Diving into Byte-code optimization in Python

SciPy India,

IIT Bombay Dec 05th 2011

Chetan Giridhar and Vishal Kanaujia

Fundamentals of Bytecode

- Python source code compiled into Python byte code by the CPython interpreter
- “.pyc”?
  - Automatic compilation : importing a module
  - Explicit compilation : *py_compile.compile(“module.py”)* – generates ‘module.pyc’
- The module ‘compileall’{}

Attribution-NonCommercial CC BY-NC

Fundamentals | more

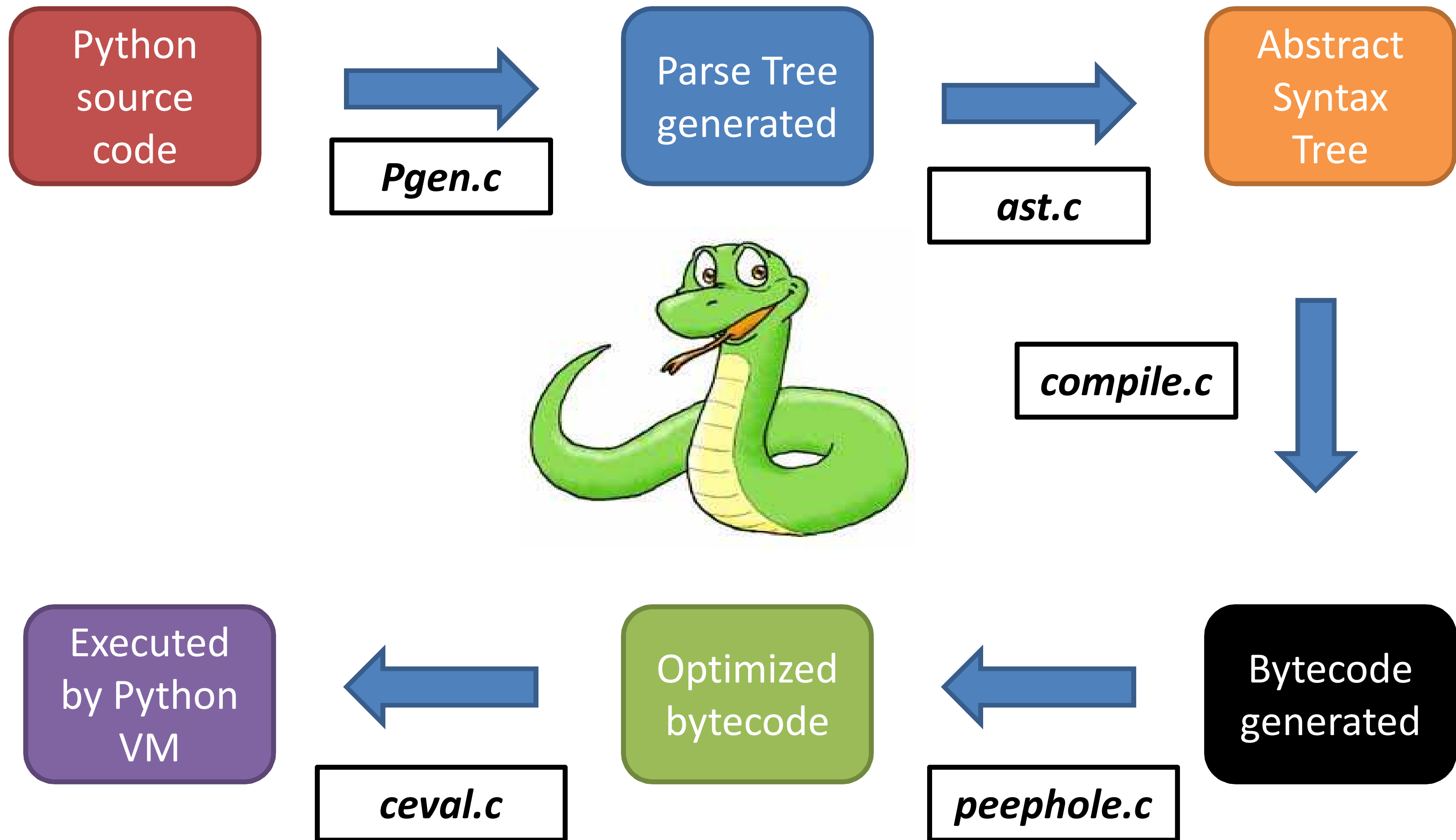
- A program doesn't run any faster when it is read from a ‘.pyc’ file. But, why?
- “.pyc” for a script executed on the command line
- “.pyc” files – good enough to distribute your code, but with a caveat!

