PyPy in Production

Antonio Cuni
Armin Rigo

EuroPython 2011

June 23 2011

What is PyPy?

- Past EuroPython talks:
  - **2004**: PyPy
  - **2005**: PyPy as a compiler
  - **2006**: An introduction to PyPy, PyPy architecture session, What can PyPy do for you
  - **2007**: PyPy 1.0 and Beyond, PyPy Python Interpreter(s) Features, PyPy: Why and how did it (not) work?
  - **2008**: PyPy for the rest of us, PyPy status talk
  - **2009** PyPy: Complete and Fast
  - **2010**: PyPy 1.3: Status and News

- You should know by now :-)

What is PyPy?

- Past EuroPython talks:
  - ▶ **2004**: PyPy
  - ▶ **2005**: PyPy as a compiler
  - ▶ **2006**: An introduction to PyPy, PyPy architecture session, What can PyPy do for you
  - ▶ **2007**: PyPy 1.0 and Beyond, PyPy Python Interpreter(s) Features, PyPy: Why and how did it (not) work?
  - ▶ **2008**: PyPy for the rest of us, PyPy status talk
  - ▶ **2009** PyPy: Complete and Fast
  - ▶ **2010**: PyPy 1.3: Status and News

- You should know by now :-)

What is PyPy?

- Past EuroPython talks:
  - **2004**: PyPy
  - **2005**: PyPy as a compiler
  - **2006**: An introduction to PyPy, PyPy architecture session, What can PyPy do for you
  - **2007**: PyPy 1.0 and Beyond, PyPy Python Interpreter(s) Features, PyPy: Why and how did it (not) work?
  - **2008**: PyPy for the rest of us, PyPy status talk
  - **2009** PyPy: Complete and Fast
  - **2010**: PyPy 1.3: Status and News

- You should know by now :-)

What is PyPy? (seriously)

- PyPy
  - ▸ started in 2003
  - ▸ Open Source, partially funded by EU and others
  - ▸ framework for fast dynamic languages
  - ▸ **Python implementation**
- as a Python dev, you care about the latter

PyPy 1.5

- Released on 30 April, 2011
- Python 2.7.1
- The most compatible alternative to CPython
- Most programs just work
- (C extensions might not)

- **fast**

PyPy 1.5

- Released on 30 April, 2011
- Python 2.7.1
- The most compatible alternative to CPython
- Most programs just work
- (C extensions might not)

- **fast**

PyPy features

- JIT
  - automatically generated
  - complete/correct by construction
  - multiple backends: x86-32, x86-64, ARM

- Stackless
  - not yet integrated with the JIT (in-progress)

- cpyext
  - CPython C-API compatibility layer
  - not always working
  - often working: wxPython, PIL, cx_Oracle, mysqldb, pycairo, ...

- compact instances (as using `__slots__`)

PyPy features

- JIT
  - automatically generated
  - complete/correct by construction
  - multiple backends: x86-32, x86-64, ARM

- Stackless
  - not yet integrated with the JIT (in-progress)

- cpyext
  - CPython C-API compatibility layer
  - not always working
  - often working: wxPython, PIL, cx_Oracle, mysqldb, pycairo, ...

- compact instances (as using `__slots__`)

PyPy features

- JIT
  - automatically generated
  - complete/correct by construction
  - multiple backends: x86-32, x86-64, ARM

- Stackless
  - not yet integrated with the JIT (in-progress)

