

PyPy 1.2: snakes never crawled so fast

Antonio Cuni
Armin Rigo

Pycon Italia Qu4ttro

May 8 2010



Outline

- PyPy 1.2: what's new and status update
- Overview of the JIT
- Demo: how to use PyPy right now

Part 0: What is PyPy? :-)

- Python interpreter written in Python
 - Framework for developing dynamic languages
 - etc. etc.
-
- From the user point of view
 - An alternative to CPython
 - with more features!

Part 0: What is PyPy? :-)

- Python interpreter written in Python
- Framework for developing dynamic languages
- etc. etc.

- From the user point of view
- An alternative to CPython
- with more features!

Part 1

- What's new and status update

What's new in PyPy 1.2

- Released on March 12th, 2010
- Main theme: speed
- JIT compiler
- speed.pypy.org
- Ubuntu packages

Speed: PyPy vs Psyco

Benchmark	Time (s)	std dev	Current change	Trend	Times	(log2 scale)								
						4x	2x	equal	2x	4x	8x	16x		
twisted_iteration	0.0329	0.0003	0.10%	-0.64%	3.71									
chaos	0.0426	0.0327	4.59%	1.31%	3.67									
django	0.2834	0.0071	-0.48%	-1.05%	2.79									
nbody_modified	0.0810	0.0086	2.48%	0.44%	2.28									
float	0.0895	0.0285	-1.83%	-2.37%	2.23									
spectral-norm	0.0417	0.0060	-4.43%	-3.90%	2.17									
richards	0.0246	0.0018	0.32%	-0.23%	2.15									
fannkuch	0.3754	0.0122	1.64%	0.60%	1.84									
telco	0.3382	0.0155	-0.41%	-1.13%	1.70									
html5lib	10.1353	1.5026	2.37%	3.04%	1.59									
twisted_names	0.0063	0.0000	-0.68%	-1.19%	1.55									
twisted_web	0.1008	0.0079	-0.39%	-1.04%	1.33									
ai	0.4504	0.0078	-3.11%	1.08%	1.31									
rietveld	0.4946	0.2217	-0.87%	0.21%	1.29									
spitfire_cstringio	4.1262	0.1360	-7.40%	-1.18%	1.27									
twisted_pb	0.0313	0.0003	-0.67%	-2.43%	1.00									
spitfire	7.1760	0.1540	-0.82%	2.35%	0.59									
twisted_tcp	1.6573	0.0176	-2.88%	-1.52%	0.59									
meteor-contest	0.3509	0.0135	-2.62%	-2.47%	0.49									
slowspitfire	1.2217	0.1532	-7.55%	-0.77%	0.38									
spambayes	0.8845	0.0646	-3.65%	-0.24%	0.34									

Speed: Demo

- Django application
- Mandelbrot fractal
 - ▶ fished randomly on the net :-)
- Run both on CPython and PyPy
 - ▶ django trunk!

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
 - ▶ reccounting: `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
 - ▶ really?
 - ▶ drum roll...

What does not work on PyPy

- Pure Python modules should Just Work (TM)
 - ▶ ... unless they don't :-)
- Programs that rely on CPython-specific behavior
 - ▶ reccounting: `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
 - ▶ really?
 - ▶ drum roll...

What does not work on PyPy

- Pure Python modules should Just Work (TM)
 - ▶ ... unless they don't :-)
- Programs that rely on CPython-specific behavior
 - ▶ reccounting: `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
 - ▶ really?
 - ▶ drum roll...

What does not work on PyPy

- Pure Python modules should Just Work (TM)
 - ▶ ... unless they don't :-)
- Programs that rely on CPython-specific behavior
 - ▶ reccounting: `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
 - ▶ really?
 - ▶ drum roll...

What does not work on PyPy

- Pure Python modules should Just Work (TM)
 - ▶ ... unless they don't :-)
- Programs that rely on CPython-specific behavior
 - ▶ reccounting: `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
 - ▶ really?
 - ▶ drum roll...

cpyext

- CPython extension modules in PyPy
- `pypy-c setup.py build`
- still beta
- not 100% of CPython API is supported
- not included in PyPy 1.2
- Known to work:
 - ▶ wxPython (after a patch)
 - ▶ `_sre`
 - ▶ PyCrypto
 - ▶ PIL

wxPython on PyPy (1)

The screenshot shows the wxPython: (A Demonstration) application window. The title bar includes the text "wxPython: (A Demonstration)" and standard window control buttons. The menu bar contains "File", "Demo", and "Help".