Attribution-NonCommercial CC BY-NC

Compilation phases

- Uses a ‘lexer’ to prepare tokens of Python code
- A parser arranges token according to the language grammar and prepares concrete syntax tree
- Concrete syntax tree is transformed in to AST
- AST is compiled to produce Byte-codes

Attribution-NonCommercial CC BY-NC


Python source code
Pgen.c
Parse Tree generated
ast.c
Abstract Syntax Tree
compile.c
Bytecode generated
peephole.c
Optimized bytecode
ceval.c
Executed by Python VM

Python “ast” module

```
$ cat myast.py
import ast
nod = ast.parse('a +2')
print ast.walk(nod)
print ast.dump(nod)
```

```
$python myast.py
<generator object walk at 0xb784c8ec>
Module(body=[Expr(value=BinOp
(left=Name(id='a', ctx=Load()),
op=Add(),
 right=Num(n=2)))])
```

- Convenient for analysis, code transformations and generation
- ASTs are compiled to code objects

Attribution-NonCommercial CC BY-NC

A peek into Bytecodes

```
$ cat scipy.py
 1 import dis
 2
 3 def foo():
 4     i = 10
 5     print i
 6
 7 print dis.dis(foo)
```

```
$ python scipy.py
  4       0 LOAD_CONST          1 (10)
          3 STORE_FAST          0 (i)

  5       6 LOAD_FAST           0 (i)
          9 PRINT_ITEM
         10 PRINT_NEWLINE
         11 LOAD_CONST          0 (None)
         14 RETURN_VALUE
```

- Bytecode stream: An opcode mix
- Defined in “Python-2.7.2/Include/opcode.h”

Attribution-NonCommercial CC BY-NC

Python VM

- Engine to execute Bytecodes
- CPython VM is a C implementation
- Stack based process VM
  – PUSH/ POP operations
- Implementation of Python VM?

Attribution-NonCommercial CC BY-NC

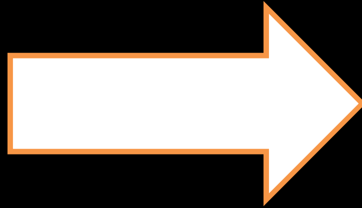
Python VM: Implementation

Python/ceval.c
--> PyEval_EvalFrameEx()

```
for(; ;)  {
      /* Extract opcode and argument */
            opcode = NEXTOP();
            if (HAS_ARG(opcode))
                oparg = NEXTARG();
                switch(opcode) {
                        case LOAD_CONST:
                                 ……
                 }
   }
```

Attribution-NonCommercial CC BY-NC

Optimizations



Tools

Attribution-NonCommercial CC BY-NC

Pyrex

- Python like language to create C module for Python
- Create your “pyx” files and compile them in “.c” files
- Import them as modules in your applications
- Pyrex used as:
  - speed up the execution of Python code
  - Python interface to existing C modules/libraries
- Lot of work for developer ☹
  - .py to .pyx?
  - thinking in C

Attribution-NonCommercial CC BY-NC

Psyco

- An intelligent option – JIT compilation
- Profiles dynamically an application for hot-spots
- “in-memory” prepares C extension and hook them appropriately
- Solves “duck” typing
- Memory footprint?
- Support till CPython 2.5

Attribution-NonCommercial CC BY-NC

Psyco| Intelligent use

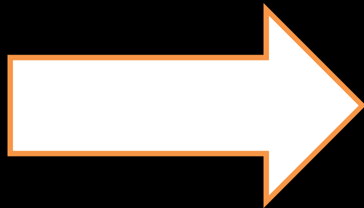
```
import psyco
from datetime import datetime

x = 5
def foo(number):
    iterations = number
    for i in range(iterations):
        print x*i

startTime  = datetime.now()
foo(100000)
endTime = datetime.now()
diffTime = endTime - startTime
print diffTime
```

Iterations	Without Pysco(ms)	With Pysco(ms)
1000	125	151
100000	12900	12570

- from psyco.classes import *
- pysco.bind(func)

Attribution-NonCommercial CC BY-NC

Optimizations



Bytecode level

Attribution-NonCommercial CC BY-NC

Why optimize Bytecode?

- Python optimization are ineffective, sometime ☹
- Duck typing
  - Run-time types
  - Optimizer misses many opportunities
  - TCO

```
def foo():
        i = 0
        i  = i + 1
        print i
```

```
4          0 LOAD_CONST           1 (0)
           3 STORE_FAST           0 (i)

5          6 LOAD_FAST            0 (i)
           9 LOAD_CONST           2 (1)
          12 BINARY_ADD
          13 STORE_FAST           0 (i)

6         16 LOAD_FAST            0 (i)
          19 PRINT_ITEM
          20 PRINT_NEWLINE
          21 LOAD_CONST           0 (None)
```

Attribution-NonCommercial CC BY-NC

Optimizations: Tail Recursive Calls

```
$ cat runbytecode3.py
import dis

def foo():
    x = 10
    if x == 10:
        x = 0
        foo()

print dis.dis(foo)
```

```
$ python runbytecode3.py

 5       1 LOAD_CONST          10
         2 STORE_FAST          x

 6       4 LOAD_FAST           x
         5 LOAD_CONST          10
         6 COMPARE_OP          ==
         7 POP_JUMP_IF_FALSE   to 17

 7       9 LOAD_CONST          0
        10 STORE_FAST          x

 8      12 LOAD_GLOBAL         foo
        13 CALL_FUNCTION       0
        14 POP_TOP
        15 JUMP_FORWARD        to 17
    >>  17 LOAD_CONST          None
        18 RETURN_VALUE
None
```

Attribution-NonCommercial CC BY-NC

BytePlay: Do it yourself!

