PyPy 1.4: Status and News

Maciej Fijalkowski

Cape Town PUG

November 27 2010

Outline

- What is PyPy?
- Overview of the PyPy JIT
- `cpyext`: load CPython extensions in PyPy!

What is PyPy?

- A very compliant Python interpreter
- with a just in time compiler
- ... other features

Speed

- cool charts

1.4 release

- 1.2: released on March 12th, 2010
  - Main theme: speed
  - JIT compiler
  - speed.pypy.org
- 1.3: released on June 26th, 2010
  - Stability: lot of bugfixes, thanks for the feedback :-)
  - More speed!
  - cpyext
- 1.4: release yesterday
  - even more speed and stability
  - jitted regexes, 64bit backend

What works on PyPy

- Pure Python modules should Just Work (TM)
  - django trunk
  - twisted, nevow
  - pylons
  - bittorrent
  - ...
- lot of standard modules
  - *__builtin__ __pypy__ _codecs _lsprof _minimal_curses _random _rawffi _socket _sre _weakref bz2 cStringIO crypt errno exceptions fcntl gc itertools marshal math md5 mmap operator parser posix pyexpat select sha signal struct symbol sys termios thread time token unicodedata zipimport zlib*
  - *array binascii cPickle cmath collections ctypes datetime functools grp md5 pwd pyexpat sha sqlite3 syslog*
  - *ctypes*

What works on PyPy

- Pure Python modules should Just Work (TM)
  - django trunk
  - twisted, nevow
  - pylons
  - bittorrent
  - ...

- lot of standard modules
  - *__builtin__ __pypy__ _codecs _lsprof _minimal_curses _random _rawffi _socket _sre _weakref bz2 cStringIO crypt errno exceptions fcntl gc itertools marshal math md5 mmap operator parser posix pyexpat select sha signal struct symbol sys termios thread time token unicodedata zipimport zlib*
  - *array binascii cPickle cmath collections ctypes datetime functools grp md5 pwd pyexpat sha sqlite3 syslog*

  - *ctypes*

What does not work on PyPy

- Pure Python modules should Just Work (TM)
  - ... unless they don't :-)
- Programs that rely on CPython-specific behavior
  - refcounting: `open('xxx', 'w').write('stuff')`
  - non-string keys in dict of types (try it!)
  - exact naming of a list comprehension variable
  - exact message matching in exception catching code
  - ...
- Extension modules
  - try cpyext!

What does not work on PyPy

- Pure Python modules should Just Work (TM)
  - ... unless they don't :-)
- Programs that rely on CPython-specific behavior
  - refcounting: `open('xxx', 'w').write('stuff')`
  - non-string keys in dict of types (try it!)
  - exact naming of a list comprehension variable
  - exact message matching in exception catching code
  - ...
- Extension modules
  - try cpyext!

What does not work on PyPy

- Pure Python modules should Just Work (TM)
  - ... unless they don't :-)
- Programs that rely on CPython-specific behavior
  - refcounting: `open('xxx', 'w').write('stuff')`
  - non-string keys in dict of types (try it!)
  - exact naming of a list comprehension variable
  - exact message matching in exception catching code
  - ...
- Extension modules
  - try cpyext!

What does not work on PyPy

- Pure Python modules should Just Work (TM)
  - ... unless they don't :-)
- Programs that rely on CPython-specific behavior
  - refcounting: `open('xxx', 'w').write('stuff')`
  - non-string keys in dict of types (try it!)
  - exact naming of a list comprehension variable
  - exact message matching in exception catching code
  - ...
- Extension modules
  - try cpyext!

Mandelbrot demo XXX maybe

Part 2: Just-in-Time compilation

Snakes never crawled so fast

Overview of implementations

- CPython
- Stackless
- Psyco
- Jython
- IronPython
- PyPy

Features

- it just works
- it may give good speed-ups (better than Psyco)
- it may have a few bugs left (Psyco too)
- it is not a hack (unlike Psyco)
- PyPy also has excellent memory usage
  - half that of CPython for a program using several hunderds MBs

Features

- it just works
- it may give good speed-ups (better than Psyco)
- it may have a few bugs left (Psyco too)
- it is not a hack (unlike Psyco)
- PyPy also has excellent memory usage
  - half that of CPython for a program using several hunderds MBs

