PyPy JIT (not) for dummies

Antonio Cuni

PyCon Sei

April 17, 2015

About me

- PyPy core dev
- `pdb++`, `fancycompleter`, ...
- Consultant, trainer
- `http://antocuni.eu`

What is PyPy

- Alternative, fast Python implementation
- Performance: JIT compiler, advanced GC
- STM: goodbye GIL
- PyPy 2.5.1 (2.7.8)
- Py3k as usual in progress (3.2.5 out, 3.3 in development)
- `http://pypy.org`

STM

- pypy-stm-2.5.1 is out
  - 64 bit Linux only
- no GIL!
- 25-40% slowdown for single core programs
  - still `7*0.75 = 5.25x` faster than CPython :)
- parallelism up to 4 threads
- concurrency slow-but-correct by default
  - compared to fast-but-buggy by using threads
- conflict detection
- TransactionQueue: parallelize your program without using threads!

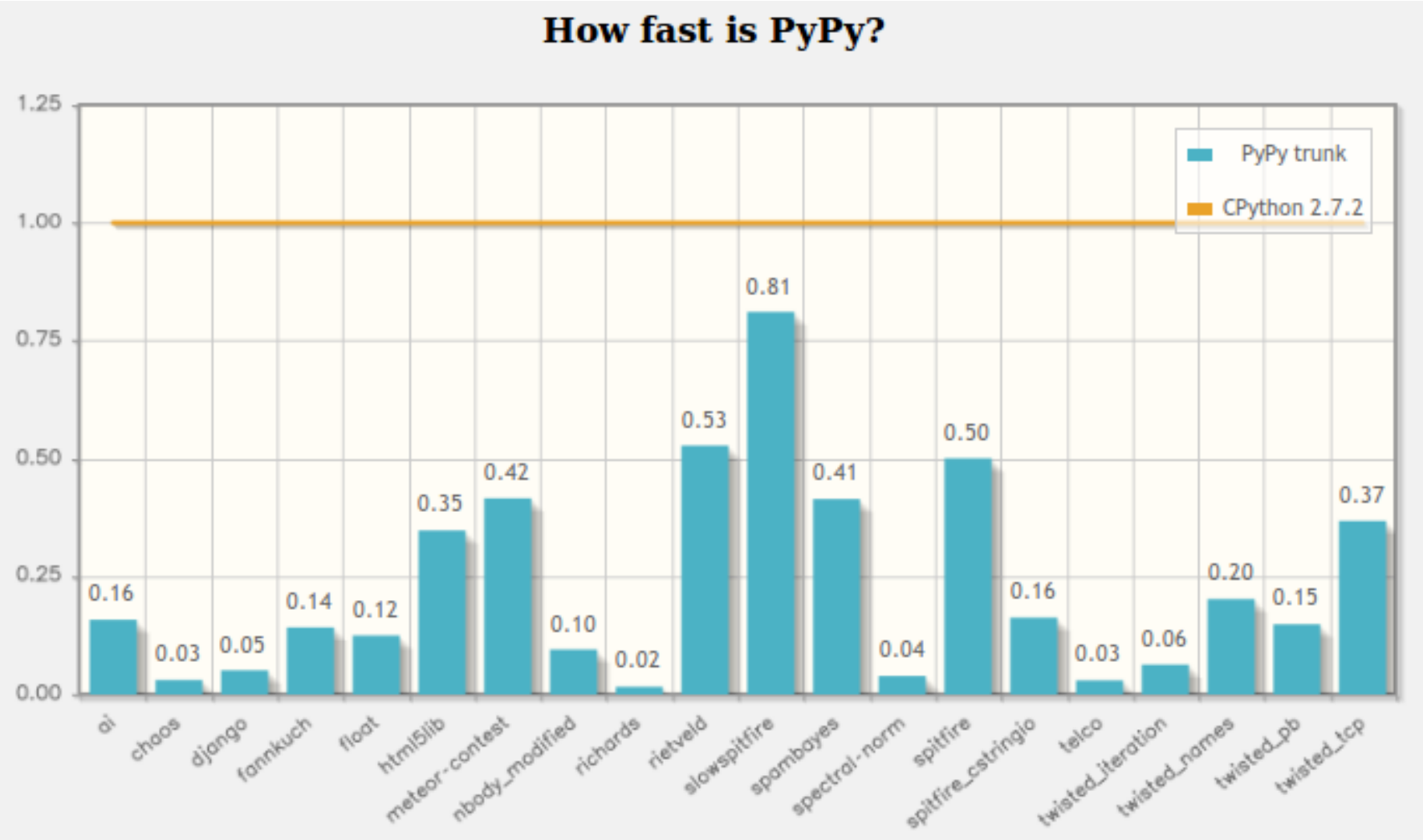
