How PyPy runs your programs

Maciej Fijałkowski

PyCon ZA 2015

Oct 2nd, 2015

About me

- PyPy core developer for 8 years
- running consulting business `http://baroquesoftware.com`
- always interested in tooling and improving experience
- originally from Poland, living in Cape Town

This talk

The idea is to learn:

- how pypy runs your programs
- how to assess the **performance** of your program
- additionally, why a lot of **common folklore** is not true

The basics of PyPy

- python interpreter, just that
- uses magic to run code faster (most of the time)
- completely different codebase, not written in C
- ~**7x faster** than cpython

PyPy - the wider angle

- download it, should come in your distribution
- x86, x86_64, arm; windows, os x, linux
- open source (MIT)

PyPy - usage

- mostly long running server programs
- call C using cffi, a lot of libraries just work
- use virtualenv (you should anyway)

PyPy - magic

- just in time compiler, replaces bytecode to assembler under your feet
- takes a while to warm up, which defeats most short running programs
- most of the time faster, sometimes slower
- heavily optimizing, tons of heuristics for **typical** python programs

PyPy - smallish example

- take python code
- run python in interpreted mode (slow)
- run python in meta-interpreter mode (VERY SLOW)
- compile to optimized assembler
- repeat if necessary

Tracing JIT

- follow what the program is doing
- enter special mode where all the operations are recorded
- compile the recorded list of operations
- add a bunch of “guards” that check that we’re following the correct path and correct optimizations
- if guard fails, jump to interpreter
- if guard fails enough jump to metainterpreter
- repeat until all the paths are compiled to assembler

Performance

- you need a metric (response time, number of requests)
- the less you're trying to measure, the better
- benchmarks are a vast improvement
- repeatability is the key

Optimization for dummies

- Obligatory citation
  - *premature optimization is the root of all evil* (D. Knuth)
- Pareto principle, or 80-20 rule
  - 80% of the time will be spent in 20% of the program
  - 20% of 1 mln is 200 000
- Two golden rules:
  1. Identify the slow spots
  2. Optimize them

Guidos points about optimizing python

Why we're here?

- because the points above don't really work
- tradeoffs between productivity and performance
- once you fix obvious mistakes, profiles tend to look flat
- let's look at some of them

the basics

- have a metric
- a number, the shorter iteration the better
- use science!

a word about timeit

- don’t use it
- really, never
- no, not even that time
- minimum is a terrible thing
- disables the GC

introducing vmprof

- low-overhead profiler
- statistical
- visualization tools (work in progress)

vmprofdemo

- optimize for yourself
- cpython, pypy

more complicated example

- django admin
- 30 req/s
- in 2015, that's 90mln operations to show you a simple website

baroquesoftware.com

- monetizing open source is a difficult question, warrants another talk
- small consultancy on execution of programs
- talk to me

Questions?

- http://pypy.org
- http://baroquesoftware.com
- http://vmprof.com