- Experiment with Bytecodes
- Generates, recompiles and runs code on-the-fly
- Very useful to evaluate different code optimizations
- Example

Attribution-NonCommercial CC BY-NC

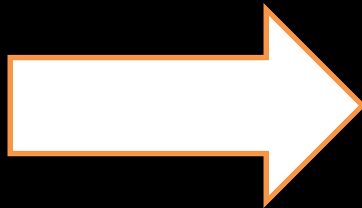
Playing with Bytecode

```
$cat runbytecode2.py

from byteplay import *
from pprint import pprint

def foo():
    x = 10
    print x

c = Code.from_code(foo.func_code)
print printcodelist(c.code)

# Modify the byte code
c.code[0] = (LOAD_CONST, 1000)

foo.func_code = c.to_code()

print printcodelist(c.code)

# Call the modified function
foo()
```

```
$ python runbytecode2.py
reassembled 'byteplay.py' imported.

 5       1 LOAD_CONST        10  <<<-------------------------
         2 STORE_FAST        x

 6       4 LOAD_FAST         x
         5 PRINT_ITEM
         6 PRINT_NEWLINE
         7 LOAD_CONST        None
         8 RETURN_VALUE
None

 5       1 LOAD_CONST        1000  <<<---------------------------
         2 STORE_FAST        x

 6       4 LOAD_FAST         x
         5 PRINT_ITEM
         6 PRINT_NEWLINE
         7 LOAD_CONST        None
         8 RETURN_VALUE
None
1000
```

Attribution-NonCommercial CC BY-NC

Going forward



PyPy

Attribution-NonCommercial CC BY-NC

Does PyPy help?

- Python interpreter in Python itself
- A target function needs to be written with subset of Python (Restricted Python)
- PyPy can translate this function to runtime of your choice ☺
  - Options: C/POSIX, CLI/.NET and Java/JVM
- Seamless and transparent

Attribution-NonCommercial CC BY-NC

PyPy | more

```
from pypy.translator.interactive import
    Translation

class compdec:
  def __init__(self, func):
    self.func = func
    self.argtypes = None

  def __call__(self, *args):
    argtypes = tuple(type(arg) for arg in
    args)
    if argtypes != self.argtypes:
      self.argtypes = argtypes
      t = Translation(self.func)
      t.annotate(argtypes)
      self.cfunc = t.compile_c()

    return self.cfunc(*args)
```

```
@compdec
def fact(n):
  if 0 == n:
    return 1
  return n * fact(n -1)
fact(10)
```

What PyPy does?

- The translation of the RPython function to C.
- Invoking the C compiler to create a C extension module.
- Importing the compiled function back into the Python interpreter.

Attribution-NonCommercial CC BY-NC

Recommendations

- Always profile your applications
- 90/10 rule of performance gain
- Performance critical portion could be written as C extensions
  - Wrappers like SWIG could be used to bridge Python/C applications
  - Pyrex, Psyco, and PyPy
  - Tuning Bytecodes manually
- Evaluate and use ☺

Attribution-NonCommercial CC BY-NC

References

- http://docs.python.org/release/2.5.2/lib/module-dis.html
- http://www.cosc.canterbury.ac.nz/greg.ewing/python/Pyrex/
- http://www.dalkescientific.com/writings/diary/archive/2010/02/22/instrumenting_the_ast.html
- http://cs.utsa.edu/~danlo/teaching/cs4713/lecture/node7.html
- http://www.devshed.com/c/a/Python/How-Python-Runs-Programs/4/
- http://www.enthought.com/~ischnell/paper.html
- http://codespeak.net/pypy/dist/pypy/doc/translation.html
- http://effbot.org/pyref/type-code.htm

Attribution-NonCommercial CC BY-NC

Questions

Thank you for your time and attention ☺

- Please share your feedback/ comments/ suggestions to us at:
- cjgiridhar@gmail.com , http://technobeans.com
- vishalkanaujia@gmail.com, http://freethreads.wordpress.com

Attribution-NonCommercial CC BY-NC

Backup slides

Attribution-NonCommercial CC BY-NC

Bytecodes | more

- Atomic unit of execution (thread safe)
- A Python module is compiled and saved in form of a “pyc” file
- An optimization; avoid compilation phase for future uses of a module
- If source file is changed, pyc is recompiled
- “pyc” file is coded in bytecode!

Attribution-NonCommercial CC BY-NC

Code Objects

- Bytecode representation
- Immutable objects (No ref to mutable objects)
- Function object (Code objects that reference global variables)
- “co_code” = Bytecodes

Attribution-NonCommercial CC BY-NC

Benefits of using Bytecode

- Architecture neutral code
- Portability
- Simple optimizations

Attribution-NonCommercial CC BY-NC