height = a.shape[0]
11     cdef int width = a.shape[1]
12     cdef np.ndarray result = np.zeros((height, width), dtype=DTYPE)
13
14     cdef int x, y
15     for y in range(height):
16         for x in range(width):
17             result[y,x] = a[y,x] + b[y,x]
18
19     return result
```

Listing 3: add3.pyx

Time: ~16 minutes (2048x2048, 100x)

Cython: Efficient indexing

```
1  import numpy as np
2  cimport numpy as np
3
4  DTYPE = np.uint8
5  ctypedef np.uint8_t DTYPE_t
6
7  def my_add(np.ndarray[DTYPE_t, ndim=2] a, np.ndarray[DTYPE_t, ndim=2] b
       ):
8      (...) # validate parameter
9
10     cdef int height = a.shape[0]
11     cdef int width = a.shape[1]
12     cdef np.ndarray[DTYPE_t, ndim=2] result = np.zeros((height, width),
           dtype=DTYPE)
13
14     cdef int x, y
15     for y in range(height):
16         for x in range(width):
17             result[y,x] = a[y,x] + b[y,x]
18
19     return result
```

Listing 4: add4.pyx

Cython: Efficient indexing

```
1  import numpy as np
2  cimport numpy as np
3
4  DTYPE = np.uint8
5  ctypedef np.uint8_t DTYPE_t
