Python Best Practices on Blue Waters

Roland Haas, Victor Anisimov (NCSA)
Email: rhaas@illinois.edu
Python on Blue Waters

- HPC vendors have limited support of python on their platforms
- BWPY is NCSA supported python deployment on Blue Waters
  
  $\text{module load bwpy}$

- Other installations, such as Anaconda in your home directory, are not supported.

- BWPY resolves typical issues with python deployment
  - Lustre filesystem does not tolerate frequent open / close
  - Using MPI on Cray is different from that on a Linux cluster
  - Compiling numerous python packages is a demanding job
BWPY versioning

- BWPY uses major.minor.patch versioning.
  - Major versions are for major changes
    - Different default python version (including minor)
    - Possibly a self-contained glibc, requiring a complete rebuild
  - Minor is for package updates
  - Patch fixes problems, mostly keeping package versions the same, unless specific package versions are broken. New packages may be added.

  $ module load bwpy/x.y.z

- Current default: 1.2.4, latest: 2.0.1
### BWPY submodules

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bwpy-mpi</strong></td>
<td>MPI support enabled&lt;br&gt;(should only be used on compute nodes!)</td>
</tr>
<tr>
<td><strong>bwpy-libsci_mp</strong></td>
<td>BWPY built with OpenMP Cray BLAS libraries&lt;br&gt;(libsci_mp)</td>
</tr>
<tr>
<td><strong>bwpy-libsci_acc</strong></td>
<td>BWPY built with auto CUDA BLAS libraries&lt;br&gt;(libsci_acc)</td>
</tr>
<tr>
<td><strong>bwpy-visit</strong></td>
<td>BWPY’s VisIt&lt;br&gt;(requires older vtk, so is a separate module)</td>
</tr>
<tr>
<td><strong>bwpy-visit-mpi</strong></td>
<td>BWPY’s VisIt with MPI&lt;br&gt;(only supported on compute nodes!)</td>
</tr>
</tbody>
</table>

Default BLAS: MKL
Available python interpreters

- CPython 2.7 (alias: python2)
- CPython 3.5 (aliases: python, python3)
- Cpython 3.6
- Pypy
- Pypy3

Now with much improved CPython compatibility!

Can select interpreter by setting `EPYTHON` environment variable

```
$ export EPYTHON=python2.7
$ python --version
Python 2.7.14
```

Can set the default version of python by using virtualenv (explained later)
Behind the scenes

- BWPY is a Gentoo-Linux distribution mounted as a read-only disk image
  - Use `bwpy-environ` tool to mount the image and get access to apps
  - Image appears in `/mnt/bwpy` with subdirectories `{single,mpi}` etc.
  - Image is local to each process and its children
  - Use `bwpy-environ -- program [args ...]` to run a program
  - Can invoke `bwpy-environ` directly for debug purpose
What to expect

% which python
/usr/bin/python          # old interpreter that comes with operating system

% module add bwpy; which python
/sw/bw/bwpy/mnt/bin/python  # wrapper around bwpy-environ

% bwpy-environ -- which python
/mnt/bwpy/single/usr/bin/python  # actual binary

% which cmake
/usr/bin/cmake          # old cmake that comes with operating system

% module avail cmake
cmake/2.8.10.2          cmake/3.1.3(default) cmake/3.9.4

% module add bwpy; bwpy-environ -- cmake --version
cmake version 3.11.2      # bwpy cmake
Things to keep in mind when using bwpy-environ

- **bwpy-environ** starts a new shell
  - ENV is lost on exit from **bwpy-environ**
  - Parent variables need to be explicitly exported to be visible

- Mounting the image is expensive, best to do multiple things at once or stay in **bwpy-environ** rather than using many Python calls

- When used with **aprun**, use **-b** switch

```
$ bwpy-environ
$ mount | grep bwpy
/mnt/a/sw/xe_xk_cle5.2UP02_pe2.3.0/images/bwpy/bwpy-2.0.1.img
on /mnt/bwpy type squashfs (ro,nosuid,nodev,noatime)

#PBS
aprun -b -n1 -- bwpy-environ -- python --version
```
Building software against BWPY

- Use with `gcc/4.9.3 (bwpy/default)` or `gcc/5.3.0 (bwpy/2.0.1)`
- Export these variables, so these dirs come after `-I` and `-L`

```bash
$ module swap PrgEnv-cray PrgEnv-gnu
$ module swap gcc gcc/4.9.3
$ export CPATH="$CPATH:$BWPY_INCLUDE_PATH"
$ export LIBRARY_PATH="$LIBRARY_PATH:$BWPY_LIBRARY_PATH"
$ export LDFLAGS="$LDFLAGS -Wl,--rpath=$BWPY_LIBRARY_PATH"
```

- Do not use LD_LIBRARY_PATH to avoid potential incompatibility issues
- Use CMake from bwpy
- Software inside of BWPY has its own include paths, e.g.
  `/mnt/bwpy/single/usr/include/tensorflow/` for TensorFlow’s C++ interface
- Compilation **must** be done in a `bwpy-environ` shell!
Building scipy/1.2.0 against BWPY

module swap PrgEnv-cray PrgEnv-gnu
module load bwpy
git clone https://github.com/scipy/scipy.git scipy
cd scipy
git tag
git checkout v1.2.0

export CPATH="$CPATH:$BWPY_INCLUDE_PATH"
export LIBRARY_PATH="$LIBRARY_PATH:$BWPY_LIBRARY_PATH"
export LDFLAGS="$LDFLAGS -Wl,--rpath=$BWPY_LIBRARY_PATH"

bwpy-environ -- setup.py build
bwpy-environ -- setup.py install -user
bwpy-environ -- pip install --user pytest

cd $HOME
python
import pytest
import scipy
scipy.__version__
scipy.test()

