

Pythran – C++ for Snakes

Static Compiler for High Performance

Serge Guelton & Pierrick Brunet & Mehdi Amini

PyData - May 4th 2014



Get the slides: <http://goo.gl/6dgra0>

Disclaimer

All timings were performed using OSX 10.9, an i7-3740QM CPU @ 2.70GHz, Python 2.7.6, current Pythran (c7de8b9c), pypy 2.2.1, numba 0.13.1, gcc 4.8, Clang r207887.

I am **not** Pythonista, but I'm interested in performance in general. Daily job: driver-level C code, assembly, multi-threaded C++, GPU, ...

Disclaimer

All timings were performed using OSX 10.9, an i7-3740QM CPU @ 2.70GHz, Python 2.7.6, current Pythran (c7de8b9c), pypy 2.2.1, numba 0.13.1, gcc 4.8, Clang r207887.

I am **not** Pythonista, but I'm interested in performance in general. Daily job: driver-level C code, assembly, multi-threaded C++, GPU, ...

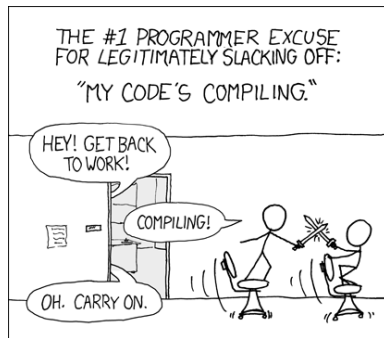
Note: I love to troll, so let's have a beer later and talk about how awful Python is! ;-)

Disclaimer

All timings were performed using OSX 10.9, an i7-3740QM CPU @ 2.70GHz, Python 2.7.6, current Pythran (c7de8b9c), pypy 2.2.1, numba 0.13.1, gcc 4.8, Clang r207887.

I am **not** Pythonista, but I'm interested in performance in general. Daily job: driver-level C code, assembly, multi-threaded C++, GPU, ...

Note: I love to troll, so let's have a beer later and talk about how awful Python is! ;-)

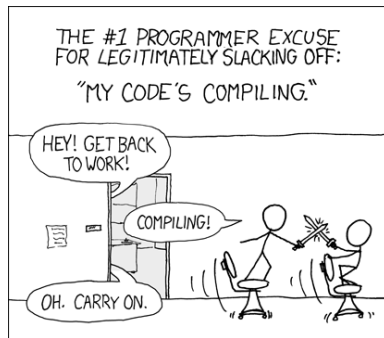


Disclaimer

All timings were performed using OSX 10.9, an i7-3740QM CPU @ 2.70GHz, Python 2.7.6, current Pythran (c7de8b9c), pypy 2.2.1, numba 0.13.1, gcc 4.8, Clang r207887.

I am **not** Pythonista, but I'm interested in performance in general. Daily job: driver-level C code, assembly, multi-threaded C++, GPU, ...

Note: I love to troll, so let's have a beer later and talk about how awful Python is! ;-)



By the way this talk is written in Latex and takes more than 10 seconds to **compile**.

Prototyping Tools for Scientific Computing



Prototyping Tools for Scientific Computing



Tools for Scientific Computing in Python



theano



FORTRAN + f2py

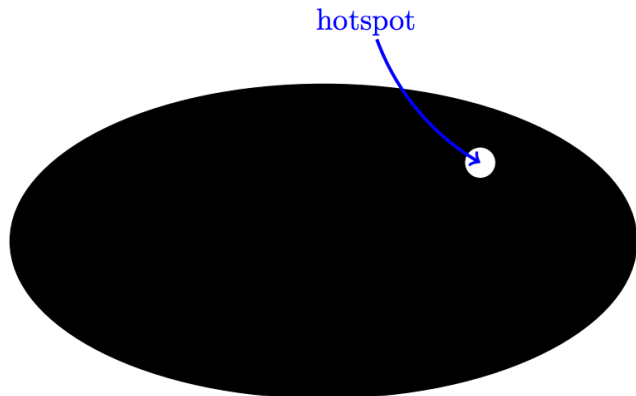
NUMBA



рyрy

C + SWIG

I Do not Know Much About Python But it Does not Matter Because...



