PyPy: A Fast and Compliant Python Implementation
An Introduction

David Schneider

STUPS - Institut für Informatik
Heinrich-Heine Universität Düsseldorf

January 29, 2014

The PyPy Project

- Python VM 2.7.3/3.2
- Just-in-Time compiler for Python
- Dynamic language toolkit
  - *RPython language and compilation/translation toolchain*
  - *PyPy Python VM, written in RPython*

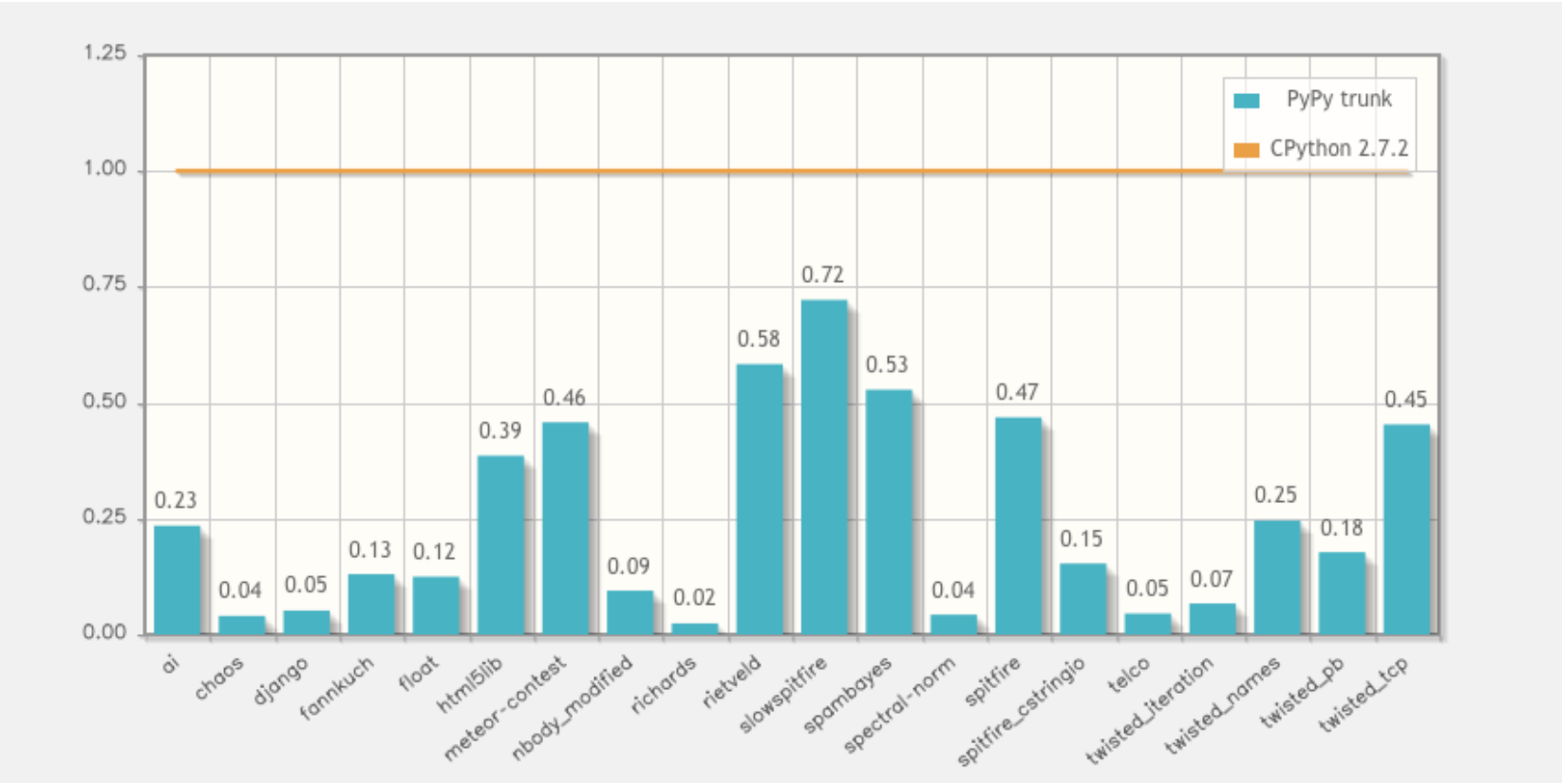
The PyPy Project

- Started in 2003
- Started as Python in Python
- Open Source, MIT Licensed
- Member of Software Freedom Conservancy