The main area is divided into two panes:

- Left Pane (wxPython Demos):** A tree view listing various demo components. The "Core Windows/Controls" folder is expanded, showing "TreeCtrl" selected.
- Right Pane (TreeCtrl Overview):** A tree view showing the structure of the selected "TreeCtrl". The root is "The Root Item", which contains "Item 0" through "Item 14". "Item 5" is expanded to show sub-items "Item 5-a" through "Item 5-e". "Item 5-b" is further expanded to show "item 5-b-0" through "item 5-b-4". "item 5-b-2" is selected and highlighted with a blue background, and it contains a yellow smiley face icon.

At the bottom of the window is a "Demo Log Messages" pane with the following text:

```
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)
```

At the bottom left, there is a "Filter Demos:" label and a search box containing the text "tree".

wxPython on PyPy (2)

The screenshot shows the wxPython: (A Demonstration) application window. The left sidebar contains a tree view of the demo modules, with 'I18N' selected. The main area displays an error message and a traceback. The error message states: "An error has occurred while trying to run the demo. Exception Info: Type: <type 'exceptions.AttributeError'> Details: 'module' object has no attribute 'getrefcount'". The traceback table shows the following details:

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

Below the traceback, a note states: "Entries from the demo module are shown in blue. Double-click on them to go to the offending line." At the bottom, the "Demo Log Messages" panel shows the following log entries:

```
OnActivate: True
OnActivate: False
OnActivate: True
Loading demo RendererNative.py...
Running demo module...
Loading demo AdjustChannels.py...
Running demo module...
Loading demo TreeMix.py...
Loading demo AUJ_MDI.py...
```

PyPy 1.2.1

- Coming soon
- Many bugfixes
 - ▶ 27 issues reported after release of 1.2
- beta version of cpyext

Part 2: Just-in-Time compilation

Snakes never crawled so fast

Overview of implementations

- CPython
- Stackless
- Psyco
- Jython
- IronPython
- PyPy (without and with JIT)
- Unladen Swallow

Demo

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 hundreds 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, Unladen Swallow
 - ▶ 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, Unladen Swallow
 - ▶ 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 tracing 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

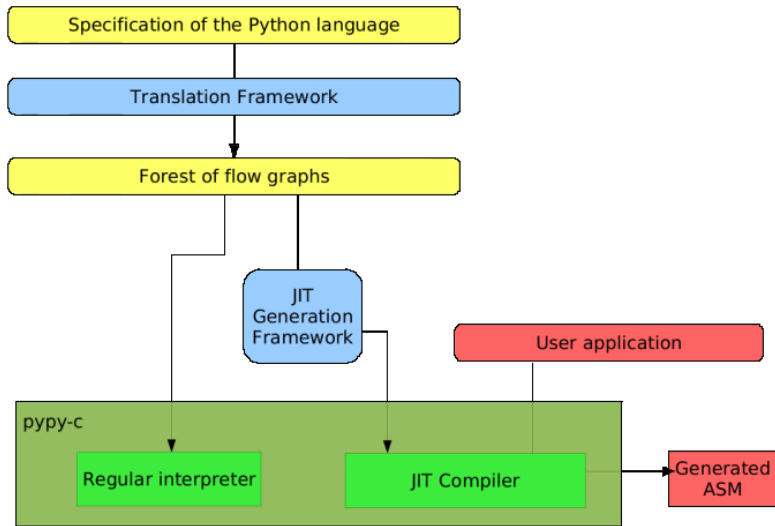
What is a tracing JIT (history)

- tracing assembler (Dynamo, ~2000)
- tracing Java (~2005)
- tracing JavaScript (~2008)
- PyPy is a “tracing JIT generator”

What is a tracing JIT (history)

- tracing assembler (Dynamo, ~2000)
- tracing Java (~2005)
- tracing JavaScript (~2008)
- PyPy is a “tracing JIT generator”

The architecture of PyPy



Speed of the PyPy JIT

Python programs that are, or are not, nicely handled by the JIT:

- loops, even across many calls, are nicely handled
- loops with very many taken paths are not
 - ▶ e.g. Python programs that look like interpreters
 - ▶ typical in tracing JITs
- bad support so far for generators and recursion

The optimizations we get

- != optimizations we *wrote* :-)
- removed frame handling
 - ▶ local variables are in CPU registers or on the C stack
 - ▶ but `sys._getframe()` works correctly
- “virtuals”: temporary objects are not constructed
 - ▶ `e = a + b + c + d`
 - ▶ and much more complex examples
- attribute and method lookups, etc.