I only care about the white spot.

Regular IPython Session

```
>>> import numpy as np
>>> def rosen(x):
...     return sum(100.*(x[1:]-x[:-1]**2.))**2.
...             + (1-x[:-1])**2.)
>>> import numpy as np
>>> r = np.random.rand(100000)
>>> %timeit rosen(r)
10 loops, best of 3: 35.1 ms per loop
```

In mathematical optimization, the Rosenbrock function is a non-convex function used as a performance test problem for optimization algorithms introduced by Howard H. Rosenbrock in 1960.[1] It is also known as Rosenbrock's valley or Rosenbrock's banana function. (Wikipedia)

IPython Session with Pythran

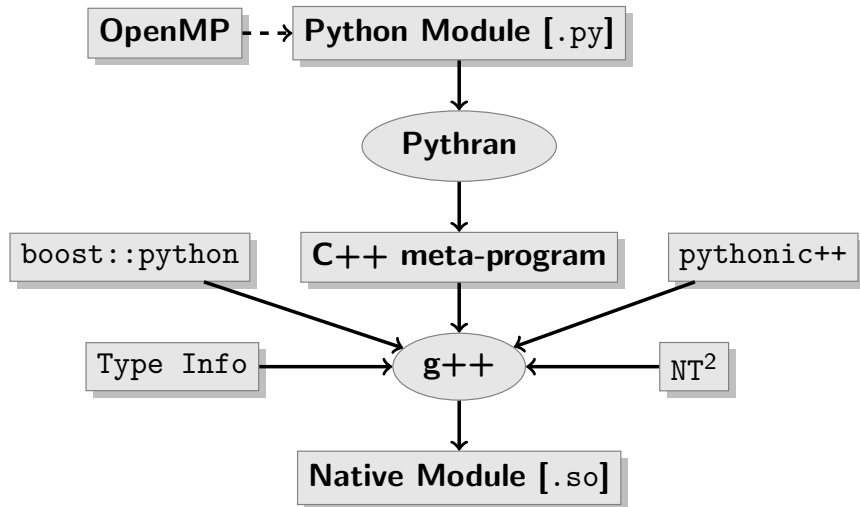
```
>>> %load_ext pythranmagic
>>> %%pythran
import numpy as np
#pythran export rosen(float[])
def rosen(x):
    return sum(100.*(x[1:] - x[:-1])**2.)**2.
    + (1-x[:-1])**2.)
>>> import numpy as np
>>> r = np.random.rand(100000)
>>> %timeit rosen(r)
10000 loops, best of 3: 121 us per loop
```

IPython Session with Pythran

```
>>> %load_ext pythranmagic
>>> %%pythran
import numpy as np
#pythran export rosen(float[])
def rosen(x):
    return sum(100.*(x[1:] - x[:-1])**2.)**2.
    + (1-x[:-1])**2.)
>>> import numpy as np
>>> r = np.random.rand(100000)
>>> %timeit rosen(r)
10000 loops, best of 3: 121 us per loop
```

That's a $\times 290$ speedup!

Pythran's Meta Program Generator



A Story of Python & C++

```
def dot(l0, l1):
    return sum(x*y for x,y in zip(l0,l1))
```

21

```
template<class T0, class T1>  
auto dot(T0&& l0, T1&& l1)  
-> decltype(/* ... */) {  
    return pythonic::sum(  
        pythonic::map(  
            operator_::multiply(),  
            pythonic::zip(std::forward<T0>(l0),  
                          std::forward<T1>(l1))  
        ) );  
}
```


