IPython: beyond the simple shell

Fernando Pérez
Fernando.Perez@berkeley.edu

Helen Wills Neuroscience Institute, UC Berkeley

SciPy India 2010
Dec 13, 2010
Why IPython?

(something other than “I’d rather not finish my dissertation”)
Why IPython?

(something other than “I’d rather not finish my dissertation”)
I is for interactive...

In scientific computing, we typically don’t know what we’re doing.

Scientific computing $\iff$ Exploratory computing
I is for interactive...

In scientific computing, we typically don’t know what we’re doing.

Scientific computing ⇔ *Exploratory* computing
I is for interactive...

In scientific computing, we typically don’t know what we’re doing.

Scientific computing ⇔ *Exploratory* computing
Interactive systems

Features

- **Execute/explore cycle instead of edit/compile/run**
- **Rich Libraries**
- **Plotting and data visualization**

Examples

- **Mathematica/Maple**: symbolic, now numerics...
- **IDL/Matlab**: numerics, image processing, ...
- **Unix system shell**: file management/text processing.
Interactive systems

Features

- Execute/explore cycle instead of edit/compile/run
- Rich Libraries
- Plotting and data visualization

Examples

- Mathematica/Maple: symbolic, now numerics...
- IDL/Matlab: numerics, image processing, ...
- Unix system shell: file management/text processing.
Python: an excellent *base* for an interactive scientific system

- Dynamic code evaluation
- No variable declarations
- Powerful *introspection*
- Very *regular* object model
- Excellent *string* processing
Python: an excellent *base* for an interactive scientific system

- **Dynamic** code evaluation
- No variable declarations
- **Powerful** introspection
- Very **regular** object model
- Excellent **string** processing
I said a base...

dreamweaver[~] > python
Python 2.6.6 (r266:84292, Sep 15 2010, 16:22:56)
[GCC 4.4.5] on linux2
Type "help", "copyright", "credits" or "license" for more information.

>>> ls
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ls' is not defined

>>>
Mmh, introspection?

```
dreamweaver[~]> python
Python 2.6.6 (r266:84292, Sep 15 2010, 16:22:56)
[GCC 4.4.5] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> ls
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ls' is not defined
>>> os?
  File "<stdin>", line 1
    os?
      ^
SyntaxError: invalid syntax
>>> 
```
Basic comforts?

dreamweaver[~] > python
Python 2.6.6 (r266:84292, Sep 15 2010, 16:22:56)
[GCC 4.4.5] on linux2
Type "help", "copyright", "credits" or "license" for more information.

>>> ls
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ls' is not defined

>>> os?
  File "<stdin>", line 1
os?
  ^
SyntaxError: invalid syntax

>>> execfile('~/scratch/err.py')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory: '~/scratch/err.py'

>>>
Useful error info

Python 2.6.6 (r266:84292, Sep 15 2010, 16:22:56)
[GCC 4.4.5] on linux2
Type "help", "copyright", "credits" or "license" for more information.

>>> ls
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ls' is not defined

>>> os?
  File "<stdin>", line 1
    os?
    ^
SyntaxError: invalid syntax

>>> execfile('~/scratch/err.py')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory: '~/scratch/err.py'

>>> execfile('/home/fperez/scratch/err.py')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "/home/fperez/scratch/err.py", line 9, in <module>
      foo33
NameError: name 'foo33' is not defined

>>>
We can do better...
My files, thankyouverymuch

```
dreamweaver[~] > ipython
Python 2.6.6 (r266:84292, Sep 15 2010, 16:22:56)
Type "copyright", "credits" or "license" for more information.

IPython 0.11.dev -- An enhanced Interactive Python.
?    -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help    -> Python's own help system.
oject?  -> Details about 'object', use 'object??' for extra details.

In [1]: ls ~/scratch/er*py
/home/fperez/scratch/err25.py   /home/fperez/scratch/error.py*
/home/fperez/scratch/err_comps.py /home/fperez/scratch/err.py

In [2]:  
```
OS routines for Mac, NT, or Posix depending on what system we're on.

This exports:
- all functions from posix, nt, os2, or ce, e.g. unlink, stat, etc.
- os.path is one of the modules posixpath, or ntpath
- os.name is 'posix', 'nt', 'os2', 'ce' or 'riscos'
- os.curdir is a string representing the current directory ('.' or ':')
- os.pardir is a string representing the parent directory ('..' or ':')
- os.sep is the (or a most common) pathname separator ('/' or ':' or '\\')
- os.extsep is the extension separator ('.' or '/')
- os.altsep is the alternate pathname separator (None or '/')
- os.pathsep is the component separator used in $PATH etc
- os.linesep is the line separator in text files ('\r' or '\n' or '\r\n')
- os.defpath is the default search path for executables
- os.devnull is the file path of the null device ('/dev/null', etc.)

