Interactive Parallel Computing with Python and IPython

Brian Granger
Research Scientist
Tech-X Corporation, Boulder CO

Collaborators: Fernando Perez (CU Boulder), Benjamin Ragan-Kelley (Undergraduate Student, SCU)

MSRI Workshop on Interactive Parallel Computation in Support of Research in Algebra, Geometry and Number Theory (January 2007)

Python and IPython

Python

- Freely available (BSD license).
- Highly portable: OS X, Windows, Linux, supercomputers.
- Can be used interactively (like Matlab, Mathematica, IDL)
- Simple, expressive syntax readable by human beings.
- Supports OO, functional, generic and meta programming.
- Large community of scientific/HPC users.
- Powerful built-in data types and libraries
  - Strings, lists, sets, dictionaries (hash tables)
  - Networking, XML parsing, threading, regular expressions...
- Larger number of third party libraries for scientific computing
- Easy to wrap existing C/C++/Fortran codes

IPython: Enhanced Interactive Python Shell

- Freely available (BSD license) @ http://ipython.scipy.org
- Goal: provide an efficient environment for exploratory and interactive scientific computing.
- The de facto shell for scientific computing in Python.
- Available as a standard package on every major Linux distribution. Downloaded over 27,000 times in 2006 alone.
- Interactive Shell for many other projects:
  - Math (SAGE)
  - Astronomy (PyRAF, CASA)
  - Physics (Ganga, PyMAD)
  - Biology (Pymerase)
  - Web frameworks (Zope/Plone, Turbogears, Django)

IPython: Capabilities

- Input/output histories.
- Interactive GUI control: enables interactive plotting.
- Highly customizable: extensible syntax, error handling,...
- Interactive control system: magic commands.
- Dynamic introspection of nearly everything (objects, help, filesystem, etc.)
- Direct access to filesystem and shell.
- Integrated debugger and profiler support.
- Easy to embed: give any program an interactive console with one line of code.
- Interactive Parallel/Distributed computing...

Traditional Parallel Computing

Compiled Languages

- C/C++/Fortran are FAST for computers, SLOW for you.
- Everything is low-level, you get nothing for free:
  - Only primitive data types.
  - Few built-in libraries.
  - Manual memory management: bugs and more bugs.
  - With C/C++ you don't even get built-in high performance numerical arrays.
- No interactive capabilities:
  - Endless edit/compile/execute cycles.
  - Any change means recompilation.
- Awkward access to plotting, 3D visualization, system shell.

Message Passing Interface: MPI

- Pros
  - Robust, optimized, standardized, portable, common.
  - Existing parallel libraries (FFTW, ScaLAPACK, Trillinos, PETSc)
  - Runs over Ethernet, Infiniband, Myrinet.
  - Great at moving data around fast!
- Cons
  - Trivial things are not trivial. Lots of boilerplate code.
  - Orthogonal to how scientists think and work.
  - Static: load balancing and fault tolerance are difficult to implement.
  - Emphasis on compiled languages.
  - Non-interactive and non-collaborative.
  - Doesn't play well with other tools: GUIs, plotting, visualization, web.
  - Labor intensive to learn and use properly.

Case Study: Parallel Jobs at NERSC in 2006

- NERSC = DOE Supercomputing center at Lawrence Berkeley National Laboratory
- Seaborg = IBM SP RS/6000 with 6080 CPUs
  - 90% of jobs used less than 113 CPUs
  - Only 0.26% of jobs used more than 2048 CPUs
- Jacquard = 712 CPU Opteron system
  - 50% of jobs used fewer than 15 CPUs
  - Only 0.39% of jobs used more than 256 CPUs

* Statistics (used with permission) from NERSC users site (http://www.nersc.gov/nusers)

Realities

- Developing highly parallel codes with these tools is extremely difficult and time consuming.
- When it comes to parallel computing WE (the software developers) are often the bottleneck.
- We spend most of our time writing code rather than waiting for those “slow” computers.
- With the advent of multi-core CPUs, this problem is coming to a laptop/desktop near you.
- Parallel speedups are not guaranteed!