Let's Dive Into the Backend Runtime...

```
template <class Op, class S0, class... Iters>
    auto map_aux (Op &op, S0 const &seq, Iters... iters)
        -> sequence <decltype(op(*seq.begin(), *iters...))>
    {
        decltype(_map(op, seq, iters...)) s;
        auto iter = std::back_inserter(s);
        for(auto& iseq : seq)
            *iter += op(iseq , *iters++...);
        return s;
    }

template <class Op, class S0, class... SN>
    auto map ( Op op, S0 const &seq, SN const &... seqs )
        -> decltype(_map( op, seq, seqs.begin ()...))
    {
        return _map(op, seq, seqs.begin()...);
    }
}
```

Let's Dive Into the Backend Runtime... I'm Kidding!

```
template <class Op, class S0, class... Iters>
    auto map_aux (Op &op, S0 const &seq, Iters... iters)
        -> sequence <decltype(op(*seq.begin(), *iters...))>
    {
        decltype(_map(op, seq, iters...)) s;
        auto iter = std::back_inserter(s);
        for(auto& iseq : seq)
            *iter += op(iseq , *iters++...);
        return s;
    }

template <class Op, class S0, class... SN>
    auto map ( Op op, S0 const &seq, SN const &... seqs )
        -> decltype(_map( op, seq, seqs.begin ()...))
    {
        return _map(op, seq, seqs.begin()...);
    }
}
```



Static Compilation

Buys time for many time-consuming analyses

Points-to, used-def, variable scope, memory effects, function purity...

Unleashes powerful C++ optimizations

Lazy Evaluation, Map Parallelizations, Constant Folding

Requires static type inference

```
#pythran export foo(int list, float)
```

Only annotate exported functions!

Gather as many information as possible

(Typing is just one information among others)

Example of Analysis: Points-to

```
def foo(a,b):  
    c = a or b  
    return c*2
```

Where does c points to ?

Example of Analysis: Argument Effects

```
def fib(n):  
    return n if n<2 else fib(n-1) + fib(n-2)  
  
def bar(l):  
    return map(fib, l)  
  
def foo(l):  
    return map(fib, random.sample(l, 3))
```

Do fibo, bar and foo update their arguments?

Example of Analysis: Pure Functions

```
def f0(a):
    return a**2

def f1(a):
    b = f0(a)
    print b
    return b

l = list(...)
map(f0, l)
map(f1, l)
```

Are f0 and f1 pure functions?

Example of Analysis: Use - Def Chains

```
a = '1'
if cond:
    a = int(a)
else:
    a = 3
print a
a = 4
```

Which version of `a` is seen by the `print` statement?

Gather as many information as possible

Turn them into Code Optimizations!

Example of Code Optimization: False Polymorphism

```
a = cos(1)
if cond:
    a = str(a)
else:
    a = None
foo(a)
```

Is this code snippet statically typable?

Example of Code Optimization: Lazy Iterators

```
def valid_conversion(n):
    l = map(math.cos, range(n))
    return sum(l)

def invalid_conversion(n):
    l = map(math.cos, range(n))
    l[0] = 1
    return sum(l) + max(l)
```

Which `map` can be converted into an `imap`

Gather as many information as possible

Turn them into Code Optimizations

Vectorize! Parallelize!

Explicit Parallelization

```
def hyantes(xmin, ymin, xmax, ymax, range_x, range_y, t):
    pt = [[0]*range_y for _ in range(range_x)]
    "#omp_parallel_for"
    for i in xrange(range_x):
        for j in xrange(range_y):
            s = 0
            for k in t:
                tmp = 6368.* math.acos(math.cos(xmin+step*i)*math.cos( k[0] ) *
                                         math.cos((ymin+step*j)-k[1]) +
                                         math.sin(xmin+step*i)*math.sin(k[0]))

                if tmp < range_:
                    s+=k[2] / (1+tmp)
            pt[i][j] = s
    return pt
```

Tool	CPython	Pythran	OpenMP
Timing	639.0ms	44.8ms	11.2ms
Speedup	$\times 1$	$\times 14.2$	$\times 57.7$

Julia Set, a Cython Example

```
def run_julia(cr, ci, N, bound, lim, cutoff):
    julia = np.empty((N, N), np.uint32)
    grid_x = np.linspace(-bound, bound, N)
    t0 = time()
    "#omp_parallel_for"
    for i, x in enumerate(grid_x):
        for j, y in enumerate(grid_x):
            julia[i,j] = kernel(x, y, cr, ci, lim, cutoff)
    return julia, time() - t0
```

From Scipy2013 Cython Tutorial.