Programs that import and use 'os' stand a better chance of being
Utilities needed to emulate Python's interactive interpreter.

# Inspired by similar code by Jeff Epler and Fredrik Lundh.

```python
import sys
import traceback
from codeop import CommandCompiler, compile_command

__all__ = ['InteractiveInterpreter', 'InteractiveConsole', 'interact', 'compile_command']

def softspace(file, newvalue):
    oldvalue = 0
    try:
        oldvalue = file.softspace
    except AttributeError:
        pass
    try:
        file.softspace = newvalue
```

lines 1-28
When things go wrong

In [13]: run ~/scratch/error
reps: 5

ValueError: Traceback (most recent call last)
/home/fperez/scratch/error.py in <module>()
    70     if __name__ == '__main__':
    71         #explode()

--> 72     main()
    73     g2='another global'

/home/fperez/scratch/error.py in main()
    60     array_num = zeros(size,'d')
    61     for i in xrange(reps):

--> 62         RampNum(array_num, size, 0.0, 1.0)
    63     RNtime = time.clock()-t0
    64     print 'RampNum time:', RNtime

/home/fperez/scratch/error.py in RampNum(result, size, start, end)
    43         tmp = zeros(size+1)
    44         step = (end-start)/(size-1-tmp)

--> 45         result[:] = arange(size)*step + start
    46
    47 def main():

ValueError: shape mismatch: objects cannot be broadcast to a single shape

In [14]:
Plotting at the console

```python
In [1]: import math, numpy
In [2]: from scipy.integrate import quad
In [3]: from scipy.special import j0
In [4]: def j0i(x):
   ...:     """Integral form of J_0(x)"""
   ...:     def integrand(phi):
   ...:         return math.cos(x*math.sin(phi))
   ...:     return (1.0/math.pi)*quad(integrand,0,math.pi)[0]
   ...
In [5]: x = numpy.linspace(0,20,200)  # sample grid: 200 points between 0 and 20
In [6]: y = j0(x)  # sample J0 at all values of x
In [7]: x1 = x[:10]  # subsample the original grid every 10th point
In [8]: y1 = map(j0i,x1)  # evaluate the integral form at all points in x1
In [9]: # Make a plot with these values (the ; suppresses output)
In [10]: plot(x,y,label="J_0(x)")
In [11]: plot(x1,y1,'ro',label="J_0-integ")
In [12]: axhline(0,color='green',label='nolegend_')
In [13]: title(r'\textbf{Verify $J_0(x) = \frac{1}{x} \int_0^\pi \cos(x \sin \phi) \, d\phi$}')
In [14]: xlabel('x')
In [15]: legend()
In [16]: matplotlib.figure.Figure instance at 0x4630042c
```

Out[16]: `<matplotlib.figure.Figure instance at 0x4630042c>`
A new, rich console (Qt based)
In a web browser

```
In [16]:
    t = linspace(-pi, pi, 1024)
    s = sin(10*t) / t
    plot(t, s, linewidth=1.0)
    xlabel('time (s)')
    ylabel('voltage (mV)')
    title('About as simple as it gets, folks')
    grid(True)
```
Getting all the power from interactive computing in Python

1. A better Python shell: object introspection, system access, ’magic’ commands, ...

2. A flexible, embeddable interpreter:
   1. debugging, mix batch/interactive work.
   2. build custom systems based on Python with new syntax, etc.


4. A rich toolkit: terminal, Qt console, HTTP client.

5. High level (and interactive!) parallel computing interfaces.
How did we get here?
A brief history of IPython

October/November 2001: “just a little afternoon hack“

- My own $PYTHONSTARTUP hack:
  - ipython-0.0.1.py: 259 lines of code.
- IPP (Interactive Python Prompt) by Janko Hauser (Oceanography)
- LazyPython by Nathan Gray (CS Caltech)
- First break Feb 2002, v0.0.2, ~8K LOC.

