Extensible Message Passing Application Development and Debugging with Python

David M. Beazley

University of Utah
Salt Lake City, Utah 84112
beazley@cs.utah.edu

April 3, 1997

Collaborators: P. Lomdahl, S. Zhou, B. Holian, T. Germann, N. Jensen
Background

The SPaSM Short-Range Molecular Dynamics Code

- Study of fracture, dislocations, and other material properties in solids.
- 1 - 300 million atom simulations in 3D
- Written in ANSI C
- Uses explicit message passing.
- 1993 IEEE Gordon Bell Prize

I have been working in the trenches...

- Parallel software development.
- Performance tuning and portability.
- Debugging.
Parallel Computing Woes

For several years, we tried to get better and better performance...

- But, we were getting crushed by the data being generated.
- Code development was becoming increasingly difficult.
- Debugging was a nightmare.

Concerns

- Why are parallel codes so hard to manage?
- Why are we unable to deal with the vast amounts of data?
- Why is debugging so difficult?
- Why do all of these things need to be so hard?

Our motivations

- Find a way to make parallel machines work better.
- Solve our data analysis and visualization problems.
- Maintain high performance while emphasizing simplicity.
It’s All About Software (well, mostly)

“The best performance improvement is the transition from the nonworking state to the working state” -- John Ousterhout.

• Just buying expensive hardware isn’t going to solve the problem.
• Software outlives most machines.
• If you can’t solve your problem, then all of the performance in the world isn’t going to matter too much.
• Solutions need to be cost-effective.
• Need to work with existing applications
The Python Language

Python is a freely available object oriented language
- Highly portable (Unix, Windows, MacOS)
- Small extensible core.
- Designed to be easily integrated with C/C++ code.
- Can be run interactively.
- Large number of extension modules already available.
- Being used increasingly in scientific applications.
- Source is available and can be used without restriction.
- Often compared to scripting languages such as Tcl and Perl.

To make our parallel application more usable, we are using Python as a high-level control language.
- Think MATLAB, Mathematica, IDL
- Interactive control, debugging, rapid prototyping, etc...
- The trick: making this work on a parallel machine.
Using Python with a C/C++ Application

A Typical Program

C Functions

main()

... 
initcond();
init_force();
...
for (i = 0; i < nsteps, i++) {
  force();
  integrate();
  redistribute();
  ...
}

Interactive Version

Python

C Functions

# A Python script
initcond()
init_force()
...
for i in range(0, nsteps):
  force()
  integrate()
  redistribute()
...
The Problem with Parallelizing Python

- Python interpreter and C/C++ program run independently on each node.
- C/C++ program uses message passing for communication (it was already a parallel application).
- Python knows nothing about message passing or the underlying implementation.

This is a problem!
- Python needs to execute commands and scripts.
- Python also needs to perform file I/O during normal operation.
Remapping Python I/O

Need to remap I/O operations within Python

- ~ 60000 lines of C source code.
- Not written for a parallel machine.
- We didn’t write it.

Strategy

- Write a library of I/O wrappers
- Implement wrappers over the message passing library (ie. MPI).
- Transparently install them into the Python source.
- Works for all but 1 line in Python source

Parallel I/O wrappers

PIO_fopen()
PIO_fclose()
PIO_fprintf()
PIO_open()
PIO_write()
PIO_fputs()
...

pstdio.h

#define fopen PIO_fopen
#define fclose PIO_fclose
#define fprintf PIO_fprintf
#define open PIO_open
#define write PIO_write
...

Python

/* Python.h */
#include <pstdio.h>
...
Implementation of I/O Wrappers

We run Python in a SPMD mode

Broadcast mode
- Single input stream
- Reader broadcasts to everyone
- Only one output stream (which can be selected from any node)
- Used for interactive mode

Broadcast write mode
- Each node can independently read files, but only one may write.

Message passing
- Used for broadcasting, synchronization, etc...
Current Implementation

I/O Wrappers
- ~ 1000 lines of C
- < 10 lines of modification to Python source
- Versions for CMMD, MPI, and Cray shared memory.

Platforms
- CM-5
- Cray T3D
- SP-2
- Sun Ultra. (With MPI)
Building Python Interfaces with SWIG

SWIG is a compiler we have developed to turn ANSI C/C++ declarations into scripting language interfaces.

- Supports Python, Tcl/Tk, Perl, and Guile.
- Uses ANSI C/C++ syntax and supports most C/C++ features (pointers, structures, classes, etc...)
- Automatic documentation generation
- Runs on Unix, Windows, and MacOS.
- Freely available and fully documented.

Completely automated. Building an interface requires no additional programming or knowledge of the nasty underlying details.