Extension modules

- CFFI: stable, mature and widely used
  - psycopg2cffi, lxml-cffi, pysdl2-cffi, etc.
  - should be used even for CPython-only projects!
- numpy:
  - support for linalg
  - support for pure Python, JIT friendly ufuncs
  - object dtype in-progress
- scipy: see next slide :)

Pymetabiosis

- embed CPython in PyPy
- import and use CPython modules in PyPy
- ALPHA status
- slow when passing arbitrary objects
- but fast for numpy arrays
- matplotlib and scipy works
- `https://github.com/rguillebert/pymetabiosis`

Speed: 7x faster than CPython

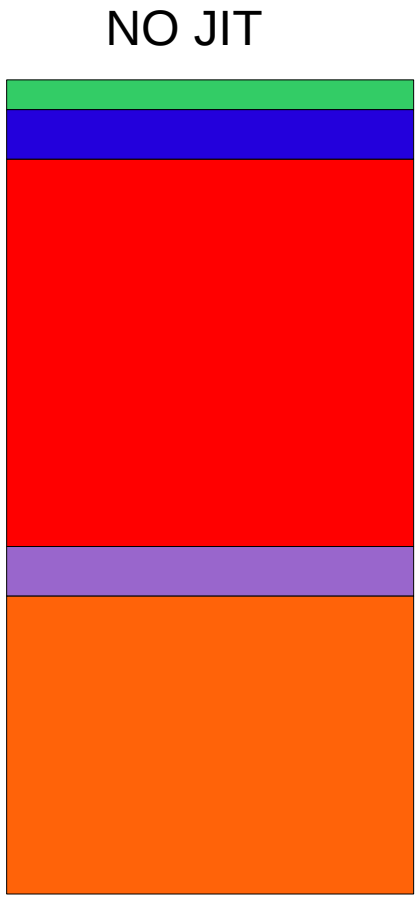


The JIT

```
def main():
    init()
    some_quick_code()
    for x in large_list:
        do_something(x)
    some_other_code()
    while condition():
        expensive_computation()
```