Features

- it just works
- it may give good speed-ups (better than Psyco)
- it may have a few bugs left (Psyco too)
- it is not a hack (unlike Psyco)

- PyPy also has excellent memory usage
  - half that of CPython for a program using several hunderds MBs

What is a JIT

- CPython compiles the program source into bytecodes
- without a JIT, the bytecodes are then interpreted
- with a JIT, the bytecodes are further translated to machine code (assembler)

What is a JIT (2)

The translation can be:

- syntactic: translate the whole functions into machine code
    - "the obvious way"
    - e.g. Pyrex/Cython
    - not good performance, or needs tricks
- semantic: translate bits of the function just-in-time
    - only used parts
    - exploit runtime information (e.g. types)
    - Psyco, PyPy

What is a JIT (2)

The translation can be:

- syntactic: translate the whole functions into machine code
  - “the obvious way”
  - e.g. Pyrex/Cython
  - not good performance, or needs tricks
- semantic: translate bits of the function just-in-time
  - only used parts
  - exploit runtime information (e.g. types)
  - Psyco, PyPy

What is a mixed JIT

- start by interpreting normally
- find loops as they are executed
- turn them into machine code
- 80% of the time is spent in 20% of the code

Speed of the PyPy JIT

- most programs work quite well
- dynamizm that can't be reduced is not good
- meta-programming to the rescue
- reading frames or setting trace hooks has a bad effect
- generators are slower than they should be
- pure python is generally better than lower-level loops (like itertools)

Optimizations part 1 - no frames

- frame access is delayed
- if we want frames (`sys._getframe`, `sys.exc_info`, ...), it works by reading out of processor stack

Optimizations part 2 - malloc removal

- in python each object (including `int`) has to be allocated
- if we can prove that the object does not escape, we can ignore allocation
- in case we exit the JIT, we can create it on the fly
- much more powerful than proving compilation time

Optimizations part 3 - inlining

- very powerful optimization
- python calling is insane
- if we remove everything, we can even not allocate the frame
- huge wins

Optimizations part 4 - map dicts

- avoiding dict lookups
- work as good as `__slots__` memory-wise
- much faster when jitted

Part 3

cpyext

cpyext

- CPython extension modules in PyPy
- `pypy-c setup.py build`
- included in PyPy 1.3
- still beta
- 50% of the CPython API is supported
  - enough for 90% of extension modules

features

- C API written in Python!
- Testable on top of an interpreted py.py
- Written on top of the object space
- Source compatibility
    - PyString_AS_STRING is actually a function call (instead of a macro)

```
@cpython_api([PyObject], Py_ssize_t, error=-1)
def PyDict_Size(space, w_obj):
    return space.int_w(space.len(w_obj))
```

implementation

- It was not supposed to work!
  - different garbage collector
  - no “borrowed reference”
  - all the PyTypeObject slots
- not faster than python code!
- PyObject contains ob_type and ob_refcnt
  - The “abstract object interface” is used.
- Some objects contain more:
  - PyString_AsString() must keep the buffer alive at a fixed location
  - PyTypeObject exposes all its fields

The Reference Counting Issue

- pypy uses a moving garbage collector, starts with static roots to find objects.
- CPython objects don't move, and PyObject* can point to deallocated memory.
- cpyext builds PyObject as proxies to the "real" interpreter objects
- one dictionary lookup each time the boundary is crossed
- More tricks needed for borrowing references
    - The object lifetime is tied to its container.
    - "out of nothing" borrowed references are kept until the end of the current pypy->C call.