Tool	CPython	Cython	Pythran	+OpenMP
Timing	3630ms	4.3ms	3.71ms	1.52ms
Speedup	$\times 1$	$\times 837$	$\times 970$	$\times 2368$

Mandelbrot, a Numba example

```
@autojit
def mandel(x, y, max_iters):
    """
    Given the real and imaginary parts of a complex
    determine if it is a candidate for membership in
    set given a fixed number of iterations.
    """
    i = 0
    c = complex(x, y)
    z = 0.0j
    for i in range(max_iters):
        z = z**2 + c
        if abs(z)**2 >= 4:
            return i

    return 255
```

```
@autojit
def create_fractal(min_x, max_x, min_y, max_y, image, iters):
    height = image.shape[0]
    width = image.shape[1]

    pixel_size_x = (max_x - min_x) / width
    pixel_size_y = (max_y - min_y) / height

    for x in range(width):
        real = min_x + x * pixel_size_x
        for y in range(height):
            imag = min_y + y * pixel_size_y
            color = mandel(real, imag, iters)
            image[y, x] = color

    return image
```

Tool	CPython	Numba	Pythran
Timing	8170ms	56ms	47.2ms
Speedup	$\times 1$	$\times 145$	$\times 173$

"gg++ -Ofast" here

Nqueens, to Show Some Cool Python

```
# Pure-Python implementation of
# itertools.permutations()
# Why? Because we can :-)
```

```
def permutations(iterable, r=None):
    pool = tuple(iterable)
    n = len(pool)
    if r is None:
        r = n
    indices = range(n)
    cycles = range(n-r+1, n+1)[::-1]
    yield tuple(pool[i] for i in indices[:r])
    while n:
        for i in reversed(xrange(r)):
            cycles[i] -= 1
            if cycles[i] == 0:
                indices[i:] = indices[i+1:] + indices[i:i+1]
                cycles[i] = n - i
            else:
                j = cycles[i]
                indices[i], indices[-j] = \
                    indices[-j], indices[i]
            yield tuple(pool[i] for i in indices[:r])
            break
        else:
            return
```

```
#pythran export n_queens(int)
def n_queens(queen_count):
    """N-Queens solver.

    Args:
        queen_count: the number of queens to solve
        for. This is also the board size.

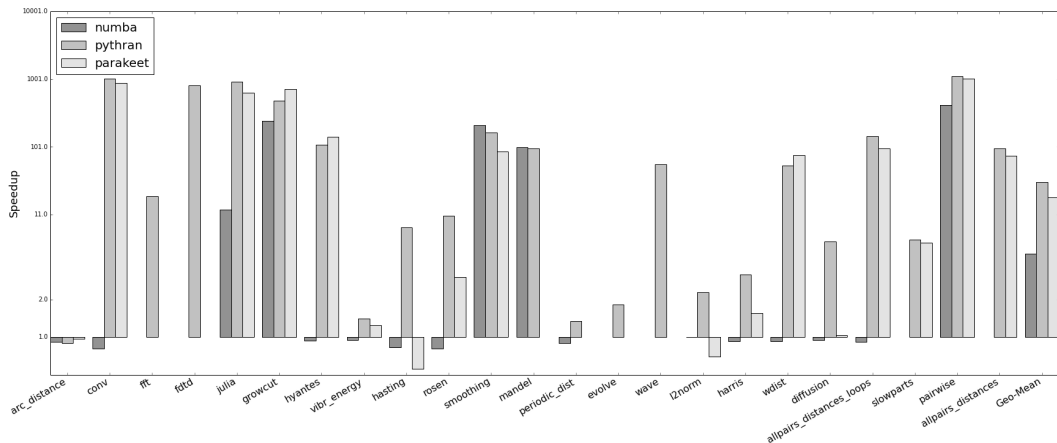
    Yields:
        Solutions to the problem. Each yielded
        value is looks like (3, 8, 2, ..., 6)
        where each number is the column position
        for the queen, and the index into the
        tuple indicates the row.
    """
    out = list()
    cols = range(queen_count)
    for vec in permutations(cols, None):
        if (queen_count == len(set(vec[i]+i
                                     for i in cols)))):
            out.append(vec)
    return out
```

Solving the NQueen problem, using generator, generator expression, list comprehension, sets. . .