2002: Ignore John Hunter’s Gnuplot support patches

- ... let there be matplotlib
- and the world rejoiced.
- (actually finish my PhD!)
How did we get here?
A brief history of IPython

- **October/November 2001: “just a little afternoon hack“**
  - My own $PYTHONSTARTUP$ hack:
    - ipython-0.0.1.py: 259 lines of code.
  - IPP (Interactive Python Prompt) by Janko Hauser (Oceanography)
  - LazyPython by Nathan Gray (CS Caltech)
  - First break Feb 2002, v0.0.2, ~8K LOC.

- **2002: Ignore John Hunter’s Gnuplot support patches**
  - ... let there be matplotlib
  - and the world rejoiced.
  - (actually finish my PhD!)
2004: Brian Granger, Min Ragan-Kelley  
Physics @ U. Santa Clara  
- Killer team for parallel/networking ideas.  
- Min’s senior undergrad thesis (Physics @ U. Santa Clara, CA):  
  - first parallel implementation

2005: Ville Vainio  
- Core maintenance while we worked on machinery for parallelism.

2008: Gaël Varoquaux: ipythonx  
- WX widget: standalone and embeddable (MayaVi)

2008: Laurent Dufréchou: ipython-wx  
- WX app: standalone rich console
- **2004**: Brian Granger, Min Ragan-Kelley
  Physics @ U. Santa Clara
  - Killer team for parallel/networking ideas.
  - Min’s senior undergrad thesis (Physics @ U. Santa Clara, CA):
    - first parallel implementation

- **2005**: Ville Vainio
  - Core maintenance while we worked on machinery for parallelism.

- **2008**: Gaël Varoquaux: ipythonx
  - WX widget: standalone and embeddable (MayaVi)

- **2008**: Laurent Dufréchou: ipython-wx
  - WX app: standalone rich console
2004: Brian Granger, Min Ragan-Kelley
Physics @ U. Santa Clara

- Killer team for parallel/networking ideas.
- Min’s senior undergrad thesis (Physics @ U. Santa Clara, CA):
  - first parallel implementation

2005: Ville Vainio

- Core maintenance while we worked on machinery for parallelism.

2008: Gaël Varoquaux: ipythonx

- WX widget: standalone and embeddable (MayaVi)

2008: Laurent Dufréchou: ipython-wx

- WX app: standalone rich console
• **2004: Brian Granger, Min Ragan-Kelley**  
  *Physics @ U. Santa Clara*
  
  - Killer team for parallel/networking ideas.
  - Min’s senior undergrad thesis (Physics @ U. Santa Clara, CA):
    - first parallel implementation

• **2005: Ville Vainio**
  
  - Core maintenance while we worked on machinery for parallelism.

• **2008: Gaël Varoquaux: ipythonx**
  
  - WX widget: standalone and embeddable (*MayaVi*)

• **2008: Laurent Dufréchou: ipython-wx**
  
  - WX app: standalone rich console
February: discover ØMQ (ZeroMQ)
- A remarkable networking library in C++
- Sockets on steroids. Sockets done right.
- 3 days later, Brian has Cython bindings going on github!

March: proof of concept of shell over ØMQ
- 2-day sprint with Brian
- Simple, clean, performant. We’re thrilled.

May: switch from Bazaar/Launchpad to Git/Github
- Learn Git NOW.
- Github rocks. Seriously.

Google Summer of Code: 2 real prototypes on ØMQ
- Omar Zapata (U. de Antioquia - Colombia, CS): terminal.
- Gerardo Gutiérrez (U. de Antioquia - Colombia, Physics): Qt notebook.
February: discover ØMQ (ZeroMQ)
- A remarkable networking library in C++
- Sockets on steroids. _Sockets done right._
- 3 days later, Brian has Cython bindings going on github!

March: proof of concept of shell over ØMQ
- 2-day sprint with Brian
- Simple, clean, performant. We’re thrilled.

May: switch from Bazaar/Launchpad to Git/Github
- Learn Git _NOW_.
- Github rocks. Seriously.

Google Summer of Code: 2 real prototypes on ØMQ
- Omar Zapata (U. de Antioquia - Colombia, CS): terminal.
- Gerardo Gutiérrez (U. de Antioquia - Colombia, Physics): Qt notebook.
February: discover ØMQ (ZeroMQ)
- A remarkable networking library in C++
- Sockets on steroids. Sockets done right.
- 3 days later, Brian has Cython bindings going on github!

March: proof of concept of shell over ØMQ
- 2-day sprint with Brian
- Simple, clean, performant. We’re thrilled.

May: switch from Bazaar/Launchpad to Git/Github
- Learn Git NOW.
- Github rocks. Seriously.