- cpyext
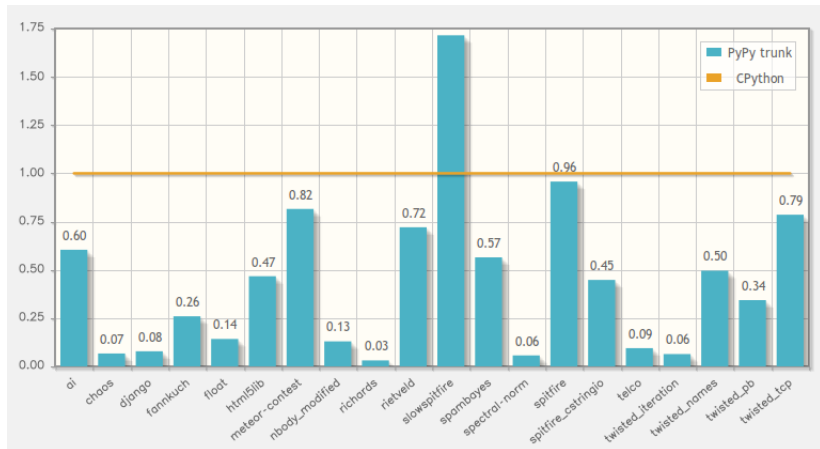 - CPython C-API compatibility layer
  - not always working
  - often working: wxPython, PIL, cx_Oracle, mysqldb, pycairo, ...

- compact instances (as using __slots__)

PyPy features

- JIT
  - automatically generated
  - complete/correct by construction
  - multiple backends: x86-32, x86-64, ARM

- Stackless
  - not yet integrated with the JIT (in-progress)

- cpyext
  - CPython C-API compatibility layer
  - not always working
  - often working: wxPython, PIL, cx_Oracle, mysqldb, pycairo, ...
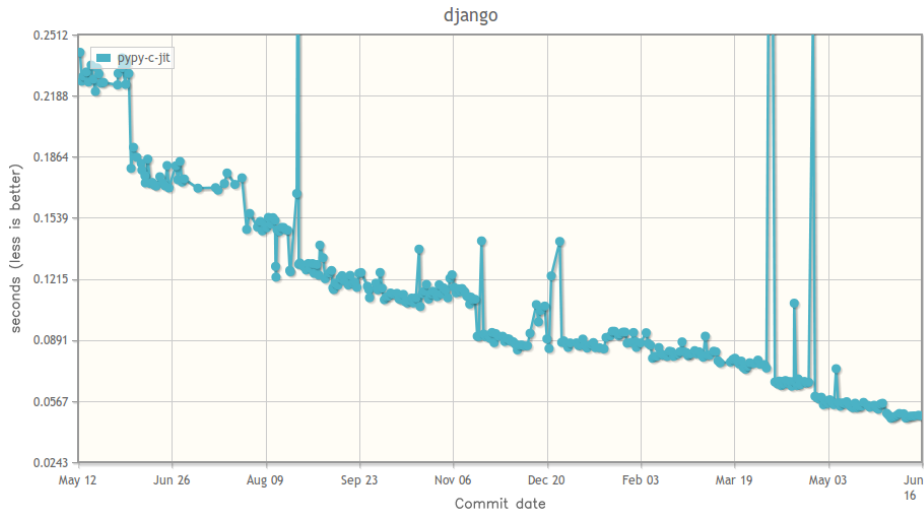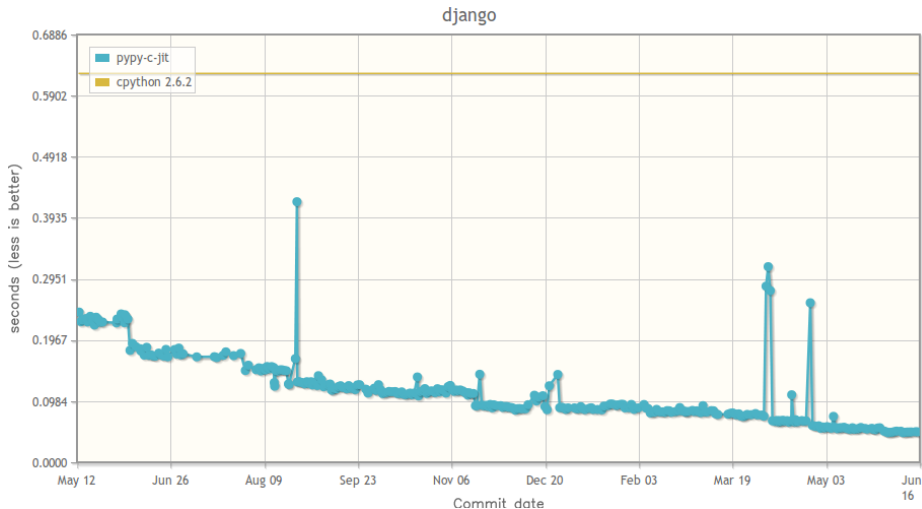
- compact instances (as using `__slots__`)

