

Sage Days 24: Cython

William Stein

July 18, 2010

Further motivation: <http://shootout.alioth.debian.org/>

Program	Source Code	CPU secs	Elapsed secs	Memory KB	Code B	≈ CPU Load			
pidigits									
Python CPython		6.17	6.17	4,260	476	0%	1%	0%	100%
C GNU gcc		2.72	2.71	1,012	541	0%	4%	0%	100%
regex-dna									
Python CPython		25.48	25.48	195,188	342	0%	0%	0%	100%
C GNU gcc		5.94	6.00	289,252	2579	0%	0%	0%	100%
reverse-complement									
Python CPython		6.75	6.75	553,172	288	0%	0%	0%	100%
C GNU gcc		1.37	1.37	125,196	722	0%	0%	1%	99%
k-nucleotide									
Python CPython		426.01	427.19	439,448	475	0%	0%	0%	100%
C GNU gcc		51.86	51.87	153,692	2439	0%	0%	0%	100%
binary-trees									
Python CPython		515.84	515.81	221,620	365	0%	0%	1%	100%
C GNU gcc		12.25	12.25	99,532	850	0%	0%	1%	100%
n-body									
Python CPython		1,245.03	1,244.96	3,124	1105	0%	0%	0%	100%
C GNU gcc		23.52	23.53	412	1429	0%	0%	0%	100%
spectral-norm									
Python CPython		796.93	796.89	3,596	378	0%	0%	0%	100%
C GNU gcc		11.95	11.95	700	1139	0%	0%	0%	100%
mandelbrot									
Python CPython		2,050.85	2,050.73	18,592	425	0%	1%	0%	100%
C GNU gcc		24.31	24.32	30,204	822	0%	0%	0%	100%
fasta									
Python CPython		186.55	186.54	3,056	779	0%	1%	1%	100%
C GNU gcc		1.75	1.75	376	1321	1%	1%	0%	100%


binary-trees:


	×	Program Source Code	CPU secs	Elapsed secs	Memory KB	Code B	≅	CPU Load		
1.0	C	GNU gcc #7	12.60	12.61	149,584	850	0%	0%	0%	100%
1.3	C++	GNU g++ #6	15.95	15.95	296,220	892	0%	0%	0%	100%
1.3	ATS	#3	16.58	16.59	296,564	2143	0%	0%	0%	100%
1.3	Java	6 steady state #2	16.80	16.81	855,760	675	0%	0%	0%	100%
1.4	ATS		17.58	17.58	198,368	926	0%	0%	0%	100%
1.4	C	GNU gcc #2	17.99	18.00	99,268	1641	0%	0%	0%	100%
1.5	Scala	#4	18.48	18.58	572,604	494	0%	0%	0%	100%
1.5	Java	6 -server #2	18.90	19.01	563,476	603	0%	0%	0%	100%
1.7	Haskell	GHC	22.01	22.01	345,764	512	0%	0%	0%	100%
2.6	C	GNU gcc	33.25	33.25	131,640	706	0%	0%	0%	100%
2.7	C++	GNU g++ #2	34.57	34.57	197,840	553	0%	0%	0%	100%
2.9	Lisp	SBCL	36.51	36.52	404,612	612	0%	0%	0%	100%
3.0	Ada	2005 GNAT	37.40	37.40	198,136	955	0%	0%	0%	100%
3.4	Erlang	HIPE	42.75	42.75	374,284	441	0%	0%	0%	100%
3.5	Clean	#3	44.51	44.51	262,688	539	0%	0%	0%	100%
3.5	Pascal	Free Pascal	44.51	44.52	131,420	769	0%	0%	0%	100%
3.6	Fortran	Intel	44.77	44.77	131,544	826	0%	0%	0%	100%
4.0	OCaml	#2	50.90	50.90	157,648	784	0%	0%	1%	100%
4.1	OCaml	#5	51.95	51.96	228,556	496	0%	0%	0%	100%
4.9	Erlang	HIPE #2	62.10	62.10	349,432	499	0%	0%	0%	100%
4.9	Racket	#2	62.18	62.21	380,620	640	0%	0%	0%	100%
5.2	C	GNU gcc #5	65.12	65.20	560,376	963	1%	3%	0%	100%
6.0	F#	Mono #3	74.18	75.35	236,456	756	0%	0%	0%	100%
6.1	Racket		77.31	77.30	445,204	495	0%	0%	0%	100%
7.2	C#	Mono #2	89.39	90.19	463,044	650	0%	1%	0%	100%
7.4	JavaScript	V8	93.68	93.68	393,884	467	0%	0%	0%	100%
10	Java	6 -Xint #2	128.49	128.63	538,544	603	0%	0%	0%	100%
10	Haskell	GHC #4	131.81	131.81	439,984	604	0%	0%	0%	100%
11	Lua	LuaJIT #2	133.98	133.98	888,484	446	1%	1%	0%	100%
12	F#	Mono #2	152.24	152.90	262,652	685	0%	0%	0%	100%
15	Go	6g 8g	192.14	192.14	348,036	516	0%	0%	0%	100%
22	Ruby	1.9	274.92	274.79	599,568	412	0%	0%	0%	100%
26	Smalltalk	VisualWorks	5 min	5 min	208,980	722	0%	0%	0%	100%
42	Python	3 #6	8 min	8 min	1,225,120	626	0%	0%	1%	100%
51	Lua	#2	10 min	10 min	1,579,792	446	0%	0%	0%	100%

Conclusion: Python is perhaps the *worst possible language* in which to implement mathematical software!

