Python on ARCHER

Nick Johnson

CSE Team, EPCC

Overview

- Motivation
- What is available & where
- Running on ARCHER:
  - on the login nodes
  - on the PP/serial nodes
  - on the compute nodes
- Adding your own modules
- “Gotchas”
- Scaling performance
- Where to get more help
- Questions

Motivation

- Why would you want to run an interpreted language like Python on a supercomputer like ARCHER?
  - You could do it all in C or Fortran
- Portability
- Ease of coding + testing
- Quick data analysis
- Pre and Post processing of data sets, input files

Disclaimer

- I'll be using an unreleased module in this talk to illustrate some points which I don't recommend
- It's `python/2.7.6-experimental`
- Eventually, these will become part of the python/2.7.6 module so you will have access to them.

Running Python on ARCHER

- There are three places to run Python on ARCHER:
  - Login nodes
  - PP/serial nodes
  - Compute nodes

- There are, just to add to the confusion, three version of Python available on ARCHER
  - 2.6.8
  - 2.7.6 (default)
  - 3.3.3

- 2.7.6 (the default) is automatically loaded as a module when you login.

Login nodes

- You can run any of the versions of Python but we recommend the default and only using the login node if it's a tiny job.

```
eslogin006:> python helloworld.py
Hello, World
```

- Acceptable

```
eslogin006:> python monster_data_analysis.py
Res1 = 10
Res2 = 11
```

- Not acceptable – you will likely get an email about it from the helpdesk!

PP nodes/serial nodes

- The best place to run your large analysis codes
- Surely you just do as you would do for any other PP job?

```
njohnso1@eslogin006:~/work> cat serialjob.pbs
#!/bin/bash –login
#PBS -l select=serial=true:ncpus=1
#PBS -l walltime=00:01:00#PBS -A z01-cse
# Make sure any symbolic links are resolved to absolute path
export PBS_O_WORKDIR=$(readlink -f $PBS_O_WORKDIR)
# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR
python helloworld.py
```

What just happened?

- The **system** version of python (2.6.8) has numpy installed
- The **default** version does not (but will soon)
- We didn't load any modules in our job script so we used the **system** python (`/usr/bin/python`)

- Always check by inserting `which python` into your job script to be sure.
- The **system** python *might* change version with a CNL upgrade, numpy might cease to be available.
- I recommend not using it.
- Explicitly load the **default** module in your script

Interactive jobs

- If you want to run interactive python but are worried about saturating a login node, use an interactive job on the PP nodes:

```
qsub -IVl select=serial=true:ncpus=1,walltime=1:0:0 -A
budget
```

- The same caveats about versions apply

Compute nodes

- Running python on the compute nodes is entirely possible
- But, if you only have a single process code, it might be a waste of resources…
- Python is inherently single process and current threading doesn't help much other than for process control.
  - StackOverflow has many, many articles on parallelism in Python for the curious.
- Parallelism is possible on the compute (but not yet released to users).
- The trick is to use mpi4py which allows python processes to communicate using MPI.

What's happening

- aprun starts N seperate processes, each of which is running an instance of the python interpreter
- They communicate using MPI calls which are passed to a C library which pass them down the stack, as with any other MPI code.
- There is no magic involved – you can use other modules with each of these processes, numpy for example…

Compiling & installing

- You are free to compile and install your own modules
- If it's a pure python module (no C code) there will be no problem.
- If compilation is required, you should use PrgEnv-gnu
- I recommend using the setup.py that comes with the module:

```
python setup.py build options
python setup.py install –home=<dir>
```

- If you want to use your module on the compute nodes, you MUST install to /work
- Don't forget to set PYTHONPATH in your jobscript **AND** load the **default** module if you built against that.
- Don't forget that the CSE team are here to help you, for free!
  - If you are struggling, get in touch via the helpdesk.

Gotchas

- Things to watch out for when using Python on ARCHER:
  - The **wrong** Python – jobs submitted directly to the back-end (PP or compute nodes) via qsub will not have the default python loaded or have the correct `PYTHONPATH` set.
  - `#!/usr/bin/python`
    - This will load the system python and potentially cause pain
    - Either change to `#!/usr/bin/env python`
    - Or just comment it out and run as a script
  - Cannot find modules
    - If you have compiled and installed yourself, make sure you have correctly set `PYTHONPATH`
    - As best practice, echo it at the start of your jobscript along with the output of `which python`

Scaling

- If your python code loads a lot of dynamic modules, ie shared libraries that you call into, you will notice horrible scaling performance.
  - Really horrible
- The problem is that every process (say 1056) is trying to load the same file from the file-system and this bottlenecks, badly.
- There are some solutions being trialled by the CSE team, but they are not ready yet.
  - If this is a pressing need for you, please get in touch

Scaling times