Speed

Improvements in the past year

django

Compare to CPython

django

Real world use case (1)

- LWN's gitdm
  - http://lwn.net/Articles/442268/
  - data mining tool
  - reads the output of `git log`
  - generate kernel development statistics

- Performance
  - CPython: 63 seconds
  - PyPy: **21 seconds**

lwn.net

[...] PyPy is ready for prime time; it implements the (Python 2.x) language faithfully, and it is fast.

Real world use case (1)

- LWN's gitdm
  - http://lwn.net/Articles/442268/
  - data mining tool
  - reads the output of `git log`
  - generate kernel development statistics

- Performance
  - CPython: 63 seconds
  - PyPy: **21 seconds**

lwn.net

[...] PyPy is ready for prime time; it implements the (Python 2.x) language faithfully, and it is fast.

Real world use case (1)

- LWN's gitdm
  - http://lwn.net/Articles/442268/
  - data mining tool
  - reads the output of `git log`
  - generate kernel development statistics

- Performance
  - CPython: 63 seconds
  - PyPy: **21 seconds**

lwn.net

> *[...] PyPy is ready for prime time; it implements the (Python 2.x) language faithfully, and it is fast.*

Real world use case (2)

- **MyHDL**: VHDL-like language written in Python
  - http://www.myhdl.org/doku.php/performance
  - (now) competitive with "real world" VHDL and Verilog simulators

myhdl.org

[...] the results are spectacular. By simply using a different interpreter, our simulations run 6 to 12 times faster.

Real world use case (2)

- **MyHDL**: VHDL-like language written in Python
  - http://www.myhdl.org/doku.php/performance
  - (now) competitive with "real world" VHDL and Verilog simulators

myhdl.org

[...] the results are spectacular. By simply using a different interpreter, our simulations run 6 to 12 times faster.

Real world use case (3)

- Translating PyPy itself
- Huge, complex piece of software
- All possible (and impossible :-)) kinds of dynamic and metaprogrammig tricks
- ~2.5x faster with PyPy
- (slow warm-up phase, though)
- Ouroboros!

Real world use case (4)

- Your own application
- Try PyPy, it might be worth it

Not convinced yet?

Real time edge detection

```python
def sobeldx(img):
    res = img.clone(typecode='d')
    for p in img.pixeliter():
        res[p] = (-1.0 * img[p + (-1,-1)] +
                   1.0 * img[p + ( 1,-1)] +
                  -2.0 * img[p + (-1, 0)] +
                   2.0 * img[p + ( 1, 0)] +
                  -1.0 * img[p + (-1, 1)] +
                   1.0 * img[p + ( 1, 1)]) / 4.0
    return res
...
...
```

Live demo

Is Python slow?

- ~~Python is slow~~
- Python is hard to optimize

- Huge stack of layers over the bare metal
- Abstraction has a cost (... or not?)

Is Python slow?

- ~~Python is slow~~
- Python is hard to optimize

- Huge stack of layers over the bare metal
- Abstraction has a cost (... or not?)

Is Python slow?

- ~~Python is slow~~
- Python is hard to optimize

- Huge stack of layers over the bare metal
- Abstraction has a cost (... or not?)

Python is complicated

How `a + b` works (simplified!):

- look up the method `__add__` on the type of a
- if there is one, call it
- if it returns NotImplemented, or if there is none, look up the method `__radd__` on the type of b
- if there is one, call it
- if there is none, or we get `NotImplemented` again, raise an exception `TypeError`

Python is a mess

How `obj.attr` or `obj.method()` works:

- ...

  - no way to write it down in just one slide

Python is a mess

How `obj.attr` or `obj.method()` works:

- ...