<http://code.google.com/p/unladen-swallow/>

Tool	CPython	PyPy	Pythran
Timing	2640.6ms	501.1ms	693.3ms
Speedup	$\times 1$	$\times 5.27$	$\times 3.8$

Numpy Benchmarks



<https://github.com/serge-sans-paille/numpy-benchmarks/>

Made with Matplotlib last night!

Debian clang version 3.5-1 exp1, x86-64, i7-3520M CPU @ 2.90GHz. No OpenMP/Vectorization for Pythran.

The Compiler Is not the Solution to Keyboard-Chair Interface, Is It?

```
def total(arr):
    s = 0
    for j in range(len(arr)):
        s += arr[j]
    return s

def varsum(arr):
    vs = 0
    for j in range(len(arr)):
        mean = (total(arr) / len(arr))
        vs += (arr[j] - mean) ** 2
    return vs
```

```
#pythran export stddev(float64 list list)
def stddev(partitions):
    ddof=1
    final = 0.0
    for part in partitions:
        m = total(part) / len(part)
        # Find the mean of the entire group.
        gtotal = total([total(p) for p in partitions])
        glength = total([len(p) for p in partitions])
        g = gtotal / glength
        adj = ((2 * total(part) * (m - g)) + ((g ** 2 - m **
            final += varsum(part) + adj
    return math.sqrt(final / (glength - ddof))
```

Version	Awful	Less Awful	OK	<i>Differently OK</i>	<i>OK</i>	Numpy
CPython	127s	150ms	54.3ms	53.8ms	47.6ms	8.2ms
Pythran						

The Compiler Is not the Solution to Keyboard-Chair Interface, Is It?

```
def total(arr):
    s = 0
    for j in range(len(arr)):
        s += arr[j]
    return s

def varsum(arr):
    vs = 0
    for j in range(len(arr)):
        mean = (total(arr) / len(arr))
        vs += (arr[j] - mean) ** 2
    return vs
```

```
#pythran export stddev(float64 list list)
def stddev(partitions):
    ddof=1
    final = 0.0
    for part in partitions:
        m = total(part) / len(part)
        # Find the mean of the entire group.
        gtotal = total([total(p) for p in partitions])
        glength = total([len(p) for p in partitions])
        g = gtotal / glength
        adj = ((2 * total(part) * (m - g)) + ((g ** 2 - m **
            final += varsum(part) + adj
    return math.sqrt(final / (glength - ddof))
```

Version	Awful	Less Awful	OK	<i>Differently OK</i>	<i>OK</i>	Numpy
CPython	127s	150ms	54.3ms	53.8ms	47.6ms	8.2ms
Pythran	1.38s	4.7ms				

The Compiler Is not the Solution to Keyboard-Chair Interface, Is It?

```
def total(arr):
    s = 0
    for j in range(len(arr)):
        s += arr[j]
    return s

def varsum(arr):
    vs = 0
    for j in range(len(arr)):
        mean = (total(arr) / len(arr))
        vs += (arr[j] - mean) ** 2
    return vs
```

```
#pythran export stddev(float64 list list)
def stddev(partitions):
    ddof=1
    final = 0.0
    for part in partitions:
        m = total(part) / len(part)
        # Find the mean of the entire group.
        gtotal = total([total(p) for p in partitions])
        glength = total([len(p) for p in partitions])
        g = gtotal / glength
        adj = ((2 * total(part) * (m - g)) + ((g ** 2 - m **
            final += varsum(part) + adj
    return math.sqrt(final / (glength - ddof))
```