Topics

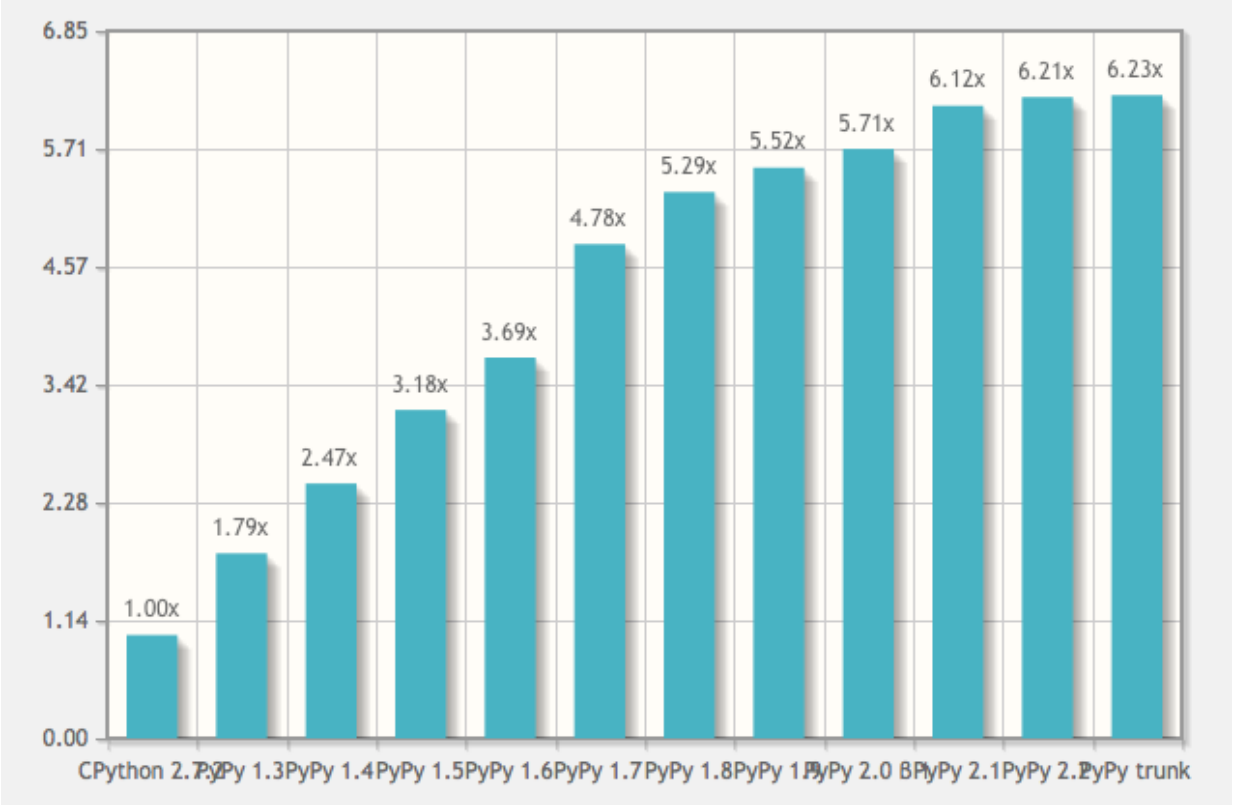
- JIT in PyPy
- Differences to CPython
- Interaction with C Code
- Status
  - *Numpy*
  - *General*
- Documentation
- Support

Motivation



`http://speed.pypy.org`

Motivation



`http://speed.pypy.org`

Just-in-Time Compiler

- Tracing JIT
- Profiling
- Tracing Hypothesis
- Record one path through loop
- Trace Optimization
  - *Aggressive inlining*
  - *Runtime Type Information*
  - *Remove the dynamic overhead, dispatching, etc.*
- Machine code generation

Traces, Loops and Bridges

- Traces: List of linear, recorded operations
- SSA based intermediate representation

- Loop is a trace recorded for a user-level loop

- Control-Flow divergence marked with guards
- Bridge is a trace recorded from a guard to a loop