- no way to write it down in just one slide

Killing the abstraction overhead

Python

```python
class Point(object):

  def __init__(self, x, y):
    self.x = x
    self.y = y

  def __add__(self, q):
    if not isinstance(q, Point):
      raise TypeError
    x1 = self.x + q.x
    y1 = self.y + q.y
    return Point(x1, y1)

def main():
  p = Point(0.0, 0.0)
  while p.x < 2000.0:
    p = p + Point(1.0, 0.5)
  print p.x, p.y
```

C

```c
#include <stdio.h>

int main() {
    float px = 0.0, py = 0.0;
    while (px < 2000.0) {
        px += 1.0;
        py += 0.5;
    }
    printf("%f %f\n", px, py);
}
```

Killing the abstraction overhead

Python

```python
class Point(object):

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, q):
        if not isinstance(q, Point):
            raise TypeError
        x1 = self.x + q.x
        y1 = self.y + q.y
        return Point(x1, y1)

def main():
    p = Point(0.0, 0.0)
    while p.x < 2000.0:
        p = p + Point(1.0, 0.5)
    print p.x, p.y
```

C

```c
#include <stdio.h>

int main() {
    float px = 0.0, py = 0.0;
    while (px < 2000.0) {
        px += 1.0;
        py += 0.5;
    }
    printf("%f %f\n", px, py);
}
```

Pointless optimization techniques

```
#
for item in some_large_list:
    self.meth(item)
```

```
meth = self.meth
for item in some_large_list:
    meth(item)
```

```
def foo():
    res = 0
    for item in some_large_list:
        res = res + abs(item)
    return res
```

```
def foo(abs=abs):
    res = 0
    for item in some_large_list:
        res = res + abs(item)
    return res
```

```
#

[i**2 for i in range(100)]
```

```
from itertools import *
list(imap(pow, count(0),
          repeat(2, 100)))
```

```
for i in range(large_number):
    ...
```

```
for i in xrange(large_number):
    ...
```

```
class A(object):
    pass
```

```
class A(object):
    __slots__ = ['a', 'b', 'c']
```

Pointless optimization techniques

```python
#
for item in some_large_list:
    self.meth(item)
```

```python
meth = self.meth
for item in some_large_list:
    meth(item)
```

```python
def foo():
    res = 0
    for item in some_large_list:
        res = res + abs(item)
    return res
```

```python
def foo(abs=abs):
    res = 0
    for item in some_large_list:
        res = res + abs(item)
    return res
```

```python
#

[i**2 for i in range(100)]
```

```python
from itertools import *
list(imap(pow, count(0),
          repeat(2, 100)))
```

```python
for i in range(large_number):
    ...
```

```python
for i in xrange(large_number):
    ...
```

```python
class A(object):
    pass
```

```python
class A(object):
    __slots__ = ['a', 'b', 'c']
```

Pointless optimization techniques

```python
#
for item in some_large_list:
    self.meth(item)
```

```python
meth = self.meth
for item in some_large_list:
    meth(item)
```

```python
def foo():
    res = 0
    for item in some_large_list:
        res = res + abs(item)
    return res
```

```python
def foo(abs=abs):
    res = 0
    for item in some_large_list:
        res = res + abs(item)
    return res
```

```python
#

[i**2 for i in range(100)]
```

```python
from itertools import *
list(imap(pow, count(0),
          repeat(2, 100)))
```

```python
for i in range(large_number):
    ...
```

```python
for i in xrange(large_number):
    ...
```

```python
class A(object):
    pass
```

```python
class A(object):
    __slots__ = ['a', 'b', 'c']
```

Pointless optimization techniques

```python
#
for item in some_large_list:
    self.meth(item)
```

```python
meth = self.meth
for item in some_large_list:
    meth(item)
```

```python
def foo():
    res = 0
    for item in some_large_list:
        res = res + abs(item)
    return res
```

```python
def foo(abs=abs):
    res = 0
    for item in some_large_list:
        res = res + abs(item)
    return res
```

```python
#

[i**2 for i in range(100)]
```

```python
from itertools import *
list(imap(pow, count(0),
        repeat(2, 100)))
```

```python
for i in range(large_number):
    ...
```

```python
for i in xrange(large_number):
    ...
```

```python
class A(object):
    pass
```

```python
class A(object):
    __slots__ = ['a', 'b', 'c']
```