Our Goals with IPython

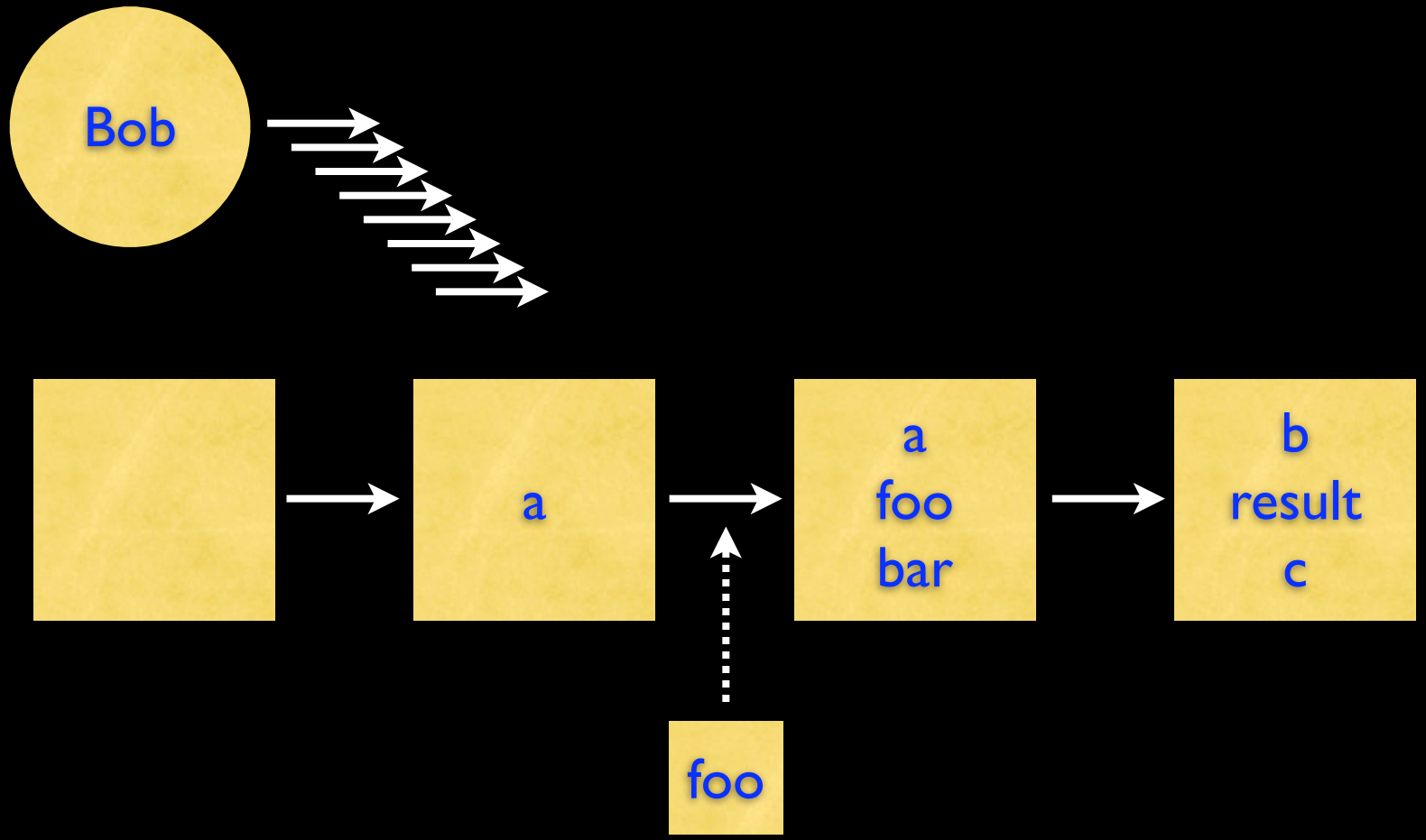
- Trivial parallel things should be trivial.
- Difficult parallel things should be possible.
- Make all stages of parallel computing fully interactive: development, debugging, testing, execution, monitoring,...
- Make parallel computing collaborative.
- More dynamic model for load balancing and fault tolerance.
- Seamless integration with other tools: plotting/ visualization, system shell.
- Also want to keep the benefits of traditional approaches:
  - Should be able to use MPI if it is appropriate.
  - Should be easy to integrate compiled code and libraries.
- Support many types of parallelism.