Traces, Loops and Bridges

- Traces: List of linear, recorded operations
- SSA based intermediate representation
- Loop is a trace recorded for a user-level loop
- Control-Flow divergence marked with guards
- Bridge is a trace recorded from a guard to a loop

Traces, Loops and Bridges

- Traces: List of linear, recorded operations
- SSA based intermediate representation
- Loop is a trace recorded for a user-level loop
- Control-Flow divergence marked with guards
- Bridge is a trace recorded from a guard to a loop

JIT

Example

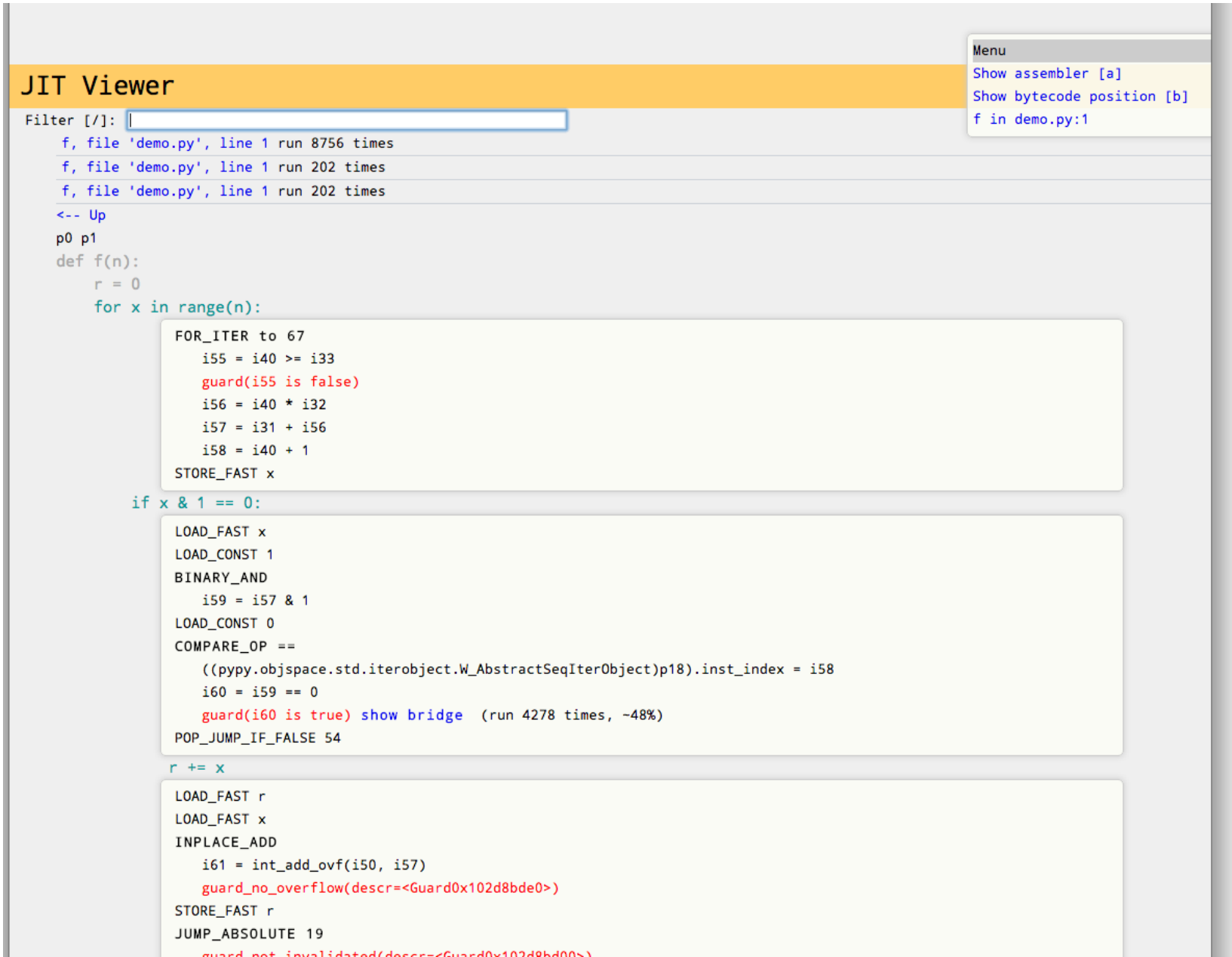
```
def f(n):
    r = 0
    for x in range(n):
        if x & 1 == 0:
            r += x
        else:
            r -= x
    return r
```