Google Summer of Code: 2 real prototypes on ØMQ
- Omar Zapata (U. de Antioquia - Colombia, CS): terminal.
- Gerardo Gutiérrez (U. de Antioquia - Colombia, Physics): Qt notebook.
2010

- **February**: discover ØMQ (ZeroMQ)
  - A remarkable networking library in C++
  - Sockets on steroids. **Sockets done right.**
  - 3 days later, Brian has Cython bindings going on github!

- **March**: proof of concept of shell over ØMQ
  - 2-day sprint with Brian
  - Simple, clean, performant. We’re thrilled.

- **May**: switch from Bazaar/Launchpad to Git/Github
  - Learn Git **NOW**.
  - Github rocks. Seriously.

- **Google Summer of Code**: 2 real prototypes on ØMQ
  - Omar Zapata (U. de Antioquia - Colombia, CS): terminal.
  - Gerardo Gutiérrez (U. de Antioquia - Colombia, Physics): Qt notebook.
2010 Summer/Fall
Enthought sponsorship, things kick into high gear.

- **Rich Qt Console**
  - Evan Patterson (Caltech physics)

- **Full kernel over ØMQ**
  - Brian Granger (now at Cal Poly San Luis Obispo physics) and FP.

- **Implement parallel machinery over ØMQ**
  - Min Ragan-Kelley (now at UC Berkeley, plasma physics)
  - (btw, this is *insane* – though par for the course for Min)

- **HTML/JavaScript frontend**
  - James Gao (UC Berkeley neuroscience)

- **Python 3 port of core terminal code**
  - Thomas Kluyver
(Incomplete) Cast of Characters

- **Brian Granger** - Physics, Cal State San Luis Obispo
- **Min Ragan-Kelley** - UC Berkeley
- **Robert Kern** - Enthought
- **Jörgen Stenarson** - Sweden.
- **Stefan van der Walt** - Applied Math, U. Stellenbosch, South Africa
- **John Hunter** - TradeLink Securities, Chicago.
- **Satra Ghosh** - MIT Neuroscience
- **Gaël Varoquaux** - Neurospin (Orsay, France)
- **Ville Vainio** - CS, Tampere University of Technology, Finland
- **Barry Wark** - Neuroscience, U. Washington.
- **Ondrej Certik** - Physics, U Nevada Reno
- **Laurent Dufréchou** - France
- **Darren Dale** - Cornell
- **Justin Riley** - MIT
- **James Gao** - UC Berkeley
- **Mark Voorhies** - UC San Francisco
- **Thomas Kluyver**
- **Thomas Spura** - Fedora project
- **Many more! (~60 commit authors)**
Some quick stats. http://www.ohloh.net/p/ipython

Ohloh Analysis Summary

- Mostly written in Python
- Mature, well-established codebase
- Increasing year-over-year development activity
- Large, active development team
- Well-commented source code

Updated 13 Oct 2010 07:10 UTC

Project Cost

This calculator estimates how much it would cost to hire a team to write this project from scratch. More »

<table>
<thead>
<tr>
<th>Include</th>
</tr>
</thead>
<tbody>
<tr>
<td>Markup And Code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Codebase</th>
</tr>
</thead>
<tbody>
<tr>
<td>77,003</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Effort (est.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>19 Person Years</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Avg. Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>$55000</td>
</tr>
<tr>
<td>$1,041,279</td>
</tr>
</tbody>
</table>
**Licenses**

Ohloh searches the source code for individual license declarations. These licenses can differ from the project's official license.

- **BSD Copyright**: 257 files
- **MIT License**: 5 files
- **GNU General Public License 2**: 2 files
- **GNU Lesser General Public License v3**: 1 file
- **Apache License 2.0**: 1 file
- **Python 2.4.2 license**: 1 file

**Languages**

Ohloh analyzes the project source code and determines the language of each line of code, excluding comments and blanks.