Options:

- 1 Ignore this fact, cross fingers, and forge ahead anyways. In 2005, at least two projects similar to Sage did just that:

-  <http://tnt.math.se.tmu.ac.jp/nzmeth/>

-  <http://code.google.com/p/sympy/>

- 2 Write a new fast open source math-oriented interpreter and compiler. In 2002, a project started on that path:

<http://www.mathemagix.org>

- 3 Find a way to make Python fast.

The Python/C API

- In 2004, I considered various languages in which to implement an alternative to Magma for my use: GAP, Pari/GP, Perl, Ocaml, C++, Ruby, Haskell, Python, etc.
- <http://shootout.alioth.debian.org/> was influential.
- I read the Python/C API reference manual
- C and the Python API resemble the Magma development environment:
Magma = (a few millions lines of C) +
 (a few hundred thousand lines of Magma scripts)
- Maybe Python – which is a beautiful readable language – can also be made fast (=faster than Magma)????

Example: Trial Division in Python with Sage Preparing

```
def trial_division_python(n):
    if n == 1: return 1
    for p in [2, 3, 5]:
        if n%p == 0: return p
    # Algorithm: only trial divide by numbers that
    # are congruent to 1,7,11,13,17,29,23,29 mod 30=2*3*5.
    dif = [6, 4, 2, 4, 2, 4, 6, 2]
    m = 7; i = 1
    while m*m <= n:
        if n%m == 0: return m
        m += dif[i%8]
        i += 1
    return n
```

```
n = 2011*10000000019; n
```

```
20110000038209
```

```
timeit('trial_division_python(20110000038209)')
```

```
625 loops, best of 3: 693 µs per loop
```

Example: Trial Division in Python without Sage Preparing

```
%python

def trial_division_python(n):
    if n == 1: return 1
    for p in [2, 3, 5]:
        if n%p == 0: return p
    # Algorithm: only trial divide by numbers that
    # are congruent to 1,7,11,13,17,29,23,29 mod 30=2*3*5.
    dif = [6, 4, 2, 4, 2, 4, 6, 2]
    m = 7; i = 1
    while m*m <= n:
        if n%m == 0: return m
        m += dif[i%8]
        i += 1
    return n
```

```
timeit('trial_division_python(20110000038209r)')
```

625 loops, best of 3: 283 μ s per loop

Example: Trial Division using Python/C API (trial.c)

```
#include <Python.h>

static PyObject * trial_division(PyObject *self, PyObject *args) {
    unsigned long n, m=7, i=1, dif[8]={6,4,2,4,2,4,6,2};
    if (!PyArg_ParseTuple(args, "k", &n)) return NULL;
    if (n==1) return PyInt_FromLong(1);
    if (n%2==0) return PyInt_FromLong(2);
    if (n%3==0) return PyInt_FromLong(3);
    if (n%5==0) return PyInt_FromLong(5);
    // Algorithm: only trial divide by numbers that
    // are congruent to 1,7,11,13,17,29,23,29 mod 30=2*3*5.
    while (m*m <= n) {
        if (n%m == 0) return PyInt_FromLong(m);
        m += dif[i%8]; i += 1;
    }
    return PyInt_FromLong(m);
}

static PyMethodDef TrialMethods[] = {
    {"trial_division", trial_division, METH_VARARGS, "Trial division."},
    {NULL, NULL, 0, NULL} /* Sentinel */
};

PyMODINIT_FUNC inittrial(void) {
    (void) Py_InitModule("trial", TrialMethods);
}
```


Example: Trial Division using Python/C API (setup.py)

```
from distutils.core import setup, Extension
setup (name = 'TrialDivision', version = '1.0',
      description = 'This is a C extension',
      ext_modules = [Extension('trial', sources = ['trial.c'])])
```

```
timeit('trial.trial_division(20110000038209r)')
```

625 loops, best of 3: 6.16 μ s per loop

```
686/6.1
```

112.459016393443

```
trial.trial_division(20110000038209)
```

Traceback (click to the left of this block for traceback)

...

TypeError: argument 1 must be integer<k>, not
sage.rings.integer.Integer

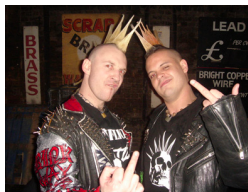
Orders of magnitude faster! But, note the problem with converting from Sage integers to unsigned long. Sense the pain!