Version	Awful	Less Awful	OK	<i>Differently OK</i>	OK	Numpy
CPython	127s	150ms	54.3ms	53.8ms	47.6ms	8.2ms
Pythran	1.38s	4.7ms	4.8ms	5.8ms		

The Compiler Is not the Solution to Keyboard-Chair Interface, Is It?

```
def total(arr):
    s = 0
    for j in range(len(arr)):
        s += arr[j]
    return s

def varsum(arr):
    vs = 0
    for j in range(len(arr)):
        mean = (total(arr) / len(arr))
        vs += (arr[j] - mean) ** 2
    return vs
```

```
#pythran export stddev(float64 list list)
def stddev(partitions):
    ddof=1
    final = 0.0
    for part in partitions:
        m = total(part) / len(part)
        # Find the mean of the entire group.
        gtotal = total([total(p) for p in partitions])
        glength = total([len(p) for p in partitions])
        g = gtotal / glength
        adj = ((2 * total(part) * (m - g)) + ((g ** 2 - m **
            final += varsum(part) + adj
    return math.sqrt(final / (glength - ddof))
```

Version	Awful	Less Awful	OK	<i>Differently OK</i>	<i>OK</i>	Numpy
CPython	127s	150ms	54.3ms	53.8ms	47.6ms	8.2ms
Pythran	1.38s	4.7ms	4.8ms	5.8ms	4.7ms	1.7ms

Get It

- ▶ Follow the source: <https://github.com/serge-sans-paille/pythran>
(*we are waiting for your pull requests*)
- ▶ Debian repo: `deb http://ridee.enstb.org/debian unstable main`
- ▶ Join us: #pythran on Freenode (*very active*),
or pythran@freelists.org (*not very active*)

Available on PyPI, using Python 2.7, +2000 test cases, PEP8 approved, clang++ & g++ (>= 4.8) friendly, Linux and OSX validated.

Conclusions

<http://pythonhosted.org/pythran/>

Compiling Python means more than typing and translating

Conclusions

<http://pythonhosted.org/pythran/>

Compiling Python means more than typing and translating

What next:

Conclusions

<http://pythonhosted.org/pythran/>

Compiling Python means more than typing and translating

What next:

- ▶ Release the PyData version (oops we're late)

Conclusions

<http://pythonhosted.org/pythran/>

Compiling Python means more than typing and translating

What next:

- ▶ Release the PyData version (ooops we're late)
- ▶ User module import (pull request already issued)

Conclusions

<http://pythonhosted.org/pythran/>

Compiling Python means more than typing and translating

What next:

- ▶ Release the PyData version (oops we're late)
- ▶ User module import (pull request already issued)
- ▶ Set-up an daily performance regression test bot (ongoing work with codespeed)

Conclusions

<http://pythonhosted.org/pythran/>

Compiling Python means more than typing and translating

What next:

- ▶ Release the PyData version (oops we're late)
- ▶ User module import (pull request already issued)
- ▶ Set-up an daily performance regression test bot (ongoing work with codespeed)
- ▶ Re-enable vectorization through boost::simd and nt2 (soon)

Conclusions

<http://pythonhosted.org/pythran/>

Compiling Python means more than typing and translating

What next:

- ▶ Release the PyData version (oops we're late)
- ▶ User module import (pull request already issued)
- ▶ Set-up an daily performance regression test bot (ongoing work with codespeed)
- ▶ Re-enable vectorization through boost::simd and nt2 (soon)
- ▶ More Numpy support, start looking into Scipy

Conclusions

<http://pythonhosted.org/pythran/>

Compiling Python means more than typing and translating

What next:

- ▶ Release the PyData version (oops we're late)
- ▶ User module import (pull request already issued)
- ▶ Set-up an daily performance regression test bot (ongoing work with codespeed)
- ▶ Re-enable vectorization through boost::simd and nt2 (soon)
- ▶ More Numpy support, start looking into Scipy
- ▶ Polyhedral transformations (in another life...)