- **Python**: 92%
- **XML**: 5%
- **Other**: 3%

**Lines of Code**

- blanks
- comments
- code

[Graph showing lines of code from Jan 00 to Jan 10]
<table>
<thead>
<tr>
<th>Language</th>
<th>Code Lines</th>
<th>Comment Lines</th>
<th>Comment Ratio</th>
<th>Blank Lines</th>
<th>Total Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>71,224</td>
<td>43,335</td>
<td>37.8%</td>
<td>27,576</td>
<td>142,135</td>
</tr>
<tr>
<td>XML</td>
<td>3,931</td>
<td>0</td>
<td>0.0%</td>
<td>115</td>
<td>4,046</td>
</tr>
<tr>
<td>Emacs Lisp</td>
<td>522</td>
<td>364</td>
<td>41.1%</td>
<td>89</td>
<td>975</td>
</tr>
<tr>
<td>CSS</td>
<td>498</td>
<td>11</td>
<td>2.2%</td>
<td>91</td>
<td>600</td>
</tr>
<tr>
<td>Perl</td>
<td>401</td>
<td>440</td>
<td>52.3%</td>
<td>208</td>
<td>1,049</td>
</tr>
<tr>
<td>Make</td>
<td>252</td>
<td>49</td>
<td>16.3%</td>
<td>107</td>
<td>408</td>
</tr>
<tr>
<td>shell script</td>
<td>143</td>
<td>106</td>
<td>42.6%</td>
<td>63</td>
<td>312</td>
</tr>
<tr>
<td>HTML</td>
<td>130</td>
<td>0</td>
<td>0.0%</td>
<td>65</td>
<td>195</td>
</tr>
<tr>
<td>Vim Script</td>
<td>124</td>
<td>2</td>
<td>1.6%</td>
<td>17</td>
<td>143</td>
</tr>
<tr>
<td>Objective-C</td>
<td>30</td>
<td>7</td>
<td>18.9%</td>
<td>12</td>
<td>49</td>
</tr>
</tbody>
</table>
Actively developed: Git/GitHub are amazing
## Projects using IPython

### Scientific
- **PyRAF**: Space Telescope Science Institute
- **CASA**: National Radio Astronomy Observatory.
- **Ganga**: CERN.
- **PyMAD**: neutron spectrometer, Institut Laue Langevin.
- **Sardana**: European Synchrotron Radiation Facility.
- **ASCEND**: engineering modeling (Carnegie Mellon).
- **JModelica**: dynamical systems.
- Denver Aerosol Sources and Health (**DASH**), CU Boulder.
- **PyIMSL Studio**, by Visual Numerics.
- **Trilinos**: Sandia National Lab.
- **Sage**: open source mathematics.
- **Pymerase**: microarray gene expression.

### Web/Other
- **Django** web framework.
- **Turbo Gears** web framework.
- **Pylons** web framework
- **Zope** and **Plone** CMS.
- Axon Shell, BBC **Kamaelia**.
- **Schevo** database.
- **Pitz**: distributed task/bug tracking.
- **iVR** (interactive Virtual Reality).
- **Movable Python** (portable Python environment).
- ...
A little detour

Python and parallel computing
Parallel computing: why should we care?

Because reality looks like this:

Sources: Intel, Microsoft (Sutter), Stanford (Olukotun, Hammond) & Berkeley (Yelick)
We can’t escape thermodynamics

Moore’s Law Extrapolation:
Power Density for Leading Edge Microprocessors

<table>
<thead>
<tr>
<th>Year</th>
<th>Power Density (Watts/cm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996</td>
<td>10</td>
</tr>
<tr>
<td>1998</td>
<td>100</td>
</tr>
<tr>
<td>2000</td>
<td>1000</td>
</tr>
<tr>
<td>2002</td>
<td>5000</td>
</tr>
<tr>
<td>2004</td>
<td>10000</td>
</tr>
<tr>
<td>2006</td>
<td>50000</td>
</tr>
<tr>
<td>2008</td>
<td>100000</td>
</tr>
</tbody>
</table>

Rocket Nozzle
Nuclear Reactor
Hot Plate

Power Density Becomes Too High to Cool Chips Inexpensively

Sources: Shekhar Borkar, Intel Corp & Kathy Yelick, UC Berkeley

The vendor’s solutions

- Multicore chips: everywhere (soon in your phone)
- Graphics cards: hundreds of specialized processors per card.
- High-density clusters: SiCortex (> 5000 processors in a cabinet).
We can’t escape thermodynamics

The vendor’s solutions
- Multicore chips: everywhere (soon in your phone)
- Graphics cards: hundreds of specialized processors per card.
- High-density clusters: SiCortex (> 5000 processors in a cabinet).
The infamous Global Interpreter Lock in CPython

Only one thread can modify Python state/variables at a time

- Historical reasons, simplicity of implementation
- All attempts at removing it have failed
  - 2× loss of performance is not acceptable
- Threads only good for i/o bound tasks.
- Mostly useless for CPU-bound ones.
- Can operate on pre-allocated arrays, but:
  - code must be in C/C++/Fortran/Cython
  - be very careful with locking if code is not atomic at Python level