6
7  def my_add(np.ndarray[DTYPE_t, ndim=2] a, np.ndarray[DTYPE_t, ndim=2] b
       ):
8      (...) # validate parameter
9
10     cdef int height = a.shape[0]
11     cdef int width = a.shape[1]
12     cdef np.ndarray[DTYPE_t, ndim=2] result = np.zeros((height, width),
           dtype=DTYPE)
13
14     cdef int x, y
15     for y in range(height):
16         for x in range(width):
17             result[y,x] = a[y,x] + b[y,x]
18
19     return result
```

Listing 4: add4.pyx

Time: 1.249 seconds (2048x2048, 100x)

Cython: Don't check boundaries

```
1  import numpy as np; cimport numpy as np; cimport cython
2
3  DTYPE = np.uint8
4  ctypedef np.uint8_t DTYPE_t
5
6  @cython.boundscheck(False)
7  def my_add(np.ndarray[DTYPE_t, ndim=2] a, np.ndarray[DTYPE_t, ndim=2] b
       ):
8      (...) # validate parameter
9
10     cdef int height = a.shape[0]
11     cdef int width = a.shape[1]
12     cdef np.ndarray[DTYPE_t, ndim=2] result = np.zeros((height, width),
           dtype=DTYPE)
13
14     cdef int x, y
15     for y in range(height):
16         for x in range(width):
