The speed of PyPy

Maciej Fijałkowski

merlinux GmbH

RuPy, November 7th 2009, Poznań

Story about Python's speed

Speed of Python

- Python is slow

Speed of Python

- Python is slow

Speed of Python

- Is Python really slow?

Speed of Python

- Is Python really slow?
- Sometimes, for some usecases

Speed of Python

- Is Python really slow?
- Sometimes, for some usecases
- Let’s have a look at some examples

Nomenclature

- Python - a programming language
- CPython - main implementation of Python
- JVM - Java Virtual Machine - VM used to run Java, among others
- JIT - Just in time compiler
- Psyco - JIT for Python

Example run

- Float example, stolen from factor blog

	CPython	Java (hotspot client mode)
Average of 10 runs:	7.6s	0.77s

Example run

- Float example, stolen from factor blog

	CPython	Java (hotspot client mode)
Average of 10 runs:	7.6s	0.77s

- Python is 10x slower than Java

Example run

- Float example, stolen from factor blog

	CPython	Java (hotspot client mode)
Average of 10 runs:	7.6s	0.77s

- Python is 10x slower than Java
- Python is 10x slower than Java on this particular benchmark

Example run

- Float example, stolen from factor blog

	CPython	Java (hotspot client mode)
Average of 10 runs:	7.6s	0.77s

- Python is 10x slower than Java
- Python is 10x slower than Java on this particular benchmark
- CPython is 10x slower than Java on this particular benchmark

More about this example

	CPython	JVM	Psyco	PyPy
Average of 10 runs	7.6s	0.77s	4.4s	1.3s

More about this example

	CPython	JVM	Psyco	PyPy
Average of 10 runs	7.6s	0.77s	4.4s	1.3s

- So, it's CPython that is slow on this particular benchmark

More about this example

	CPython	JVM	Psyco	PyPy
Average of 10 runs	7.6s	0.77s	4.4s	1.3s

- So, it's CPython that is slow on this particular benchmark
- Same example, using numpy and vectorization about 3x faster than JVM

Python’s speed

- Instead of: “Why is Python slow?”

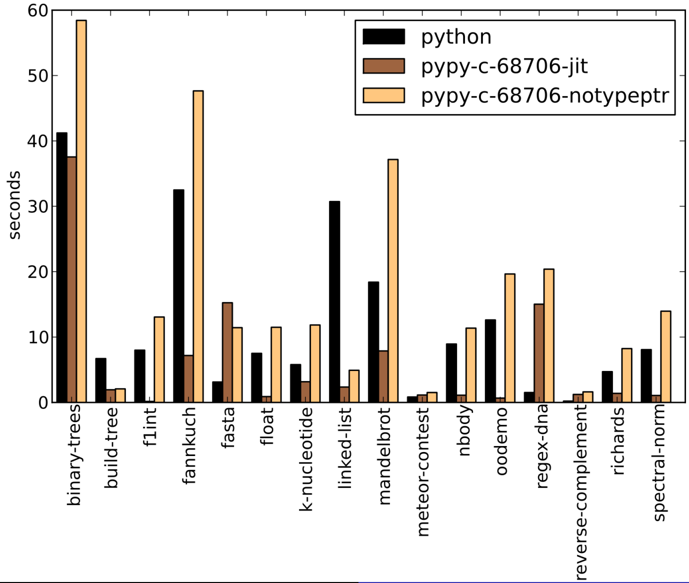
Python's speed

- Instead of: “Why is Python slow?”
- Better: “Why is Python hard to optimize?”

Python’s speed

- Instead of: “Why is Python slow?”
- Better: “Why is Python hard to optimize?”
- Even better: “How are we going to fix it?”

Some evidence



Why is Python hard to optimize?

- Duck typing (dynamic dispatch)
- Frames
- Object encapsulation
- Dictionaries of instances
- Changing globals
- Ability to dynamically change builtins

Duck typing

- Dispatching over item type
- `z = x + y`
- Needs to check what the type of `x` and `y` is

Frames

- Python interpreters use frames on heap (instead of stack)
- Locals are stored on those frames
- Intermediate results are store on valuestack of frames
- In fact, you can access frames via `sys._getframe()` `or` `traceback`

Object encapsulation

- Also called boxing
- Each object, even `int`, has to be boxed
- Requires allocations and indirection

Example - addition

- `z = x + y`
- read value for `x` from frame, store on valuestack
- read value for `x` from frame, store on valuestack
- allocate new integer
- read two values from valuestack, add and store the result in freshly allocated integer
- move the result from valuestack to locals

Example - addition

- `z = x + y`
- read value for `x` from frame, store on valuestack
- read value for `x` from frame, store on valuestack
- allocate new integer
- read two values from valuestack, add and store the result in freshly allocated integer
- move the result from valuestack to locals
- in fact, should be one assembler instruction