# run these under bwpy-environ
Building a python package against BWPY

module swap PrgEnv-cray PrgEnv-gnu
module load fftw
module load cudatoolkit
module load bwpy
module load cray-hdf5

export CRAYPE_LINK_TYPE=dynamic
export CRAY_ADD_RPATH=yes
export CXX=CC
export CC=cc
pip freeze | grep protobuf
pip freeze | grep h5py

export CPATH="$CPATH:$BWPY_INCLUDE_PATH"
export LIBRARY_PATH="$LIBRARY_PATH:$BWPY_LIBRARY_PATH"
export LDFLAGS="$LDFLAGS -Wl,--rpath=$BWPY_LIBRARY_PATH"

mkdir build
cd build
bwpy-environ -- cmake ..
bwpy-environ -- make
Creating local python environment with help of Virtualenv

- BWPY (1.2.4) contains 262 python(3) packages
- Extra packages should be installed in a virtualenv to avoid version conflicts when installing in $HOME/.local
  - use --system-site-packages option to import the existing packages
  - Python in virtualenv is **frozen** to BWPY version active at **creation**
- Use pip to install extra packages
  - **do not** use --user option in virtualenv
  - use --force-reinstall to overwrite existing packages
  - use pip install mypackage==x.y.z to force specific version
  - https+git://git-repository-with-setup.py for git repositories
Virtualenv examples

$ mkdir myvirtualenv
$ cd myvirtualenv
$ virtualenv -p python2.7 --system-site-packages $PWD
$ source bin/activate
$ pip install myfavoritepackage
$ deactivate

$ export GEOS_DIR=/mnt/bwpy/single/usr/
$ pip install pyproj==1.9.3
$ pip install git+https://github.com/matplotlib/basemap

$ pip install --force-reinstall yt
Ok to run on login nodes, within reason

```
bw$ module load bwpy
bw$ bwpy-environ -- bash -ic jupyter-notebook
```

The Jupyter Notebook is running at:

```
http://10.0.0.147:8981/
```

```
laptop% ssh -L 8888:10.0.0.147:8981 bw.ncsa.illinois.edu
laptop% open http://127.0.0.1:8888
```

Notebook server is accessible Blue Waters wide

- use password to protect the notebook server
- `jupyter` outputs connection information to stdout on startup
- use `second` ssh connection to the login node to forward the local port
- `jupyter` auto-saves notebooks in case connection is lost (or use `screen`)

See [https://bluewaters.ncsa.illinois.edu/pythonnotebooks](https://bluewaters.ncsa.illinois.edu/pythonnotebooks)
Data exploration modules

- BWPY provides large number of modules for data exploration
  - numpy, scipy, sympy
  - h5py, netCDF, gdal, pandas
  - astropy, PostCactus
  - matplotlib, yt, plotly
- use `%matplotlib notebook` to show plots

See https://bluewaters.ncsa.illinois.edu/pythonnotebooks
**Python and MPI**

- BWPY includes **mpi4py** linked against Cray MPI stack
  - load as bwpy-mpi submodule
  - cannot be used on login nodes, even when using single rank
  - **only one** `MPI_Init()` **per** `aprun`, **implicit in** `import mpi4py.MPI`
  - **use** `aprun` **to** start **Python**
  - **use** `-d` **for** multi-threaded code or job bundling

```python
$ cat hello.py
from mpi4py import MPI
print ("Hello from rank ", MPI.MPI_COMM_WORLD.Get_rank())
```

```bash
$ qsub -I -l nodes=1:ppn=32:xe -l walltime=0:30:0 -q debug
% module load bwpy
% module load bwpy-mpi
% aprun -n4 -d8 -b -- bwpy-environ -- python ./hello.py
```
Running single-threaded jobs in python

- Do not start hundreds of single-threaded python scripts via aprun
  - wasteful since each aprun claims a full node
  - slow, each aprun takes ~1min to start and finish
  - hard on the system (we will contact you if you abuse this too much)

- Use mpi4py `MPICommExecutor`
  - Put your payload code in a function taking a single argument
  - Create a list of tasks
  - Pass the list to `MPICommExecutor`

- Benefits
  - Can run multiple tasks on a single node
  - New tasks start as soon as previous ones finished
  - Pure python code
Example of job bundling

```python
from mpi4py import MPI
from mpi4py.futures import MPICommExecutor

def fun(x):
    print("on %s print %g" % (MPI.COMM_WORLD.Get_rank(), x))

with MPICommExecutor(MPI.COMM_WORLD, root=0) as executor:
    jobs = range(100)
    if executor is not None:
        executor.map(fun, jobs)

aprun -n $NRANKS -d1 -b -- bwpy-environ -- python ./run.py

See further details in https://bluewaters.ncsa.illinois.edu/job-bundling#using_multiple_nodes_and_python
```
Blue Waters documentation

- https://bluewaters.ncsa.illinois.edu/python
- https://bluewaters.ncsa.illinois.edu/pythonnotebooks
- https://bluewaters.ncsa.illinois.edu/data-transfer-doc#gcli
- https://bluewaters.ncsa.illinois.edu/job-bundling#using_multiple_nodes_and_python
Questions?

This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.