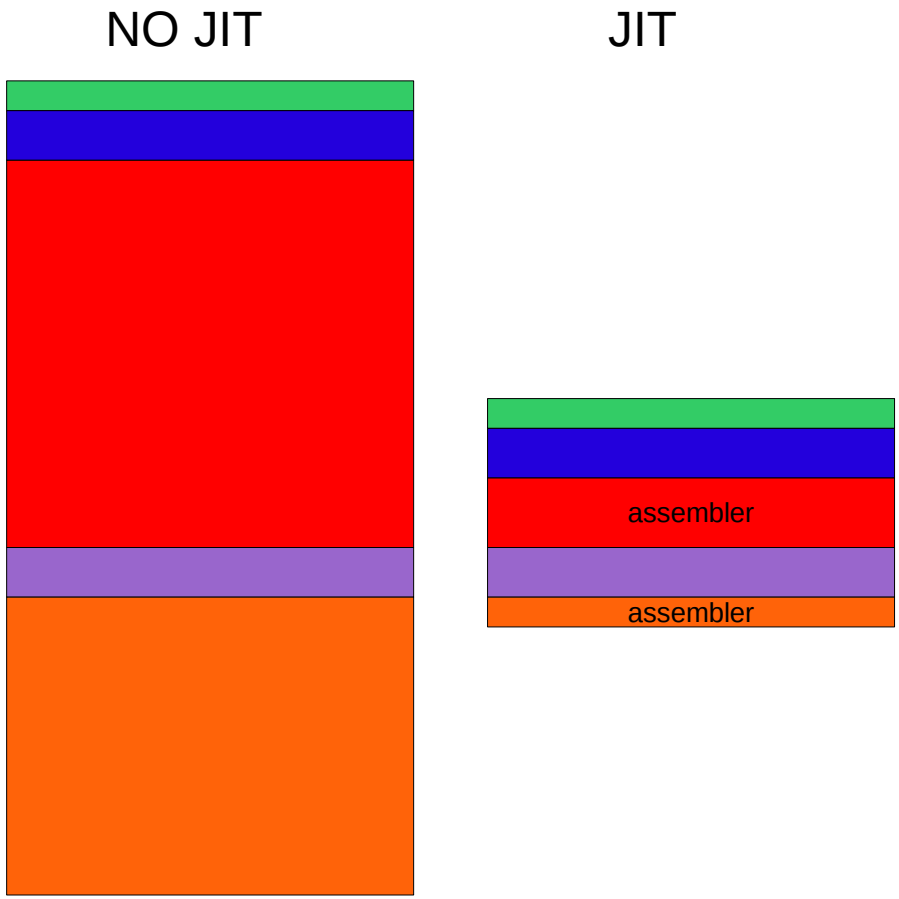
The JIT

```
def main():
    init()
    some_quick_code()
    for x in large_list:
        do_something(x)
    some_other_code()
    while condition():
        expensive_computation()
```



The JIT

```
def main():
    init()
    some_quick_code()
    for x in large_list:
        do_something(x)
    some_other_code()
    while condition():
        expensive_computation()
```



JIT overview

- Tracing JIT
  - detect and compile "hot" loops
  - (although not only loops)
- **Specialization**
- Precompute as much as possible
- Constant propagation
- Aggressive inlining

Specialization (1)

- `obj.foo()`
- which code is executed? (SIMPLIFIED)
  - lookup `foo` in obj.__dict__
  - lookup `foo` in obj.__class__
  - lookup `foo` in obj.__bases__[0], etc.
  - finally, execute `foo`
- without JIT, you need to do these steps again and again
- Precompute the lookup?

Specialization (2)

- pretend and assume that `obj.__class__` IS constant
  - "promotion"
- guard
  - check our assumption: if it's false, bail out
- now we can directly jump to `foo` code
  - ...unless `foo` is in `obj.__dict__`: GUARD!
  - ...unless `foo.__class__.__dict__` changed: GUARD!
- Too many guard failures?
  - Compile some more assembler!
- guards are cheap
  - out-of-line guards even more

Specialization (3)

- who decides what to promote/specialize for?
  - we, the PyPy devs :)
  - heuristics
- instance attributes are never promoted
- class attributes are promoted by default (with some exceptions)
- module attributes (i.e., globals) as well
- bytecode constants

Specialization trade-offs

- Too much specialization
  - guards fails often
  - explosion of assembler
- Not enough specialization
  - inefficient code

Virtuals

- Remove unnecessary allocations
- Remove unnecessary load/store

virtuals.py

```
res = 0
while res < 10000:
    obj = Foo(x, y, z)
    res += obj.x
```

Example

- Real world example
- Decoding binary messages
- Messages: strings of bytes

Point

```
struct Point {
    int x;
    int y;
    short color;
}
```

Example: low-level solution

decode0.py

```
P1 = '\x0c\x00\x00\x00"\x00\x00\x00\x07\x00\x00\x00'
P2 = '\x15\x00\x00\x00+\x00\x00\x00\x08\x00\x00\x00'

PLIST = [P1, P2] * 2000

def read_x(p):
    return struct.unpack_from('l', p, 0)[0]

def main():
    res = 0
    for p in PLIST:
        x = read_x(p)
        res += x
    print res
```

