Optimizations And Micro-Optimizations In CPython

Larry Hastings
Facebook

PyCon 2010
February 19, 2010

facebook.

Provisos

Python = CPython
    (for the next thirty minutes)
    Python = language
    CPython = implementation

~~IronPython~~, ~~Jython~~, ~~PyPy~~, ~~Pynie~~, ~~TinyPy~~, ~~Pippy~~

Few numbers / benchmarks

Hold your questions, **66** slides

An Unconventional Talk

Uninformative
Pure entertainment

The Moral Of Our Story

Don't micro-optimize your code.

Do write clean, clear, readable Python
Do write shorter Python
Do use the library

What is CPython?

An interpreter

What is an interpreter?

Reads and executes a program

Generally tokenizer → virtual machine

Tokenizer

Reads source program

Translates to form designed for virtual machine
"bytecode"

Virtual Machine

Reads and executes “bytecode”

How do you write a virtual machine?

Portable
ANSI C

The Big Eval Loop

```
for (;;) {
  switch (*token++) {
      case TOKEN_LITERAL:
        …
      case TOKEN_ADD:
        …
  }
}
```

Python's CEval Loop

In **Python/ceval.c**

Optimization 1
Remove SET_LINENO

Author Guido van Rossum
Revision 7516
Committed 1997/01/23
Shipped Python 1.5

Author Michael Hudson
Revision 28249
Committed 2002/08/15
Shipped Python 2.3

What was SET_LINENO

Opcode

Store line number in current frame object

`f_lineno`

What's it for?

`f_lineno` Tracebacks
SET_LINENO Tracing

CPython 1.5

`-O`

`co_lnotab`

SET_LINENO tracing, no `-O`

CPython 2.3

SET_LINENO gone!

Tracing uses `co_lnotab`

Optimization 2
Computed Gotos

Author Antoine Pitrou
Revision 68924
Committed 2009/01/25
Shipped Python 3.1 (2.7)

Relaxing a constraint

Portable
~~ANSI~~ C

A quick refresher on goto

```
  if (!something()) {
    goto FAIL;
  }

  return thing;

FAIL: // "FAIL" is a "label"
  return NULL;
```

Two GCC extensions to C

Labels as Values
Computed Gotos

Labels as Values

```
void *foo = &&FAIL;
// foo now points to FAIL
```

Computed Gotos

```
void *foo = &&FAIL;
// foo now points to FAIL

goto *foo;
// jump to FAIL
```

--with-computed-gotos=no

```
case OPCODE_SOMETHING:
  …
  continue;
```

--with-computed-gotos=yes

```
OPCODE_SOMETHING:
  …
  goto *opcode_targets[*next_instr++];
```

+17%

Better pipeline branch prediction

`switch` = one indirect jump
Computed gotos = many indirect jumps

Optimization 3
Unladen Swallow

Authors Collin Winter and Jeffrey Yasskin
Revision n/a
Committed n/a
Shipped n/a

Unladen Swallow

~~Portable~~ Cross-Platform

A JIT for CPython
Based on LLVM

“is a go!”

Film At 11

Saturday, 11:05am Centennial I
"Unladen Swallow: fewer coconuts, faster Python"
Collin Winter
30 minutes

Sunday, 9:20am All rooms
Keynote: "State of Unladen Swallow"
Collin Winter
15 minutes

PEP 3146 "Merging Unladen Swallow Into CPython"

Optimization 4
timsort

Author Tim Peters
Revision 27847
Committed 2002/07/31
Shipped Python 2.3

Existing sort

Not stable

Equal elements A , B may be reversed

New research in sorting

Research pays off!

"non-recursive adaptive stable natural mergesort / binary insertion sort hybrid"

timsort

“adaptive”

Capitalize on existing sorted-ness

Objects/listsort.txt

30k document

Fabulous!

often > 10x faster for mostly-sorted data

"supernatural"!

Java 7

dict

The Importance Of Dicts

Global variable **1**
Built-in function **2**
Object attribute **1+**
Object method **2+** (usually)
Dict **1**
Dict-**like 1+**

dictobject basics

Hash table
Closed hashing
Resize when 2/3 full

Write a “dummy” when removing

Optimization 5
fastlocals

Author Guido van Rossum
Revision 3428
Committed 1993/03/30
Shipped Python 0.9.8

locals

Created when variable is assigned to

Assignment inside a function is **always** a local
(unless it's a `global`)

Function arguments are locals

Python name resolution,1993

Three dicts

- locals?
- globals?
- built-ins?
- raise NameError

Further Importance Of Dicts

Locals stored in a dict

Local variables **1**

Therefore:

Global variables **2**

Built-in functions **3**

fastlocals

"Change to speed up local variables enormously"

Tokenizer assigns each local a slots in a list

`f_fastlocals` list in frame object at runtime

Default is NULL

Example of fastlocals

```
def fn(self, a):
  b = a
  return b + a
```

```
f_fastlocals array
  [0] = self
  [1] = a
  [2] = b
```