supported modules

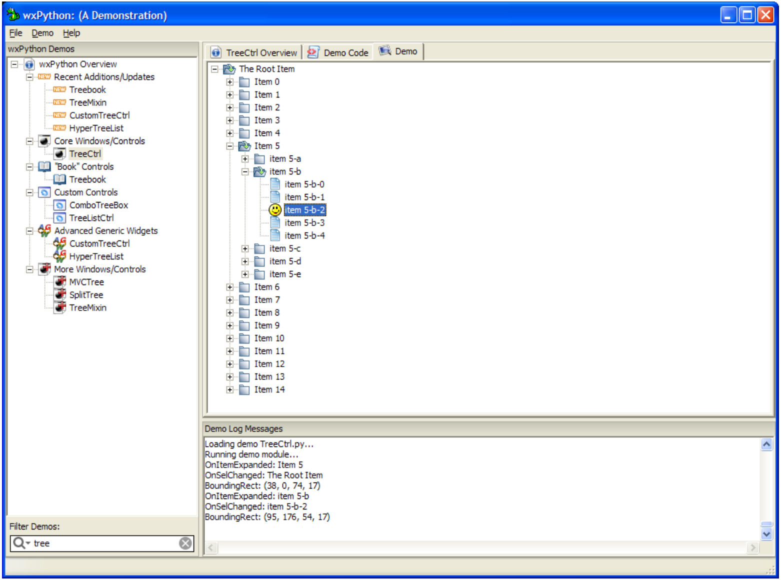
- Known to work (after small patches):
  - wxPython
  - _sre
  - PyCrypto
  - PIL
  - cx_Oracle
  - MySQLdb
  - sqlite

Why your module will crash

Likely:

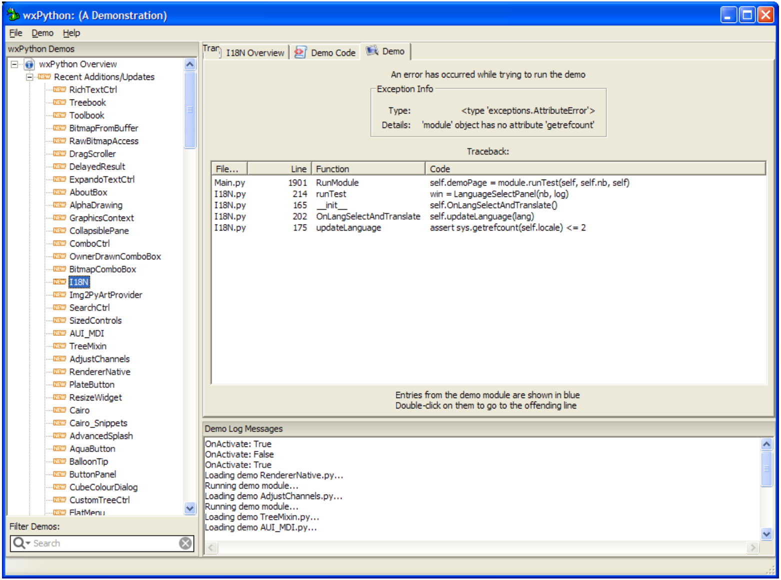
```
static PyObject *myException;

void init_foo() {
    myException = PyException_New(...);
    Py_AddModule(m, myException); // steals a reference
}

{
    PyErr_SetString(myException, "message"); // crash
}
```

wxPython on PyPy (1)



```
Loading demo TreeCtrl.py...
Running demo module...
OnItemExpanded: Item 5
OnSelChanged: The Root Item
BoundingRect: (38, 0, 74, 17)
OnItemExpanded: item 5-b
OnSelChanged: item 5-b-2
BoundingRect: (95, 176, 54, 17)
```

wxPython on PyPy (2)



An error has occurred while trying to run the demo

Exception Info

Type: <type 'exceptions.AttributeError'>

Details: 'module' object has no attribute 'getrefcount'

Traceback:

File...	Line	Function	Code
Main.py	1901	RunModule	self.demoPage = module.runTest(self, self.nb, self)
I18N.py	214	runTest	win = LanguageSelectPanel(nb, log)
I18N.py	165	__init__	self.OnLangSelectAndTranslate()
I18N.py	202	OnLangSelectAndTranslate	self.updateLanguage(lang)
I18N.py	175	updateLanguage	assert sys.getrefcount(self.locale) <= 2

Entries from the demo module are shown in blue
Double-click on them to go to the offending line

```
OnActivate: True
OnActivate: False
OnActivate: True
Loading demo RendererNative.py...
Running demo module...
Loading demo AdjustChannels.py...
Running demo module...
Loading demo TreeMixin.py...
Loading demo AUI_MDI.py...
```

Contact / Q&A

- Maciej Fijalkowski: fijall at gmail
- The #pypy IRC channel on freenode.net!
- Links:
  - PyPy: http://pypy.org/
  - PyPy speed center: http://speed.pypy.org/
  - Blog: http://morepypy.blogspot.com