Dictionaries of instances

- Need to perform a dictionary lookup for `x.y`
- There are **three** lookups per method call (descriptor, object, type)
- **Two** for attribute access (descriptor, object)
- Looks like list lookup should be enough

Changing globals

- Global symbols can change at any moment in time
- Makes for example inlining hard
- Requires global dict lookups, even for constants

Ability to dynamically change builtins

- You can say `int = my_function`
- But you can't `int.__add__ = my_method`
- Still messes up optimizations
- Global lookup is also a dictionary lookup, even if globals don't change

Interpreting vs compiling

- Apparently, processors are good at branch prediction
- We didn't measure much of a difference, less than 2x overall

CPython specific problems

- In general, CPython is fairly well optimized
- refcounting is an inefficient garbage collection scheme
- GIL

Dynamic compilation to the rescue

- You don’t pay for feature, until you actually use it
- In static compilation, compiler has to prove that bad things can’t happen
- Impossible in Python
- With dynamic compilation, you just throw away compiled code in case things go wrong or start over

Dealing with frames

- Allocate frame

Dealing with frames

- Allocate frame
- Use C stack, but remember where frame fields are living on the stack

Dealing with frames

- Allocate frame
- Use C stack, but remember where frame fields are living on the stack
- Be able to reconstruct frame on demand (for example `sys._getframe() was called`)

Dealing with frames

- Allocate frame
- Use C stack, but remember where frame fields are living on the stack
- Be able to reconstruct frame on demand (for example `sys._getframe() was called`)
- The effect is that you don't pay for frames, unless you really use them

Dealing with dynamic dispatch

- The answer is to simply specialize over types
- Provides possibly multiple versions of compiled code for single Python code

Dealing with object encapsulation

- “Virtual objects”
- Also known as escape analysis
- If object does not “escape”, don’t allocate it at all

How does it work in practice?

How does it work in practice?

- Pretty well

How does it work in practice?

- Pretty well
- XXXX speedup over CPython

Dealing with attribute access

- Fairly complex task
- Sharing dict, more or less the same effect as V8's hidden classes

Dealing with attribute access

- Fairly complex task
- Sharing dict, more or less the same effect as V8's hidden classes
- Python is a very complex language
- Shadowing methods with attributes
- Descriptors before attributes

Other important optimizations

- Caching globals
- Caching builtins
- A lot of smaller ones

CPython vs PyPy

- I use CPython for everyday usage

CPython vs PyPy

- I use CPython for everyday usage
- But personally, I hope to change it in next months

CPython vs PyPy

- I use CPython for everyday usage
- But personally, I hope to change it in next months
- ... in places where performance matters, but that don't depend on third party C modules (like numpy)

CPython vs PyPy

- I use CPython for everyday usage
- But personally, I hope to change it in next months
- ... in places where performance matters, but that don't depend on third party C modules (like numpy)
- ... like building and developing PyPy

Status of PyPy

- Very compliant Python interpreter
- Most of important stdlib modules
- Differencies are agreed to be implementation details

Status of PyPy

- Very compliant Python interpreter
- Most of important stdlib modules
- Differencies are agreed to be implementation details
- or bugs

Examples of working programs

- Django (sqlite only)
- Twisted
- PyPy's translation toolchain
- ctypes

Status of JIT

- Because the way it’s constructed, handles all Python language features (unlike for example **Psyco**)
- Changes very quickly these days
- Ready for cautious tests
- Not ready as a drop-in replacement of CPython

Status of JIT

- Because the way it's constructed, handles all Python language features (unlike for example **Psyco**)
- Changes very quickly these days
- Ready for cautious tests
- Not ready as a drop-in replacement of CPython
- yet!

Adapt today!

- A fact: People rely on deep obscure features of language
- Examples:

Adapt today!

- A fact: People rely on deep obscure features of language
- Examples:

  -
    ```
    except ImportError, e:
      if str(e) != ...:  raise
    ```

Adapt today!

- A fact: People rely on deep obscure features of language
- Examples:

  -
    ```
    except ImportError, e:
      if str(e) != ...:  raise
    ```
  - Exact naming of list comprehension variable

Adapt today!

- A fact: People rely on deep obscure features of language
- Examples:

  -
    ```
    except ImportError, e:
      if str(e) != ...:  raise
    ```
  - Exact naming of list comprehension variable
  - Immediate finalization

Profit tomorrow!

- We plan to release JIT-ready version somewhere early 2010 xxx

How you can help

- It's all open source after all ...
- Try running existing programs
- Profile, report bugs

How you can help

- It’s all open source after all ...
- Try running existing programs
- Profile, report bugs
- Talk to your boss (XXX)

Thank you!

- This talk is already online (with all examples): `http://codespeak.net/svn/pypy/dist/extradoc/talk`
- `http://morepypy.blogspot.com`
- #pypy on freenode
- If you want to know more about PyPy, feel free to bug me around (like, how does the JIT work?)
- Any questions?