Trace (simplified)

Example

```
i59 = int_and(i57, 1)
i60 = int_eq(i59, 0)
guard_true(i60)
i61 = int_add_ovf(i50, i57)
guard_no_overflow()
jump(p0, p1, p3, p6, p7, p12, i61,
     i57, p18, i58, i33, i32, i31)
```

JIT Viewer


JIT Viewer
Filter [/]:
Menu
Show assembler [a]
Show bytecode position [b]
f in demo.py:1
f, file 'demo.py', line 1 run 8756 times
f, file 'demo.py', line 1 run 202 times
f, file 'demo.py', line 1 run 202 times
<-- Up
p0 p1
def f(n):
r = 0
for x in range(n):
FOR_ITER to 67
i55 = i40 >= i33
guard(i55 is false)
i56 = i40 * i32
i57 = i31 + i56
i58 = i40 + 1
STORE_FAST x
if x & 1 == 0:
LOAD_FAST x
LOAD_CONST 1
BINARY_AND
i59 = i57 & 1
LOAD_CONST 0
COMPARE_OP ==
((pypy.objspace.std.iterobject.W_AbstractSeqIterObject)p18).inst_index = i58
i60 = i59 == 0
guard(i60 is true) show bridge (run 4278 times, ~48%)
POP_JUMP_IF_FALSE 54
r += x
LOAD_FAST r
LOAD_FAST x
INPLACE_ADD
i61 = int_add_ovf(i50, i57)
guard_no_overflow(descr=<Guard0x102d8bde0>)
STORE_FAST r
JUMP_ABSOLUTE 19

Differences to CPython

- Garbage Collection
  - *Generational/tracing GC*
  - *Resource de-allocation*
- Some builtins written in Python
- Extensions modules (CPyExt)

Differences to CPython

- Garbage Collection
  - *Generational/tracing GC*
  - *Resource de-allocation*
- Some builtins written in Python
- Extensions modules (CPyExt)

CPyExt (calling Python from C/C++)

- Emulation of the CPython C-API
- Written in RPython
- Source compatible
- Incomplete, methods added as needed
- Recompilation of modules is required
- Accessing object internals will cause it to fail
- Emulation overhead

C extensions known to work with PyPy

`https://bitbucket.org/pypy/compatibility/wiki/CCompatible`

CPyExt (calling Python from C/C++)

- Emulation of the CPython C-API
- Written in RPython
- Source compatible
- Incomplete, methods added as needed
- Recompilation of modules is required
- Accessing object internals will cause it to fail
- Emulation overhead

C extensions known to work with PyPy

`https://bitbucket.org/pypy/compatibility/wiki/CCompatible`

Fast Interaction with C/C++ from Python

- cffi
- ctypes
- cppyy
- all three well-integrated with the JIT.

cffi

- Based on the LuaJIT FFI
- Recommended way to call C from PyPy
- Works on CPython and PyPy
- Version 1.0 (hopefully soon)
- JIT Integration, can remove dynamic call overhead (libffi)
- Concept: No 3rd language for the API
- ABI/API level bindings

cffi

Example

```
>>> from cffi import FFI
>>> ffi = FFI()
>>> ffi.cdef("""
...     int printf(const char *format, ...); #*
... """)
>>> C = ffi.dlopen(None)
>>> arg = ffi.new("char[]", "world")
>>> C.printf("hi there, %s!\n", arg)
hi there, world!
```

ctypes

- Mostly complete support
- Fast/slow paths for the JIT

cppyy: C++ bindings for PyPy

- Calling C++ from Python
- Based on gcc/Reflex
- clang/cling support next goal
- Runtime Python bindings from reflection information.
- developed and used by LHC collaborators

cppyy

Example

```
class MyClass {
public:
    MyClass(int i = -99) : m_myint(i) {}

    int GetMyInt() { return m_myint; }
    void SetMyInt(int i) { m_myint = i; }

public:
        int m_myint;
};
```

cppyy

Example

```
$ pypy-c
>>>> import cppyy
>>>> cppyy.load_reflection_info("libMyClassDict.so")
<CPPLibrary object at 0xb6fd7c4c>
>>>> myinst = cppyy.gbl.MyClass(42)
>>>> print myinst.GetMyInt()
42
>>>> myinst.SetMyInt(33)
>>>> print myinst.m_myint
33
>>>> myinst.m_myint = 77
>>>> print myinst.GetMyInt()
77
```

Numpy

- Ongoing effort
- Partial support
- Partial C-API support
- Missing:
  - *dtypes*
  - *datetime64*
- Current focus on completeness
- Fast array iteration (JIT)

Installation

```
pip install git+https://bitbucket.org/pypy/numpy
```

Numpy

- Status: Based on nightly builds
  - *http://buildbot.pypy.org/numpy-status/latest.html*

NumPyPy Status: how much of numpy can you use in pypy?