17             result[y,x] = a[y,x] + b[y,x]
18
19     return result
```

Listing 5: add5.pyx

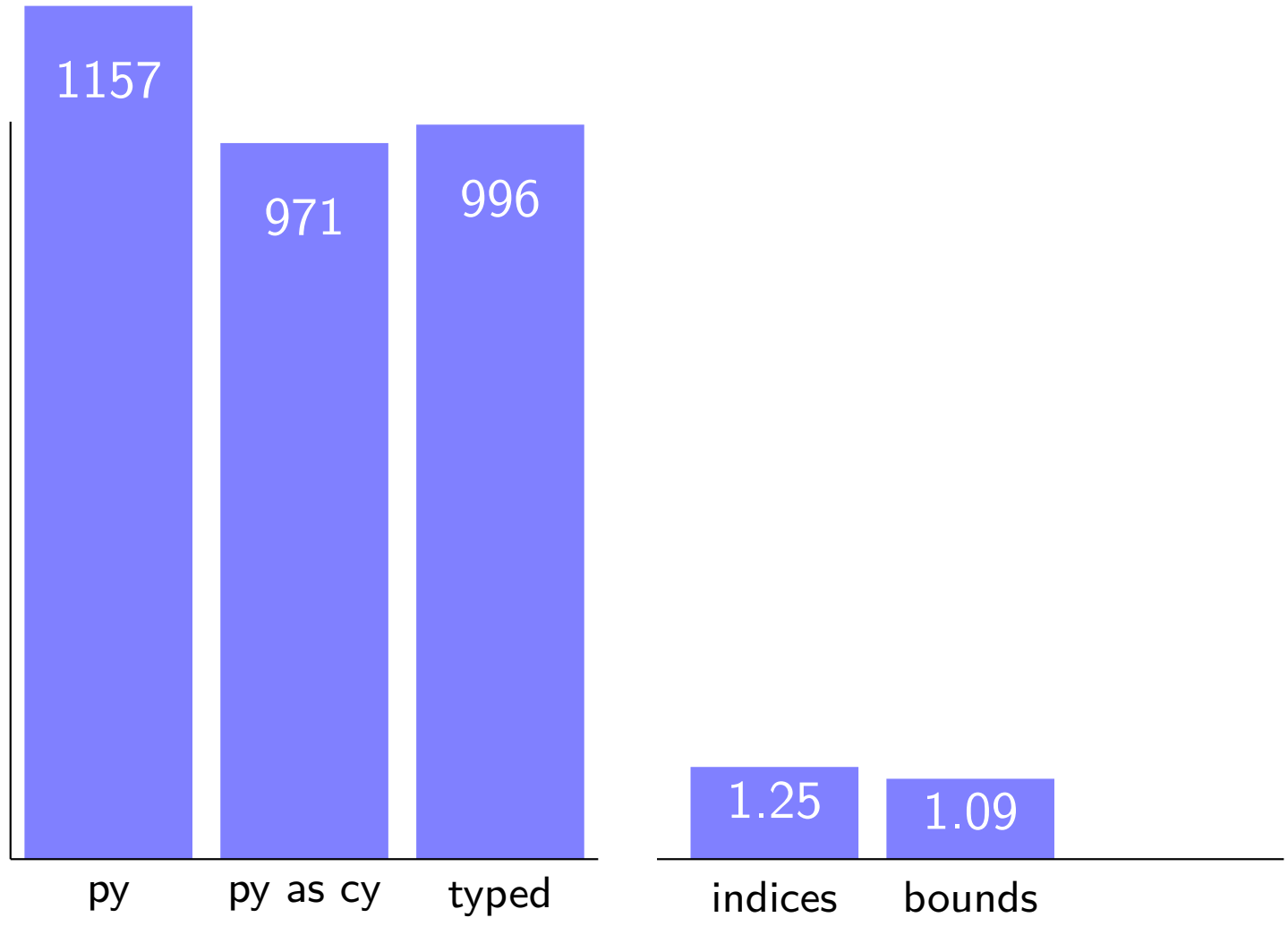
Cython: Don't check boundaries

```
1  import numpy as np; cimport numpy as np; cimport cython
2
3  DTYPE = np.uint8
4  ctypedef np.uint8_t DTYPE_t
5
6  @cython.boundscheck(False)
7  def my_add(np.ndarray[DTYPE_t, ndim=2] a, np.ndarray[DTYPE_t, ndim=2] b
       ):
8      (...) # validate parameter
9
10     cdef int height = a.shape[0]
11     cdef int width = a.shape[1]
12     cdef np.ndarray[DTYPE_t, ndim=2] result = np.zeros((height, width),
           dtype=DTYPE)
13
14     cdef int x, y
15     for y in range(height):
16         for x in range(width):
17             result[y,x] = a[y,x] + b[y,x]
18
19     return result
```

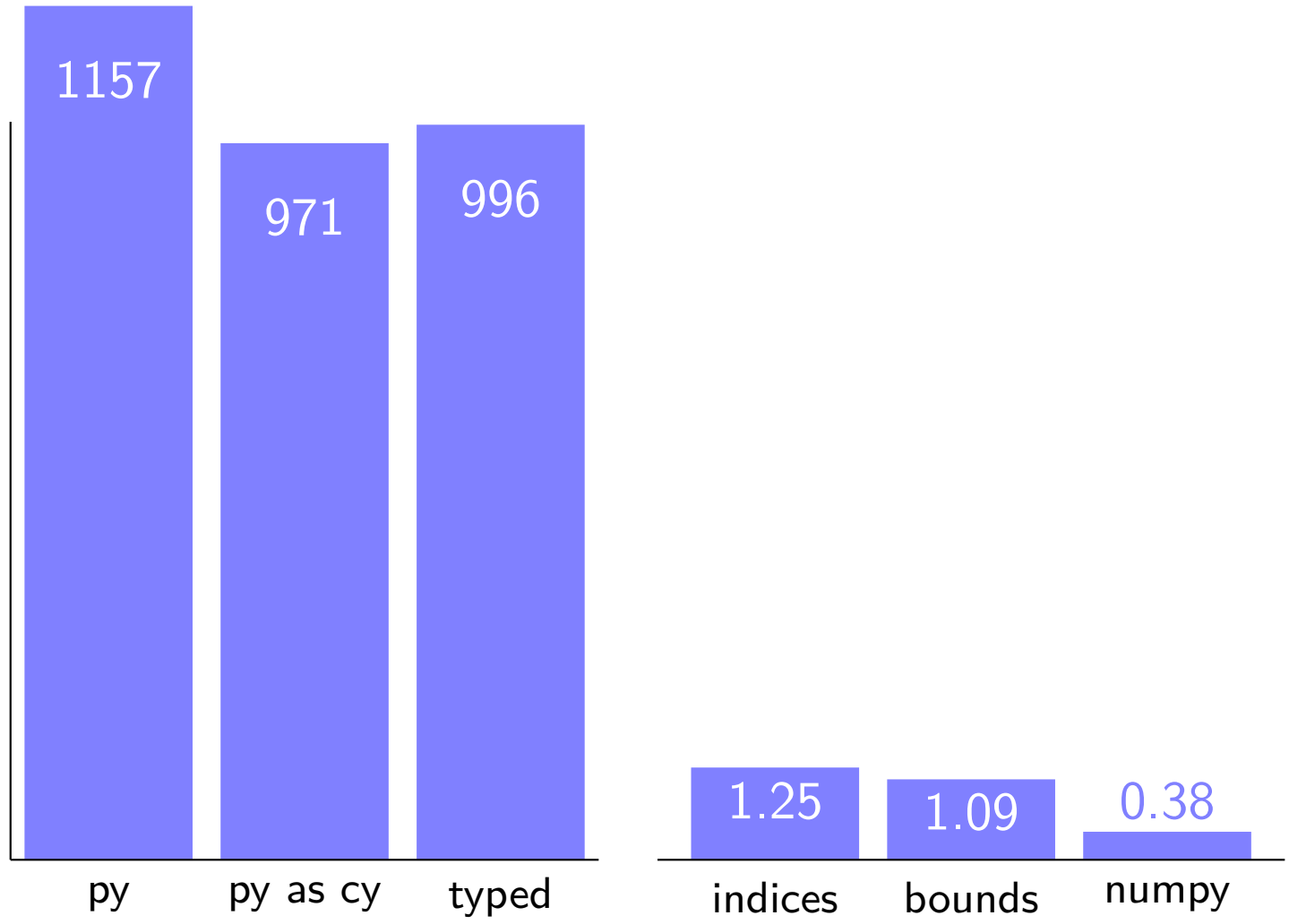
Listing 5: add5.pyx

Time: 1.086 seconds (2048x2048, 100x)

Conclusion



Conclusion


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

FIXME Flo

FIXME: speedup, how much fun to work with, easy use of features, documentation etc

The end

EOF
(FIXME: better joke)