PyPy

Maciej Fijałkowski
Alex Gaynor
Armin Rigo

Google

7 March 2011

Introduction

- The PyPy project (1): a framework in which to write interpreters for complicated dynamic languages
- The PyPy project (2): a Python interpreter, supporting the complete Python 2.7

CPython and PyPy

CPython and PyPy

- Two implementations
- Two interpreters
- CPython is written in C, PyPy is written in Python
- PyPy tries to be equivalent to CPython

...and Jython and IronPython

- Jython: Python for the Java VM
- IronPython: Python for .NET
- Both try to integrate well with their VM

What is PyPy

- A project started in 2003
- An Open Source effort of volunteers
- With some funding support: 2 years from the European Union (2005-2007), and now from Germany and Sweden (2010-2011).

What is PyPy

- Test-driven development
- Now contains about 200 KLoC, and 150 KLoc of tests

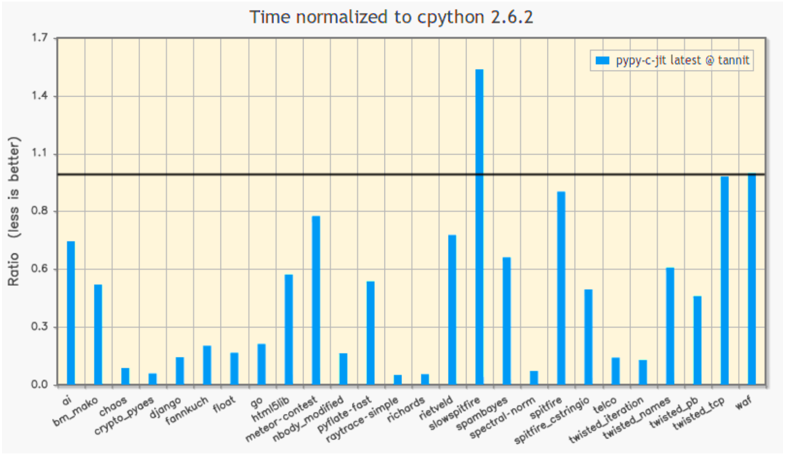
What is the point of PyPy?

- CPython is older, it's the "official" version
- PyPy is just a replacement, so why?
- Moreover PyPy is not quite complete (e.g. C extension modules are only partially supported)

Speed

- First answer: PyPy is faster, and may use less memory
- ...or at least, it is “often” the case

http://speed.pypy.org/



Why not extend CPython?

- adding a better GC or a JIT to CPython is very hard
- it has been tried

And (optionally) extra features

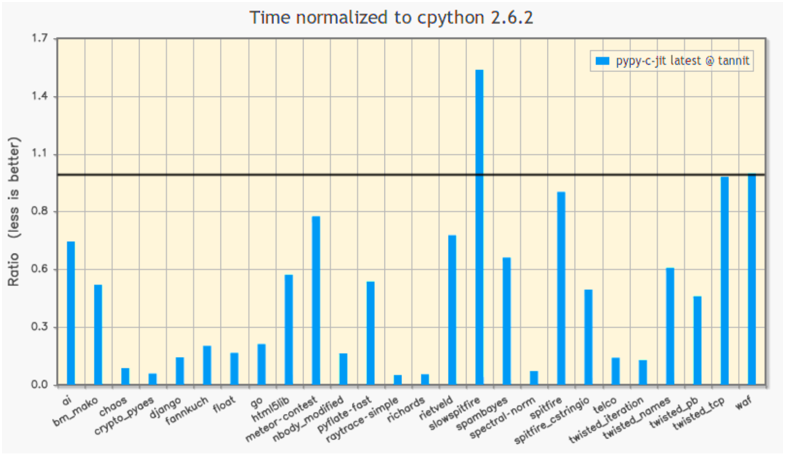
- "Stackless"
- Non-Python interpreters
- and many smaller experiments
- it is a better experimentation platform than CPython

