Python in HPC

Supercomputing 2012

Presenters:

Andy R. Terrel, PhD
Texas Advanced Computing Center
University of Texas at Austin

Travis Oliphant, PhD
Continuum Analytics

Aron Ahmadia, PhD
Supercomputing Laboratory
King Abdullah University of Science and Technoglogy

Learning Python

There are a very large number of resources for learning the python language. The issue of course is finding a resource that is attractive to the correct mindset. Very often something that a programmer is able to learn from has little resonance with an artist. For this reason, I am going to list off a couple of places that I find particularly good, but of course I have been programming for over a decade.

While learning the language is important, the real value of Python is the numerous libraries and the community around the tools. With that said finding what your particular community is doing is pretty important, but there are a core set of tools that are used by most pythonistas. I list off a few of these tools for your consumption.

Also, because we like to see what people are doing with python, I list off a few shining examples of python in the wide world of scientific computing. I have to admit that there are a huge number of possibilities but few in the realm of HPC. At the very least, a HPC person should know Python for its scripting abilities but as the other sections of this document underscore, Python is a good candidate for HPC codes as well.

Python Tutorials

- Python Doc Tutorial: This is the tutorial written by the developers of Python, best for programmers.
- Think Python: A book for non-programmers.

Python Tools

- Enthought Python Distribution: A distribution of the most commonly used tools by the community (free for academics).
- NumPy: Fast array library for Python.
- SciPy: A collection of scientific libraries.
- MatPlotLib: A highly customizable 2D plotting library
- IPython: An interactive Python shell and parallel code manager. The IPython notebook has become very popular and allows users to use an interface similar to Mathematica on a supercomputer.

Python Stories

- SciPy Conferences: The series of conferences associated with the scientific Python community see recent videos at Next Day Video Youtube Channel or PyVideo.
- Python in Astronomy: Joshua Bloom from UC Berkeley gave a keynote talk at SciPy 2012 on "Python as Super Glue for the Modern Scientific Workflow"
- NumFocus User Stories: A foundation for scientific computing tools with a growing number of user stories.
- PyCLAW: A petascale application written in Python

Performance

Despite all the great features outlined above, the (mis)perception is that Python is too slow for HPC Development. While it is true that Python might not be the best language to write your tight loop and expect a high percentage of peak flop rate, it turns out that Python has a number of tools to help switch those lower-level languages.

To discuss performance I outline three sets of tools: profiling, speeding up the python code via c, and speeding up python via python. It is my view that Python has some of the best tools for looking at what your code's performance is then drilling down to the actual bottle necks. Speeding up code without profiling is about like trying to kill a deer with an uzi.

Python tools for profiling

- profile and cProfile modules: These modules will give you your standard run time analysis and function call stack. It is pretty nice to save their statistics and using the pstats module you can look at the data in a number of ways.
- kernprof: this tool puts together many routines for doing things like line by line code timing
- memory_profiler: this tool produces line by line memory foot print of your code.
- IPython timers: The `timeit` function is quite nice for seeing the differences in functions in a quick interactive way.

Speeding up Python

- **Cython**: cython is the quickest way to take a few functions in python and get faster code. You can decorate the function with the cython variant of python and it generates c code. This is very maintable and can also link to other hand written code in c/c++/fortran quite easily. It is by far the preferred tool today.
- **ctypes**: ctypes will allow you to write your functions in c and then wrap them quickly with its simple decoration of the code. It handles all the pain of casting from PyObjects and managing the gil to call the c function.

Other approaches exist for writing your code in C but they are all somewhat more for taking a C/C++ library and wrapping it in Python.

Python-only approaches

If you want to stay inside Python mostly, my advice is to figure out what data you are using and picking correct data types for implementing your algorithms. It has been my experience that you will usually get much farther by optimizing your data structures then any low level c hack. For example:

- **numpy**: a contingous array very fast for strided operations of arrays
- **numexpr**: a numpy array expression optimizer. It allows for multithreading numpy array expressions and also gets rid of the numerous temporaries numpy makes because of restrictions of the Python interpreter.
- **blist**: a b-tree implementation of a list, very fast for inserting, indexing, and moving the internal nodes of a list
- **pandas**: data frames (or tables) very fast analytics on the arrays.
- **pytables**: fast structured hierarchical tables (like hdf5), especially good for out of core calculations and queries to large data.

Scaling Python

Right now there are a few distributed Python tools but the list is growing rapidly. Here I just give a list the tools and some domain tools that are used in HPC that provide a Python interface.

Distibuted computing libraries

- **mpi4py**: Fastest most complete mpi python wrapper.
- **disco**: Python Hadoop-like framework.
- **IPython Parallel**: A mpi or zero-mq based parallel python.
- **pathos**: framework for heterogeneous computing

Domain specific libraries

- **petsc4py**: Python bindings for PETSc, the Portable, Extensible Toolkit for Scientific Computation.
- **slepc4py**: Python bindings for SLEPc, the Scalable Library for Eigenvalue Problem Computations.
- **tao4py**: Python bindings for TAO, the Toolkit for Advanced Optimization.
- **pyTrilinos**: Trilinos wrappers