Example

```
def g(a, b):  
    if a < 5: # 2  
        return -1  
    return a - b # 3  
  
def f(x):  
    total = 0  
    # 1  
    for i in range(x):  
        d = g(i, x)  
        total += d # 4
```

```
ADD EAX, 1  
CMP EAX, EBX  
JNL <guard 1>  
CMP EAX, 0  
JL <guard 2>  
MOV ECX, EAX  
SUB ECX, EBX  
JO <guard 3>  
ADD EDX, ECX  
JO <guard 4>  
JMP
```

Example

```
def g(a, b):  
    if a < 5: # 2  
        return -1  
    return a - b # 3  
  
def f(x):  
    total = 0  
    # 1  
    for i in range(x):  
        d = g(i, x)  
        total += d # 4
```

```
ADD EAX, 1  
CMP EAX, EBX  
JNL <guard 1>  
CMP EAX, 0  
JL <guard 2>  
MOV ECX, EAX  
SUB ECX, EBX  
JO <guard 3>  
ADD EDX, ECX  
JO <guard 4>  
JMP
```

Practical results

- fast :-)
- so far, x86-32 only
- relatively easy to maintain (or port to x86-64, etc.)
- reminder: works transparently for *any* Python code
 - ▶ or any language (Prolog JIT :-) at PPDP 2010)
- viable alternative to CPython

Part 3

How to use PyPy right now

Mandelbrot demo

- Works purely on PyPy
- Not always the case
 - ▶ missing extension modules (cpyext mitigates the problem)
 - ▶ libraries that rely on CPython details
 - ▶ ...
- clear performance-critical part

CPython and PyPy side by side

- CPython: runs the main application
- PyPy: subprocess, runs only the hotspots
- How do they communicate?
- execnet
 - ▶ **The Ring of Python**, Holger Krekel, 9:00
 - ▶ oups, too late :-)

Rendering (1)

Mandelbrot

```
def render(request):  
    w = int(request.GET.get('w', 320))  
    h = int(request.GET.get('h', 240))  
  
    from py_mandel import mandelbrot  
    img = mandelbrot(w, h)  
  
    return HttpResponse(img, content_type="image/bmp")
```

Rendering (2)

Mandelbrot on PyPy

```
def pypy_render(request):  
    w = int(request.GET.get('w', 320))  
    h = int(request.GET.get('h', 240))  
  
    channel = pypy.remote_exec("""  
from py_mandel import mandelbrot  
w, h = channel.receive()  
img = mandelbrot(w, h)  
channel.send(img)  
""")  
    channel.send((w, h))  
    img = channel.receive()  
  
    return HttpResponse(img, content_type="image/bmp")
```

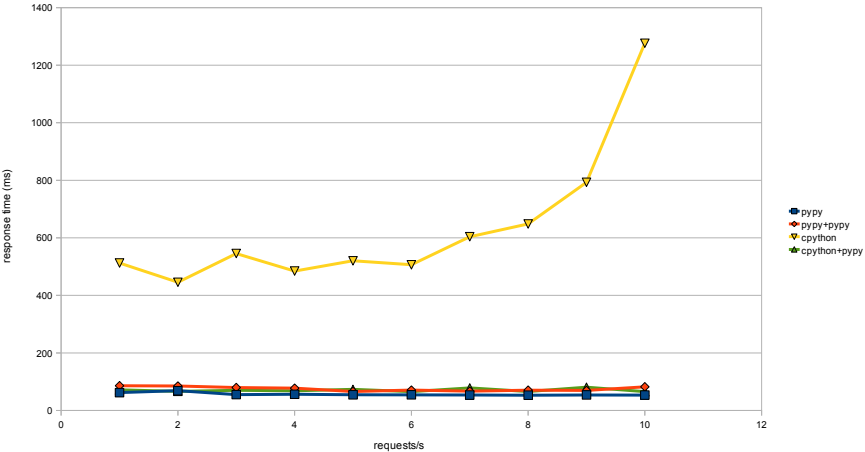
execnet setup

At startup

```
import execnet
mygroup = execnet.Group()
pypy = mygroup.makegateway("popen//python=pypy-c")
pypy.remote_exec("""
import sys
import os
os.chdir("mandelbrot")
sys.path.insert(0, '')
""")
```

Demo

Benchmarks



Contact / Q&A

- Antonio Cuni: at <http://merlinux.eu>
- Armin Rigo: arigo (at) tunes.org
- Links:
 - ▶ PyPy: <http://pypy.org/>
 - ▶ PyPy speed center: <http://speed.pypy.org/>
 - ▶ Blog: <http://morepypy.blogspot.com>