Example: low-level solution

decode0.py trace

```
debug_merge_point(1, 1, '<code object read_x> #0 LOAD_GLOBAL')
debug_merge_point(1, 1, '<code object read_x> #3 LOOKUP_METHOD')
debug_merge_point(1, 1, '<code object read_x> #6 LOAD_CONST')
debug_merge_point(1, 1, '<code object read_x> #9 LOAD_FAST')
debug_merge_point(1, 1, '<code object read_x> #12 LOAD_CONST')
debug_merge_point(1, 1, '<code object read_x> #15 CALL_METHOD')
+606: i91 = strlen(p88)
+609: i92 = int_lt(i91, 4)
guard_false(i92, descr=<Guard0xb3a14b20>)
+618: i93 = strgetitem(p88, 0)
+622: i94 = strgetitem(p88, 1)
+632: i95 = int_lshift(i94, 8)
+635: i96 = int_or(i93, i95)
+637: i97 = strgetitem(p88, 2)
+653: i98 = int_lshift(i97, 16)
+656: i99 = int_or(i96, i98)
+658: i100 = strgetitem(p88, 3)
+662: i101 = int_ge(i100, 128)
guard_false(i101, descr=<Guard0xb3a14ac0>)
+674: i102 = int_lshift(i100, 24)
+677: i103 = int_or(i99, i102)
```

Example: better API

decode1.py

```
class Field(object):
    def __init__(self, fmt, offset):
        self.fmt = fmt; self.offset = offset

class Message(object):
    def __init__(self, name, fields):
        self._name = name; self._fields = fields

    def read(self, buf, name):
        f = self._fields[name]
        return struct.unpack_from(f.fmt, buf, f.offset)[0]

Point = Message('Point', {'x': Field('l', 0),
                          'y': Field('l', 8),
                          'color': Field('i', 16)})

def main():
    res = 0
    for p in PLIST:
        x = Point.read(p, 'x')
        res += x
    print res
```

Example: better API

decode1.py trace (1)

```
debug_merge_point(1, 1, '<code object read> #34 CALL_METHOD')
p156 = getfield_gc_pure(p154, descr=<W_BytesObject.inst__value 8>)
i157 = getfield_gc_pure(p155, descr=<W_IntObject.inst_intval 8>)
p158 = new_with_vtable(-1228074336)
setfield_gc(p158, 0, descr=<CalcSizeFormatIterator.inst_totalsize 8>)
call(interpret_trampoline__v238___simple_call__function_i, p158, p156, ..
guard_no_exception(descr=<Guard0xb3a8cd00>)
i159 = getfield_gc(p158, descr=<CalcSizeFormatIterator.inst_totalsize 8>)
i160 = int_lt(i157, 0)
guard_false(i160, descr=<Guard0xb3a8ccd0>)
i161 = strlen(p141)
i162 = int_sub(i161, i157)
i163 = int_lt(i162, i159)
guard_false(i163, descr=<Guard0xb3a8cca0>)
i164 = int_ge(i159, 0)
guard_true(i164, descr=<Guard0xb3a8cc70>)
p165 = force_token()
p166 = new_with_vtable(-1228077368)
p167 = new_with_vtable(-1228077280)
p168 = new_with_vtable(-1228267680)
setfield_gc(p167, 1, descr=<FieldU rpython.rlib.buffer.Buffer.inst_readon
```