Changed observable behavior!

```
foo = 3

def bar():
   x = foo
   foo = "howdy"

bar()
```

Micro-Optimization 1
Statistical dict comparison reordering

Author Tim Peters
Revision 20777
Committed 2001/05/12
Shipped Python 2.2

May 2001: month of the dict

Probing code rewrite

replace “polynomial machinery” with simpler math

Add `ma_smalltable` to dictobject

Saves an alloc for dicts < 5 objects

Micro-optimizations

Analyzing the hash entry

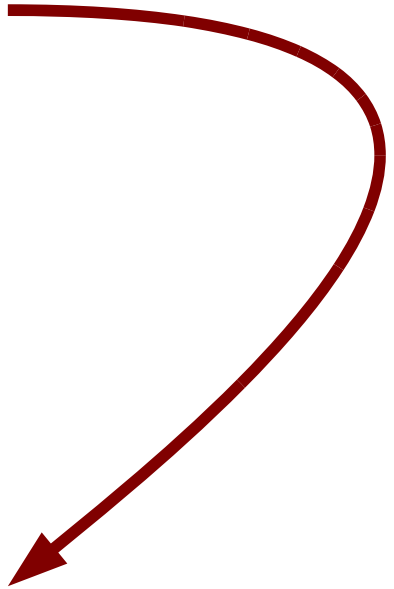
```
if (entry.key == NULL)
  // not found, return
else if (entry.key == dummy)
  // loop
else if (entry.key == key)
  // found, return
else if (entry.hash == hash)
  // rich comparison
probe
```

r20777

dummy is 200-300x less likely than key or NULL

Reordered comparisons

```
if (entry.key == NULL)
  // not found, return
else if (entry.key == dummy)
  // loop
else if (entry.key == key)
  // found, return
else if (entry.hash == hash)
  // rich comparison...
else if (entry.key == dummy)
  // loop
probe
```



Micro-Optimization 2
Aggressive dict resize

Author Raymond Hettinger
Revision 32637
Committed 2003/05/05
Shipped Python 2.3
Issue 729395

May 2003

A solid month of research into **dict**

Goal: **strictly** faster, **never** slower

Results

Objects/dictnotes.txt

And... a proposed **1** line patch

The patch

When growing a dict

```
- size * 2
+ size *(size>50000 ? 2 : 4)
```

0% - 5%

Sparser hash tables
Fewer collisions
More cache-friendly

Increased memory usage

This shows you the TLC lavished on CPython.

Micro-Optimization 3
str.repeat

Author Raymond Hettinger
Revision 30616
Committed 2003/01/06
Shipped Python 2.3

Refresher on **repeat**

"*" when used with a sequence and an integer

```
>>> "ab" * 5
"abababababab"

>>> array.array("b", [0]) * 3
array.array("b", [0,0,0])
```

Python-Dev 2003/01/06

Christian Tismer

- `"x" * 1000000` takes **63ms**
- `"x" * 1000 * 1000` takes **6.7ms**

str_repeat

```
str_repeat(str s, int n)
  new_size = length(s) * n
  tmp = new_s = alloc(new_size)
  for (i = 0; i < n; i++)
    memcpy(tmp, s, length(s))
    tmp += length(s)
  return new_s
```

Tismer's Analysis

`" " * 1000000`

1 alloc, 1000000 loops, 1000000 **memcpy()** calls

`" " * 1000 * 1000`

2 allocs, 2000 loops, 2000 **memcpy()** calls

998000 fewer loops and **memcpy()**s = faster!

Get Hettinger!

```
str_repeat(str s, int n)
  new_size = length(s) * n
  new_s = alloc(new_size)
  memcpy(new_s, s, length(s))
  tmp = new_s + length(s)
  end = new_s + new_size
  while (tmp < end)
    len = MAX(tmp - new_s, end - tmp)
    memcpy(tmp, s, len)
    tmp += len
  return new_s
```

Hettinger's version

N loops becomes $\log_2(N)$ loops

`"abc" * 1000000`

1 alloc, 20 loops, 21 **`memcpy()`** calls

`"abc" * 1000 * 1000`

2 allocs, 20 loops, 22 **`memcpy()`** calls

Hettingerization Complete

```
if s.len == 1
  memset(new_s, *s, new_size)
```

The right way is now the **fastest** way.

Don't micro-optimize your code.

For further reading

"What's New In Python"

PEPS

CPython source

- **Objects/*.txt**
- **Misc/***
  - Long comments
  - **blame** interesting lines to get revision
  - Cross-reference with bugs and Python-Dev

Questions?

Answers!
Hand-waving!
Embarassment!

The End

Thanks for coming!

larry@facebook.com

What do -O and -OO do?

-O

- Use `.pyo` (normally `.pyc`)
- Set `__debug__` to 0 (normally 1)
- Throw away asserts

-OO

- Throw away docstrings