Computing With Namespaces

Namespaces

- Namespace = a container for objects and their unique identifiers.
- An instruction stream causes a namespace to evolve with time.
- Interactive computing: the instruction stream has a human agent as its runtime source at some level.
- A (namespace, instruction stream) is a higher level abstraction than a process or thread.
- Data in a namespace can be created inplace (by instructions) or by external I/O (disk, network).
- Thinking about namespaces allows us to abstract parallelism and interactivity in a useful way.

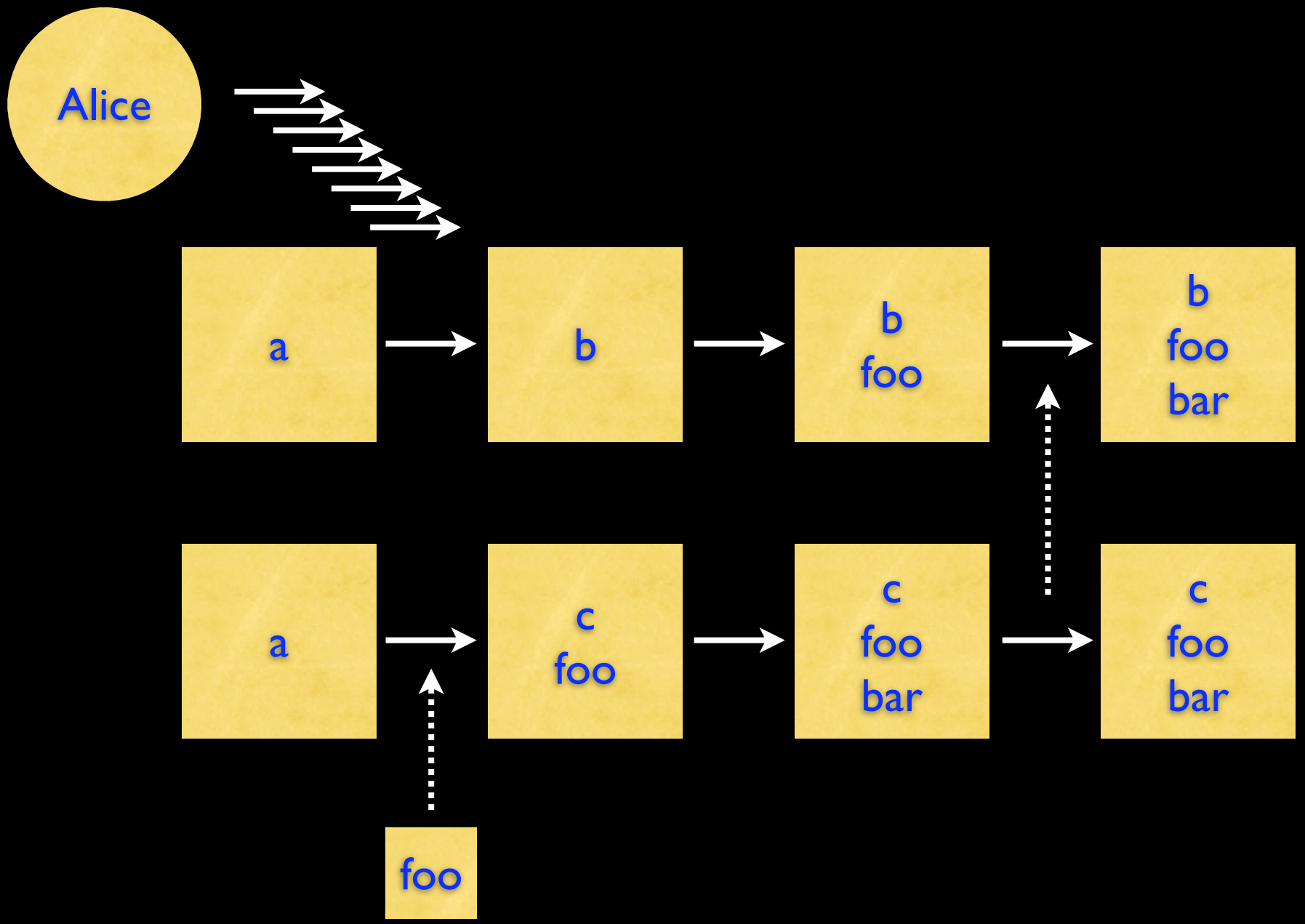
Serial Namespace Computing



Instructions

Data from Network/Disk

Parallel Namespace Computing

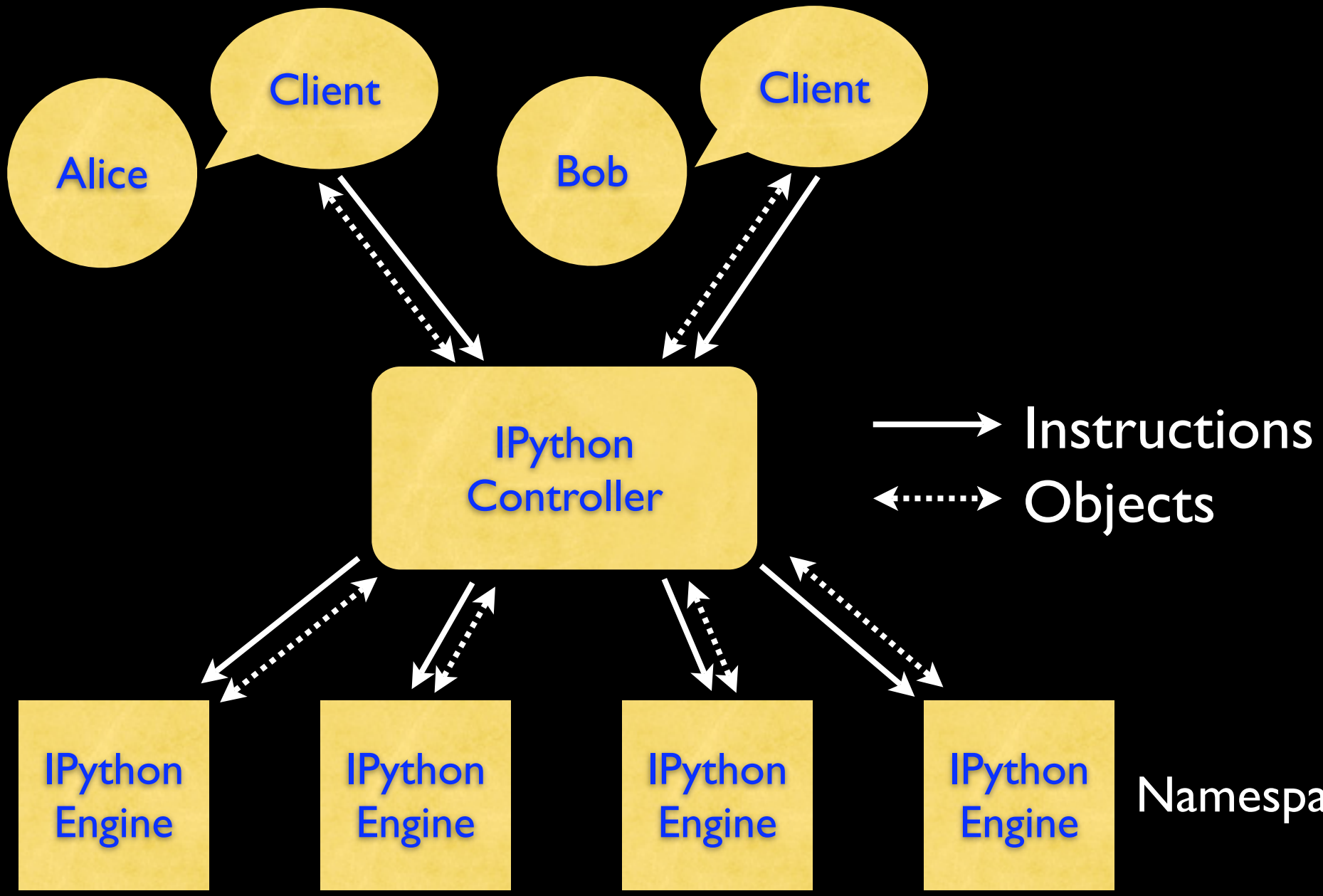


Important Points

- Requirements for Interactive Computation:
  - Alice/Bob must be able to send instruction stream to a namespace.
  - Alice/Bob must be able to push/pull objects to/from the namespace (disk, network).
- Requirements for Parallel Computation:
  - Multiple namespaces and instruction streams (for general MIMD parallelism).
  - Send data between namespaces (MPI is really good at this)
- Requirements for Interactive Parallel Computation:
  - Alice/Bob must be able to send multiple instruction streams to multiple namespaces.