<http://www.cosc.canterbury.ac.nz/greg.ewing/python/Pyrex/>

- Writing code using the Python/C API is very tricky to get right. Automate it!
- SWIG, SIP, Boost::Python, etc. – I tried to use them for months, but it was incredibly frustrating, due to the many layers of wrapping that get generated. I want *speed*. They solve a totally different problem.
- I found Pyrex, by Greg Ewing (of New Zealand). I was hooked: the combination of Pyrex + Python had potential to let me build what I wanted without having to write a new interpreter and/or compiler from scratch.

Cython

- Then I ran into all kinds of trouble with Pyrex and started writing patches. Some got in... and some very important ones got rejected with a statement that what I was fixing wasn't a bug (it was).
- I forked Pyrex: thus Cython <http://cython.org> was born.
- At the time, the top Google hit for “cython” was for a punk rocker in London...



Cython!!

With Cython you can do the following:

- 1 Speed up Sage, often by a factor of **about 100**.
- 2 Use existing C/C++ code from Sage, with almost no penalty.
- 3 Write new code that is *blazingly fast*.
- 4 But using Cython takes great care and genuine understanding – without both, you may write Cython code that is *much worse* than pure Python. Careful benchmarking is essential.

More on the Cython fork

- A got two people to get involved: Robert Bradshaw (my Ph.D. student) and Stefen Behnel. Both said they didn't want to be "lead developer", so I made them both the lead developers on the website I put up.
- They worked very, very hard on Cython. Also, via a Google Summer of Code, Dag S. got involved, and also works very hard on Cython – now the three of them are the primary developers.
- Example Additions:
 - 1 make it so Cython compiles nearly any Python code, e.g., list comprehension
 - 2 a *lot* of important speed optimizations
 - 3 important major extensions to the language: e.g., cpdef

Cython is now popular

- download numbers:
 - 1749 from pypi in last 6 months
 - 1500 visitors/week
 - but most people get cython from their Linux distribution or Sage.
- show website and list of contributors
- at euroscipy2010 nearly every talk mentioned Cython, and if they didn't somebody asked: why not? Reportedly scipy2010 was similar.

Cython in the Sage Library

Sage is the world's largest Cython project...

```
cd SAGE_ROOT/devel/sage/sage/
```

- There are 659 Cython files in Sage:

```
flat:sage wstein$ find . -print |grep "pyx\|pxi\|pxd" |wc -l  
659
```

- There are XX Cython modules in Sage

```
flat:sage wstein$ grep Extension ../module_list.py |wc -l  
283
```

- There are XX lines of Cython files

```
flat:sage wstein$ find . -print |grep "pyx\|pxi\|pxd" | xargs c  
313548
```

- There are XX unique lines of Cython files

```
flat:sage wstein$ find . -print |grep "pyx\|pxi\|pxd" | xargs c  
150991
```

In case you're interested – Python in Sage

- There are 1192 Python files in Sage:

```
flat:sage wstein$ find . -print |grep ".py$" |wc -l  
1192
```

- There are XX lines of Cython files

```
flat:sage wstein$ find . -print |grep ".py$" | xargs cat | wc -l  
635164
```

- There are XX unique lines of Cython files

```
flat:sage wstein$ find . -print |grep ".py$" | xargs cat | sort |  
309430
```

Yeah... the Sage library is pretty daunting:

```
flat:sage wstein$ find . -print |grep "pyx\|pxi\|pxd\|.py$" | xa  
948712
```

```
flat:sage wstein$ find . -print |grep "pyx\|pxi\|pxd\|.py$" | xa  
453612
```


Sample Enhancements

- Dependency checker: We (=Robert Bradshaw, Craig Citro, Gonzalo Tornaria, me) each wrote a sophisticated dependency checker just for Sage, so that if you modify one of these 659 Cython files, then type "sage -br", exactly the right cython files are rebuilt. Bradshaw is currently making this part of Cython.
- Also, we have a parallel build system, so if you do

```
export MAKE="make -j8"
sage -b
```

then 8 cython files will be compiled in parallel.

Wrap C/C++ libraries

```
$ cd SAGE_ROOT/devel/sage/sage/libs/; ls
```

```
__init__.py ecl.pyx libecm.c mpmath ratpoints.c  
all.py flint libecm.pyx mwrnk ratpoints.pxd  
cremona fplll linbox ntl ratpoints.pyx  
ecl.c ginac m4ri.pxd pari singular  
ecl.pxd gmp mpfr.pxd polybori symmetrica
```

These define very fast C level bindings for many libraries, such as NTL, Pari, Singular, etc.

How Cython is used in Sage

Look in

```
SAGE_ROOT/devel/sage/sage/rings/  
    polynomial/polynomial_zmod_flint.pyx
```

I tried benchmarking this last night, and it turns out that arithmetic is *very, very* slow because of how it is written...

Benchmarking is CRITICAL. You can easily write code in Cython, think you are very cool, and end up making Sage far slower. So watch out.

Questions