Pointless optimization techniques

```python
#
for item in some_large_list:
    self.meth(item)
```

```python
meth = self.meth
for item in some_large_list:
    meth(item)
```

```python
def foo():
    res = 0
    for item in some_large_list:
        res = res + abs(item)
    return res
```

```python
def foo(abs=abs):
    res = 0
    for item in some_large_list:
        res = res + abs(item)
    return res
```

```python
#

[i**2 for i in range(100)]
```

```python
from itertools import *
list(imap(pow, count(0),
          repeat(2, 100)))
```

```python
for i in range(large_number):
    ...
```

```python
for i in xrange(large_number):
    ...
```

```python
class A(object):
    pass
```

```python
class A(object):
    __slots__ = ['a', 'b', 'c']
```

Concrete example: ctypes

```python
import ctypes
libm = ctypes.CDLL('libm.so')
pow = libm.pow
pow.argtypes = [ctypes.c_double, ctypes.c_double]
pow.restype = ctypes.c_double
pow(2, 3) # <---
```

Layers and layers

CFuncPtrFast.__call__ (Python)

check that the cache is still valid

Layers and layers

CFuncPtrFast.__call__ (Python)

check that the cache is still valid

CFuncPtrFast._call_funcptr (Python)

some runtime checks (e.g. _flags_)

Layers and layers

CFuncPtrFast.__call__ (Python)

check that the cache is still valid

CFuncPtrFast._call_funcptr (Python)

some runtime checks (e.g. _flags_)

_ffi.FuncPtr.__call__ (RPython)

typecheck/unbox arguments, put them in raw C buffers

Layers and layers

CFuncPtrFast.__call__ (Python)

check that the cache is still valid

CFuncPtrFast._call_funcptr (Python)

some runtime checks (e.g. _flags_)

_ffi.FuncPtr.__call__ (RPython)

typecheck/unbox arguments, put them in raw C buffers

c_ffi_call (C) [libffi.so]

takes arguments from the raw C buffers

Layers and layers

CFuncPtrFast.__call__ (Python)

check that the cache is still valid

CFuncPtrFast._call_funcptr (Python)

some runtime checks (e.g. _flags_)

_ffi.FuncPtr.__call__ (RPython)

typecheck/unbox arguments, put them in raw C buffers

c_ffi_call (C) [libffi.so]

takes arguments from the raw C buffers

pow@0xf72de000 (C) [libm.so]

return 8

ctypes demo

Conclusion

- PyPy is fast
- mature
- stable
- abstractions for free!

- (I wonder why you all are still here instead of busy trying PyPy :-))
  - not all C extensions are supported (numpy anyone?)
  - too much memory (sometimes)

Conclusion

- PyPy is fast
- mature
- stable
- abstractions for free!

- (I wonder why you all are still here instead of busy trying PyPy :-))
  - not all C extensions are supported (numpy anyone?)
  - too much memory (sometimes)

How to help PyPy?

- Try it on your application
  - if it's slow, we want to know!
  - if it does not work, too :-)
  - if it works and it's fast, that as well
- Tell people about PyPy
- Contribute to PyPy! (it's not **that** hard :-))

- Give us money, to make PyPy better
  - donations
  - per feature contracts
  - consultancy (hire us to speed up your code)
  - support contracts

How to help PyPy?

- Try it on your application
  - if it's slow, we want to know!
  - if it does not work, too :-)
  - if it works and it's fast, that as well
- Tell people about PyPy
- Contribute to PyPy! (it's not **that** hard :-))

- Give us money, to make PyPy better
  - donations
  - per feature contracts
  - consultancy (hire us to speed up your code)
  - support contracts

Contacts, Q/A

- `http://pypy.org`
- blog: `http://morepypy.blogspot.com`
- mailing list: pypy-dev (at) python.org
- IRC: #pypy on freenode

?