The speed of PyPy

Maciej Fijałkowski

merlinux GmbH

Pycon 2010, February 20th 2010, Atlanta

JIT - what's that about?

- **JIT is not a magical device!**

JIT - what's that about?

- **JIT is not a magical device!**
- removes bytecode overhead
- removes frame overhead
- can make runtime decisions
- more classic optimization that can follow

How well it works in practice?

The main idea

- python has advanced features (frame introspection, arbitrary code execution, overloading globals)
- with JIT, you don't pay for them if you don't use them
- however, you pay if you use them, but they work

A piece of advice

- don't use advanced features if you don't have to

Tracing JIT

- compiler traces the actual execution of Python program
- then compiles linear path to assembler
- example
- mostly for speeding up loops and to certain extent recursion

Removing frame overhead

```
x = y + z
```

- above has 5 frame accesses
- they can all be removed (faster!)

Removing frame overhead

```
x = y + z
```

- above has 5 frame accesses
- they can all be removed (faster!)
- they prevent optimizations from happening

Removing object boxing

```
i = 0
while i < 100:
   i += 1
```

- for each iteration we do a comparison and addition
- xxx integers on valuestack and xxx integers in locals
- all of those can be removed

Access costs

- local access costs nothing
- global access is cheap, if you don't change global `__dict__` too much XXX rephrase

Frame escapes

- calling `sys._getframe()`, `sys.exc_info()`
- exception escaping

Shared dicts (aka hidden classes)

- instance `__dict__` lookup becomes a list lookup
- if you're evil, it'll bail back to dict lookup

Shared dicts (aka hidden classes)

- instance `__dict__` lookup becomes a list lookup
- if you're evil, it'll bail back to dict lookup
- only for newstyle classes!

Version tags

- dicts on types are version-controlled
- this means methods are usually in known places

Version tags

- dicts on types are version-controlled
- this means methods are usually in known places
- ... if you don't modify them too often
- counters on classes are bad

Call costs

- calls can be inlined
- simple arguments are by far the best
- avoid `*args` and `**kwds`

Allocation patterns

- PyPy uses a moving GC (like JVM, .NET, etc.)
- pretty efficient for usecases with a lot of short-living objects
- objects are smaller than on CPython
- certain behaviors are different than on CPython

Differencies

- no refcounting semantics
- `id(obj)` can be expensive as it's a complex operation on a moving GC
- a large list of new objects is a bad case behavior

General rules

- don’t try to outsmart your compiler
- simple is better than complex
- metaprogramming is your friend
- measurment is the only meaningful way to check

Problems

- long traces - tracing is slow
- megamorphic calls
- metaclasses
- class global state

Problems

- long traces - tracing is slow
- megamorphic calls
- metaclasses
- class global state
- years of optimizations against CPython

Future

- release end March
- to try it out

That's all!

- Q & A
- http://morepypy.blogspot.com
- http://pypy.org
- http://merlinux.eu