Example: better API

decode1.py trace (2)

```
p169 = new(descr=<SizeDescr 12>)
p170 = new_array_clear(0, descr=<ArrayP 4>)
p171 = new_with_vtable(-1229823908)
setfield_gc(p171, p145, descr=<JitVirtualRef.virtual_token 8>)
setfield_gc(p171, p145, descr=<JitVirtualRef.virtual_token 8>)
setfield_gc(p171, ConstPtr(null), descr=<FieldP JitVirtualRef.forced 12>)
setfield_gc(p51, p171, descr=<ExecutionContext.inst_topframeref 40>)
setfield_gc(p0, p165, descr=<PyFrame.vable_token 8>)
setfield_gc(p168, 1, descr=<Buffer.inst_readonly 8>)
setfield_gc(p168, p141, descr=<StringBuffer.inst_value 12>)
setfield_gc(p167, p168, descr=<SubBuffer.inst_buffer 12>)
setfield_gc(p167, i157, descr=<SubBuffer.inst_offset 16>)
setfield_gc(p167, i159, descr=<SubBuffer.inst_size 20>)
setfield_gc(p166, p167, descr=<UnpackFormatIterator.inst_buf 8>)
setfield_gc(p166, i159, descr=<UnpackFormatIterator.inst_length 12>)
setfield_gc(p166, 0, descr=<UnpackFormatIterator.inst_pos 16>)
setfield_gc(p169, 0, descr=<list.length 4>)
setfield_gc(p169, p170, descr=<list.items 8>)
setfield_gc(p166, p169, descr=<UnpackFormatIterator.inst_result_w 20>)
```

Example: better API

decode1.py trace (3)

```
call_may_force(interpret_trampoline__v628___simple_call__function_i), p16
guard_not_forced(descr=<Guard0xb3a8b9d0>)
guard_no_exception(descr=<Guard0xb3a8cc40>)
p172 = getfield_gc(p166, descr=<UnpackFormatIterator.inst_result_w 20>)
i173 = getfield_gc(p172, descr=<list.length 4>)
p174 = new_array_clear(i173, descr=<ArrayP 4>)
p175 = getfield_gc(p172, descr=<list.items 8>)
call(ll_arraycopy__arrayPtr_arrayPtr_Signed_Signed_Signed), p175, p174, 0
i176 = int_eq(i173, 2)
guard_false(i176, descr=<Guard0xb3a8cc10>)
```

Example: faster API

decode2.py

```
def Message(name, fields):
    class M(object):
        def read(self, buf, name):
            f = getattr(self, name)
            return struct.unpack_from(f.fmt, buf, f.offset)[0]

    for fname, f in fields.iteritems():
        setattr(M, fname, f)

    M.__name__ = name
    return M()

Point = Message('Point', {
    'x': Field('l', 0),
    'y': Field('l', 4),
    'color': Field('i', 8)
    })


 ...
 x = Point.read(p, 'x')
 ...
```

Example: faster API

decode2.py trace (3)

```
debug_merge_point(1, 1, '<code object read> #36 CALL_METHOD')
+670: i104 = strlen(p101)
+673: i105 = int_lt(i104, 4)
guard_false(i105, descr=<Guard0xb3afac10>)
+682: i106 = strgetitem(p101, 0)
+686: i107 = strgetitem(p101, 1)
+696: i108 = int_lshift(i107, 8)
+699: i109 = int_or(i106, i108)
+701: i110 = strgetitem(p101, 2)
+717: i111 = int_lshift(i110, 16)
+720: i112 = int_or(i109, i111)
+722: i113 = strgetitem(p101, 3)
+726: i114 = int_ge(i113, 128)
guard_false(i114, descr=<Guard0xb3afabe0>)
+738: i115 = int_lshift(i113, 24)
+741: i116 = int_or(i112, i115)
```

What happened?

- dict lookups inside classes are specialized
- decode1.py
  - `fields` is "normal data" and expected to change
  - one JIT code for **all** possible messages
- decode2.py
  - `fields` is expected to be constant
  - one JIT code for **each** message
- Behaviour is the same, different performance

Example: even better API :)

decode3.py

```
class Field(object):
    def __init__(self, fmt, offset):
        self.fmt = fmt
        self.offset = offset

    def __get__(self, obj, cls):
        return struct.unpack_from(self.fmt, obj._buf, self.offset)[0]

class Point(object):
    def __init__(self, buf):
        self._buf = buf

    x = Field('l', 0)
    y = Field('l', 4)
    color = Field('h', 8)

def main():
    res = 0
    for p in PLIST:
        p = Point(p)
        res += p.x
    print res
```

Contacts, Q&A

- `http://pypy.org`
- `http://morepypy.blogspot.com/`
- twitter: @antocuni
- Available for consultancy & training:
  - `http://antocuni.eu`
  - info@antocuni.eu
- Any question?