  - Alice/Bob must be able to push/pull objects to/from the namespaces .

* These requirements hold for any type of parallelism

IPython's Architecture



Architecture Details

- The IPython Engine/Controller/Client are typically different processes. Why not threads?
- Can be run in arbitrary configurations on laptops, clusters, supercomputers.
- Everything is asynchronous. Can't hack this on as an afterthought.
- Must deal with long running commands that block all network traffic.
- Dynamic process model. Engines and Clients can come and go at will at any time*.

*Unless you are using MPI

Mapping Namespaces To Various Models of Parallel Computation

Key Points

- Most models of parallel/distributed computing can be mapped onto this architecture.
  - Message Passing
  - Task farming
  - TupleSpaces
  - BSP (Bulk Synchronous Parallel)
  - Google's MapReduce
  - ???
- With IPython's architecture all of these types of parallel computations can be done interactively and collaboratively.
- The mapping of these models onto our architecture is done using interfaces+adapters and requires very little code.

The IPython RemoteController Interface

Overview

- This is a low-level interfaces that gives a user direct and detailed control over a set of running IPython Engines.
- Right now it is the default way of working with Engines.
- Good for:
  - Coarse grained parallelism without MPI.
  - Interactive steering of fine grained MPI codes.
  - Quick and dirty parallelism.
- Not good for:
  - Load balanced task farming.
- Just one example of how to work with engines.

Start Your Engines...

```
> ipcluster -n 4
Starting controller: Controller PID: 385
Starting engines:    Engines PIDs:   [386, 387, 388, 389]
Log files: /Users/bgranger/.ipython/log/ipcluster-385-*

Your cluster is up and running.

For interactive use, you can make a Remote Controller with:

import ipython1.kernel.api as kernel
ipc = kernel.RemoteController(('127.0.0.1',10105))

You can then cleanly stop the cluster from IPython using:

ipc.killAll(controller=True)

You can also hit Ctrl-C to stop it, or use from the cmd line:

kill -INT 384
```

Startup Details

- ipcluster can also start engines on other machines using ssh.
- For more complicated setups we have scripts to start the controller (ipcontroller) and engines (ipengine) separately.
- We routinely:
  - Start engines using mpiexec/mpirun.
  - Start engines on supercomputers that have batch systems (PBS, Loadleveler) and other crazy things.
  - Not always trivial, but nothing magic going on.