Multi-threading

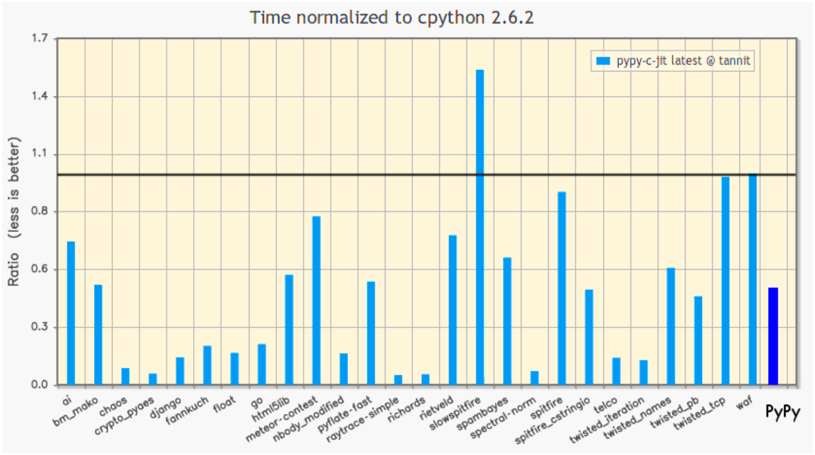
- Bad support on CPython (GIL)
- PyPy has no answer to this question (there is also a GIL)

PyPy for the user

Speed


Time normalized to cpython 2.6.2
pypy-c-jit latest @ tannit
Ratio (less is better)
1.7
1.4
1.1
0.8
0.6
0.3
0.0
ai
bm_mako
chaos
crypto_pyaes
django
fannkuch
float
go
html5lib
meteor-contest
nbody_modified
pyflate-fast
raytrace-simple
richards
rietveld
slowspitfire
spambayes
spectral-norm
spitfire
spitfire_cstringio
telco
twisted_iteration
twisted_names
twisted_pb
twisted_tcp
waf

Speed (2)


Time normalized to cpython 2.6.2
pypy-c-jit latest @ tannit
Ratio (less is better)
1.7
1.4
1.1
0.8
0.6
0.3
0.0
ai
bm_mako
chaos
crypto_pyaes
django
fannkuch
float
go
html5lib
meteor-contest
nbody_modified
pyflate-fast
raytrace-simple
richards
rietveld
slowspitfire
spambayes
spectral-norm
spitfire
spitfire_cstringio
telco
twisted_iteration
twisted_names
twisted_pb
twisted_tcp
waf
PyPy

Memory usage

- Depends on the use case
- Much better than CPython for instances of classes with no __slots__
- On running PyPy's translation toolchain on 32-bits: 1.7GB with PyPy (including the JIT machine code), versus 1.2GB with CPython
- Experimental support for 32-bit “compact pointers” on 64-bit platforms

Just-in-Time Compilation

- Tracing JIT, like TraceMonkey
- Complete by construction
- Supports Intel x86, amd64, and soon ARM

Compatibility

- "Full" compatibility with CPython
- More so than, say, Jython or IronPython
- Main difference: Garbage Collection is not refcounting (because we could get much better GCs) --- so __del__ methods are not called immediately and predictively
- Apart from that, it is really 99.99% compatible

Stackless Python

- Supports Stackless Python (microthreads)
- In-progress: not integrated with the JIT so far

CPyExt

- A layer that integrates existing CPython C extension modules
- Does not support all the details of the CPython C API
- For some extension modules, we can have a performance issue
- Work in progress

CPyExt works “often”

- wxPython
- PIL
- Boost
- cx_Oracle
- mysqldb
- pycairo

Using CPyExt

- The C sources need recompiling
- Sadly, they often contain a few details to fix
- (typically, bad usage of reference counts)

Other ways to use C libraries

- Use ctypes (it is soon going to be fast on top of PyPy). Example: pyexpat, sqlite3
- Or write it as an RPython module built into PyPy, but that's more involved
- More ways could be possible, given work (SWIG backend, Cython backend, C++ Reflex, etc...)

Conclusions & future

- we're already fast and getting faster
- our C extension story is not great, but we're working on it
- we're offering pypy consulting

Break

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

Just-in-Time Compiler

Short introduction to JITting

- run code with the interpreter
- observe what it does
- generate optimized machine code for commonly executed paths
- using runtime knowledge (types, paths taken)

Tracing JIT

- compiles one loop at a time
- generates linear code paths, recording what the interpreter did
- for each possible branch, generate a guard, that exits assembler on triggering
- if guard fails often enough, start tracing from the failure

Meta-Tracing in PyPy

- The explanation above assumes a tracing JIT for the full Python language
- Would need to be maintained whenever we change the Python version we support
- Instead, we have a "meta-tracing JIT"
- A very important point for us since we don't have a huge team to implement all Python semantics for the JIT
- We trace the python interpreter's main loop (running N times) interpreting a python loop (running once)

Tracing example

- we have cool tools!

Conclusion

- PyPy is a platform for writing efficient interpreters for dynamic languages
- http://pypy.org/
- http://speed.pypy.org/
- irc: `#pypy at freenode.net`