How PyPy makes your code run fast

Romain Guillebert

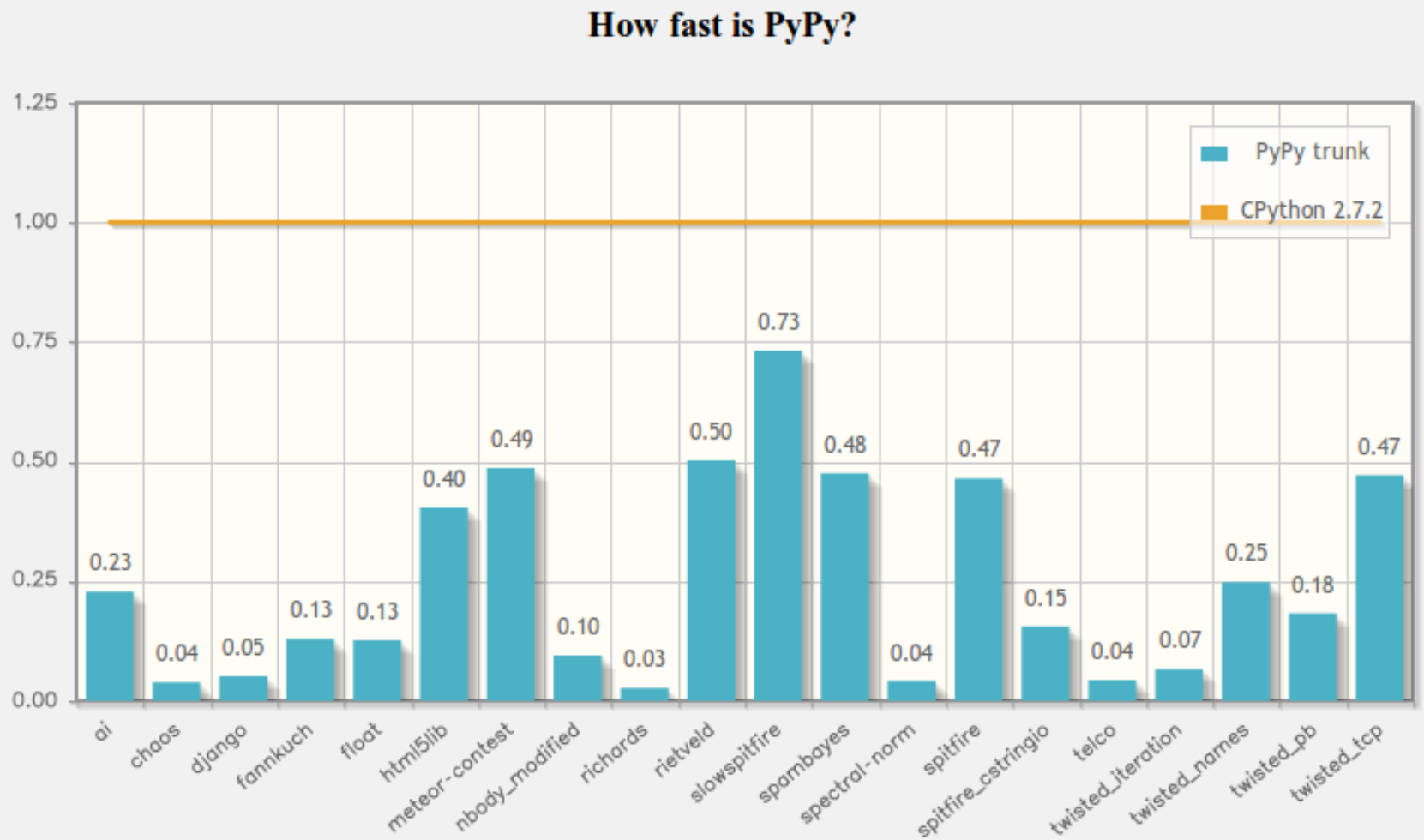
FOSDEM

February 2nd, 2014

Introduction

- Romain Guillebert, @rguillebert
- PyPy contributor for ~3 years
- NumPyPy contributor
- Please interrupt me
- How the PyPy JIT works (kind of)
- Warning : May contain traces of machine code

speed.pypy.org



Plot 1: The above plot represents PyPy trunk (with JIT) benchmark times normalized to CPython. Smaller is better.

It depends greatly on the type of task being performed. The geometric average of all benchmarks is 0.16 or **6.3** times *faster* than CPython

AOT

- Ahead of time compilation
- GCC
- Can optimize only on what it knows before running the program

Interpreter

- CPython, PyPy
- Executes an abstract representation of the program
- Not very smart

JIT

- PyPy
- Gathers information at runtime
- Produces optimized machine code

RPython

- Statically typed subset of Python
- The RPython compiler automatically generates the JIT from the annotated RPython code
- The JIT can be added with just one line of code
- More hints are needed to have an efficient JIT

Tracing JIT

- Optimizes loops
- Traces one iteration of a loop
- Produces a linear trace of execution
- Inlines almost everything
- The trace is then optimized and compiled

Guard

- The JIT produces a linear trace, but the code isn't
- The JIT can make assumptions that are not always true
- Guard : If this is true, continue, otherwise return to the interpreter
- guard_true, guard_class, guard_no_exception, ...

Bridge

- After a guard has failed X times, the other path is traced, compiled and attached to the trace

Optimizations

- Virtuals
- Virtualizables
- Promotion

Jitviewer

- Jitviewer demo

Demo

- Edge detection algorithm

Questions

- Questions ?