Live Demo

Example 1: Analysis of Large Data Sets

- IPython is being used at Tech-X for analysis of large data sets.
- Massively parallel simulations of electrons in a plasma generate lots of data:
  - 10s-100s of Gb in 1000s of HDF5 files.
- Data analysis stages:
  - Preprocessing/reduction of data.
  - Run parallel algorithm over many parameters.
  - Coarse grained parallelism (almost trivial parallelizable)
- Core algorithm was parallelized in 2 days.
- Data analysis time reduced from many hours to minutes.
- Gain benefits of interactivity.

Example 2: Multiresolution Quantum Chemsitry

- A new family of algorithms for solving multidimensional PDEs which admit an integral formulation.
- Main target is the multiparticle Schrodinger equation: a linear, multivariable PDE where correlations are fundamental to the physics.
- Methods:
  - Nonlinear approximations: unconstrained representations of the wavefunction.
  - Adaptive multiresolution application of linear (integral and differential) operators in 3D, using Gaussian expansions for integral kernels.
  - Adaptive accuracy control.

Example 2: Multiresolution Quantum Chemsitry

- Minor modifications to our serial code and IPython machinery allowed us to easily distribute over a cluster.
- Next week will move to ORNL large systems.
- Talk to Fernando for details if you are interested.
- Robert Harrison will present the massively parallel evolution of these ideas on Friday here.

MPI

- Without full and robust MPI support, these tools would be a no-go for many applications
- Engines can be started using mpiexec and call MPI_Init. From then on, instruction streams sent to engines can contain arbitrary MPI calls.
- Can use MPI through:
  - Low-level C/C++/Fortran bindings
  - Python bindings (see http://mpi4py.scipy.org)
- Remains fully interactive/collaborative.
- Fully supported today!

Task Based Computing

- Common style of parallelism for loosely coupled or independent tasks.
- Great when dynamic load balancing is needed.
- Similar to distributed SAGE. Plan on working/talking with the SAGE developers about these ideas.
- This can be implemented by a very lightweight adapter around our core architecture. You get a lot for free. We take care of networking stuff.
- We have begun the initial work on this interface.

Stepping Back a Bit

- Event based systems are a nice alternative to threads:
  - Scale to multiple CPU systems.
  - Build asynchronous nature of things in at low level.
  - No deadlocks to worry about.
- The networking framework used by IPython and SAGE (Twisted) has an abstraction (called a Deferred) for a result that will arrive at some point in the future:
  - Like a promise in E
- We have an interface that uses Deferreds to encapsulate asynchronous results/errors.

Benefits of an Event Based System

- Arbitrary configurations of namespaces are immediately possible without worrying about deadlocks
  - Our test suite create a controller/client and multiple engines in a single process.
- Possibilities:
  - Hierarchies of client/controller/engine/client
  - Recursive systems.

Error Propagation

- Building a distributed system is easy...
- Unless you want to handle errors well.
- We have spent a lot of time working/thinking about this issue. Not always obvious what should happen.
- Our goal: error handling/propagation in a parallel/distributed context should be a nice analytic continuation of what happens in a serial context.
- Remote exceptions should propagate to the user in a meaningful manner.
- Policy of safety: don't ever let errors pass silently.

This Week

- All the core developers of IPython's parallel capabilities are here at the workshop.
- We will be working on the code and talking with people.
- We want feedback, good and bad. Lots of things to talk about and work on.
- Want to talk/work with SAGE developers and others on parallel/distributed things.

Conclusions

- Namespaces provide a useful starting point for thinking about interactive parallel computing.
- Interactivity is a separate question from the model of parallel computation.
- IPython provides interactivity for many kinds of parallelism.
- Lots of interesting questions about interactivity in event based systems to think about and explore.