Given a C header file, SWIG automatically makes a Python interface.
Putting it all together: The SPaSM code

Application consists of a variety of modules
- Simulation
- Visualization
- Data Analysis

SWIG used to bind these to parallel Python
- Completely automated
- Performed during compilation

How it works
- C functions turn into Python commands (via SWIG)
- User controls system by running Python and issuing “commands”.
- In fact, it’s like MATLAB except that you’re running on a parallel machine.
Building and Running a Simulation

- Simulations now run inside Python
- Python interface automatically generated at compile time.

```python
# Simple Python simulation script
from spasm import *

nx = 50
ny = 50
nz = 15
...
ic_shock(nx, ny, nz, shock_vel, gap, ...)
init_lj(1, 1, cutoff)
set_boundary_periodic()
...
for i in range(0, nsteps):
    integrate_adv_coord(Dt)
    boundary()
    redistribute()
    force()
    integrate_adv_velocity(Dt)
    if (i % examine_freq) == 0:
        output_data()
        make_plots()
    ...
```
Interactive Message Passing

Message passing library can be wrapped with SWIG

Message passing operations can be typed interactively!
- Follows same rules as in C.
- Each node is running Python.
- Can deadlock the machine, etc...

Machine is running in a SPMD mode when used in this manner (ie. commands go to all nodes).

Tremendously useful as a debugging aid or for just playing around.

```python
# python
Starting Python on 32 processors
Python 1.3 (Aug 6 1996) [C]
>>> from pvm3 import *
>>> me = pvm_get_PE(pvm_mytid())
>>> nproc = pvm_gsize(""")
>>> if me == 0:
      ...    a = [1,2,3,4]
      ...  else:
      ...    a = []
>>> if me == 0:
      ...    for i in range(1,nproc):
      ...        pvm_initsend(PvmDataRaw)
      ...        pack_list(a)
      ...        pvm_send(i,1)
      ...  else:
      ...    pvm_recv(0,1)
      ...    a = unpack_list()
      ...
>>> 
```
Object-Oriented Visualization System

Have developed a high-performance visualization and data analysis system.

- Uses a simple graphics library we’ve developed.
- C used for high performance
- Python used for organization and configuration.

Runs in parallel and allows us to easily work with huge datasets.

- 100 million atoms on 512 node CM-5 (5-20 seconds)
- 35 million atoms on 16 node T3D (5-15 seconds)

Runs remotely and over slow network connections.
Internet Programming

Python is used extensively in Internet applications.

We’ve written a simple web-server entirely in Python
- Serves real-time simulation data.
- On-the-fly image generation.
- Configurable.
- Only 150 lines of code.
- Works on machines without web-servers and behind firewalls.

Allows remote monitoring of running simulations
- Simulations run for 10s to 100s of hours.
- Can check on its status periodically with any browser.
Results

Have been using this approach for approximately 2 years
- Has revolutionized the way in which we perform MD simulations.
- Allows data analysis and visualization to be tightly coupled with the simulation.
- We can now analyze and work with data from *ANY* simulation that we perform.
- Interactive application specific debugging.
- Code size has decreased by more than 25% while becoming more reliable.

Coding implications
- Encourages modular design.
- Improves reliability.
- Works with existing code.
- Maintains simplicity. In fact, our current system can be easily compiled without a Python interface at all.
Future Directions

This is a work in progress

- Code development is shifting away from MPPs to high-end symmetric multiprocessing systems (Our Sun Ultraserver in particular).
- Work on providing more powerful tools and novel data analysis methods.
- Internet applications. Primarily remote simulation monitoring and control.

Data management!

- Still need better tools for managing huge volumes of simulation data and images.
- Data mining?

Parallel computing concerns

- System designers need to realize that interactivity, I/O, and system tools are of great importance. (Not everyone runs Fortran batch jobs).
- Rewriting everything in a “parallel language” is not always desirable either.
Acknowledgments

- Los Alamos National Laboratory
- University of Utah
- Lawrence Livermore National Laboratory
- Cornell Theory Center
- Scientific Computing and Imaging Group
- Advanced Computing Laboratory
- Guido van Rossum (Python)
- SWIG Users
Availability

Python

- [http://www.python.org](http://www.python.org)
- Can get books at the local bookstore.

SWIG

- [http://www.cs.utah.edu/~beazley/SWIG](http://www.cs.utah.edu/~beazley/SWIG)
- Fully documented (> 200 pages)
- Being used in scientific applications, semiconductor CAD, database, financial, remote sensing, and a number of other applications.

SPaSM Information (graphics library, parallel Python, etc...)

- [http://bifrost.lanl.gov/software/](http://bifrost.lanl.gov/software/)