The best possible reference on the GIL: David Beazley’s work

http://www.dabeaz.com/GIL
The infamous Global Interpreter Lock in CPython

Only one thread can modify Python state/variables at a time

- Historical reasons, simplicity of implementation
- All attempts at removing it have failed
  - $2\times$ loss of performance is not acceptable
- Threads only good for i/o bound tasks.
- Mostly useless for CPU-bound ones.
- Can operate on pre-allocated arrays, but:
  - code must be in C/C++/Fortran/Cython
  - be very careful with locking if code is not atomic at Python level

The best possible reference on the GIL: David Beazley’s work

http://www.dabeaz.com/GIL
In-process (mind the GIL)

- Data parallellism with threaded libraries
- Numpy/Scipy can use a threaded ATLAS
- Numexpr: a 'numpy VM'
- Theano: a library that thinks it's a compiler
- GPU-based solutions: PyCuda/PyOpenCL, scikits.cuda.
- Hand-written threaded code...

Out-of-process

- The multiprocessing module
- Python futures: coming in Python 3.2.
- Communicating Sequential Processes, ParallelPython, ... many more
- IPython
Parallelism in Python

- **In-process (mind the GIL)**
  - **Data parallelism** with threaded libraries
  - Numpy/Scipy can use a **threaded ATLAS**
  - **Numexpr**: a 'numpy VM'
  - **Theano**: a library that thinks it's a compiler
  - GPU-based solutions: **PyCuda/PyOpenCL, scikits.cuda**.
  - Hand-written threaded code...

- **Out-of-process**
  - The **multiprocessing** module
  - Python **futures**: coming in Python 3.2.
  - Communicating Sequential Processes, ParallelPython, ... many more
  - **IPython**
Network-aware IPython

The "IPython VM"

Kernel Controller
- Non-blocking
- World-visible

Private link

Kernel Engine
*May Block!*

Twisted

Networks
Parallel design

Alice  Client

Bob  Client

IPython Controller

IPython Engine  IPython Engine  IPython Engine  IPython Engine

→ Instructions
← Objects
Easy reuse and distribution of existing serial (‘normal’) codes.

High-level abstractions for ‘embarrassingly parallel’ problems.

- Direct execution of code over the network: multiengine interface.
- Out-of-the box task farming tools: task interface.

Mix and match parallel approaches:

- Synchronous or asynchronous execution of code on nodes.
- Task farming.
- Traditional Message Passing (MPI).

Everything can be used interactively.
IPVision: visual distributed computing
Michel Sanner, Jose Unpingco, Ananth Devulapalli [Ohio Supercomputing Center/OSU]
The future is now!

Fast, multi-process interactive Python
ØMQ - The Supersocket Library

“Sockets done right”

- Pure C++ library
- Python bindings in Cython (Brian Granger, Min RK)
- Python bindings run messaging in native threads - no GIL
- Abstractions are at the message delivery level, not the raw-bytes level.
- Socket types encapsulate messaging patterns
Interactive IPython on ØMQ

- Kernel raw_input
- Requests to kernel
- Kernel output broadcast
- Request/Reply direction
Same ideas for parallel computing
Min Ragan-Kelley
Phenomenal task latency
Back to the clients: a rich Qt Console

Enthought: sponsorship, Evan Patterson.

Feels like a console, runs like a GUI

- Inline and floating images
- Syntax highlighting, full multiline editing
- Session saving
  - HTML (with PNG or SVG)
  - PDF/printing
- Help viewer
- %magics, !system access, IPython...
- Detach/reattach support
Forward ØMQ to HTTP: a web frontend!

James Gao
Multiple users of one process instead of many processes for one user

These could be two different hosts on separate networks
A short demo
(time permitting)
Where to next?

IPython is now much more welcoming for new contributors!

- ZeroMQ-based architecture
  - solid design, plenty of work to do...

- **Release 0.11**: ASAP
  - But we still have blocker issues pending. **We need manpower!**
  - 0.10.1 was out recently.

- Clients
  - **Terminal**: 2-process client, curses client?
  - Continue improving **Qt console**: lots and lots of ideas.
  - Rich **Qt “notebook” client**: prototype exists (Google SoC 2010).
  - Collaborative **web client**.
  - Persistent notebook.

- Parallel architecture on ZeroMQ
  - The potential is enormous (competitive with MPI)

- **Python3 support**: experimental branch already on GitHub.
Thank You!

http://ipython.scipy.org

http://github.com/ipython