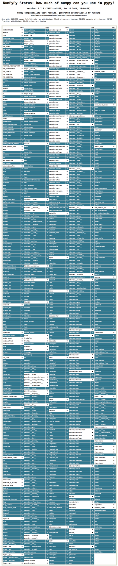
Version: 2.7.3 (79512ccd52df, Jan 17 2014, 23:00:18)

numpy compatability test results, generated automatically by running
pypy/module/micronumpy/tool/numready <path-to-latest-pypy>

Overall: 510/558 names 141/161 ndarray attributes, 37/48 dtype attributes, 79/134 generic attributes, 28/32 flatiter attributes, 20/28 ufunc attributes

	PyPy		PyPy		PyPy		PyPy		PyPy
ALLOW_THREADS	✖	dtype.__hash__	✔	generic.argsort	✖	nansum	✔	poly	✔
BUFSIZE	✖	dtype.__init__	✔	generic.astype	✔	nbytes	✔	poly1d	✔
CLIP	✖	dtype.__le__	✖	generic.base	✖	ndarray	✔	polyadd	✔
ComplexWarning	✔	dtype.__len__	✔	generic.byteswap	✖	ndarray.T	✔	polyder	✔
DataSource	✔	dtype.__lt__	✖	generic.choose	✖	ndarray.__abs__	✔	polydiv	✔
ERR_CALL	✔	dtype.__mul__	✖	generic.clip	✖	ndarray.__add__	✔	polyfit	✔
ERR_DEFAULT	✖	dtype.__ne__	✔	generic.compress	✖	ndarray.__and__	✔	polyint	✔
ERR_DEFAULT2	✔	dtype.__new__	✔	generic.conj	✖	ndarray.__array__	✔	polymul	✔
ERR_IGNORE	✔	dtype.__reduce__	✔	generic.conjugate	✔	ndarray.__array_finalize__	✔	polynomial	✔
ERR_LOG	✔	dtype.__reduce_ex__	✔	generic.copy	✔	ndarray.__array_interface__	✔	polysub	✔
ERR_PRINT	✔	dtype.__repr__	✔	generic.cumprod	✖	ndarray.__array_prepare__	✔	polyval	✔
ERR_RAISE	✔	dtype.__rmul__	✖	generic.cumsum	✖	ndarray.__array_priority__	✖	power	✔

Numpy



NumPyPy Status: how much of numpy can you use in pypy?

Version: 2.7.3 (7951[illegible], Jan 17 2014, 23:00:18)

numpy compatability test results, generated automatically by running

Current Status

- Platforms
  - *Linux 32/64*
  - *Mac OS X 64*
  - *Windows 32*
  - *ARM/Linux*
- PyPy 2.2.1: Python 2.7.3
- Py3k 2.1 beta: Python 3.2.3
- Numpy
- STM

How to get PyPy

- Platform package manager debian/ubuntu, gentoo, Mac OS X homebrew, etc.
- Releases for supported platforms `http://pypy.org`
- Nightly builds `http://buildbot.pypy.org/nightly/trunk/`

Documentation

- Repository: `http://bitbucket.org/pypy/pypy`
- Docs: `http://doc.pypy.org`
- Blog: `http://morepypy.blogspot.de`
- Compatibility wiki: `http://bitbucket.org/pypy/compatibility`
- Bugs: `https://bugs.pypy.org`
- Buildbot: `http://buildbot.pypy.org`
- cffi Docs: `http://cffi.readthedocs.org`

Support

- irc: #pypy on freenode.net
- mailing lists:
  - *pypy: pypy-dev@python.org*
  - *cffi: python-cffi@googlegroups.com*

Demo

- Sobel filter written in Python (Loop-Aware Optimizations in PyPy's Tracing JIT, Ardö et. al. DLS 2012)