HPC Python Programming Hands-On Exercises

Ramses van Zon

August 22, 2019

These are the instructions for the hands-on for the Python for HPC sesion of the Petascale Computing Institute 2019. There are 4 hands-on exercises, each of which should take about 10 minutes. The first two exercises have optional parts for students that finish the main part of that exercise early.

Setting Up On Blue Waters

To get set up for the session using the Blue Waters supercomputer at NCSA, perform the following steps.

1. Login to Blue Waters:

```
$ ssh -Y USERNAME@bwbay.ncsa.illinois.edu
```

2. Copy code for this session:

```
$ cp -r /u/training/instr041/hpcpython/ $HOME
```

3. Request resources on a compute node (if you want accurate timing results):

```
$ qsub -I -X -lnodes=1:ppn=16,walltime=2:00:00
```

4. Setup the environment:

```
$ source /u/training/instr041/hpcpyenv/activate
```

Setting Up On Cori

To get set up for the session using the Cori supercomputer at NERSC, perform the following steps.

1. Login to Blue Waters:

```
$ ssh -Y USERNAME@cori.nersc.gov
```

2. Copy code for this session:

```
$ cp -r /global/homes/r/rzon/hpcpython/ $HOME
```

3. Request resources on a compute node (if you want accurate timing results):

```
$ salloc -N 1 -n 16 -C haswell -q interactive -t 02:00:00
```

4. Setup the environment:

```
$ source /global/homes/r/rzon/hpcpyenv/activate
```

Setting Up On Your Own Computer

Alternatively, you can use the code of the exercises on your own computer by downloading https://support.scinet.utoronto.ca/~rzon/petapy.zip .

You will need to have Python 3 installed with the following packages:

```
numpy     scipy          numexpr          matplotlib
psutil    line_profiler  memory_profiler  mpi4py
cython    numba
```

The examples are aimed for a Linux environment with g++/gfortran/make, but if you do not have that, you can still use the Python code in the zip file if the above packages are installed.

The code also comes with a setup script, so if you're on Linux, open a terminal, go to the code directory and type the command 'source setup'. This will try to setup a virtual environment with the above packages and will try to compile the `rarray` and `pgplot` libraries.

1. Hands-on 1: Profiling

Profile the auc_serial.py code

- Consider the Python code for computing the area under the curve in `auc_serial.py`.
- Put the code in a wrapper function. Make sure it still works!
- Add @profile to the main function.
- Run this through the line profiler and see what line(s) cause the most cpu usage.

Optional (if you finish early): Profile the diff2d.py code

- Reduce the resolution and runtime in `diff2dparams.py`, i.e., increase `dx` to 0.5, and decrease `runtime` to 2.0.
- In the same file, ensure that `graphics=False`.
- Add @profile to the main function in `diff2d.py`
- Run this through the line profiler and see what line(s) cause the most cpu usage.

2. Hands-on 2: Vectorizing

Vectorize the auc_serial.py code

- Copy the Python code in `auc_serial.py` to a new file `auc_vector.py`
- Remove any `@profile` decorators.
- Reexpress the code using numpy arrays.
- Make sure you are using vectorized operations.
- Measure the speed-up (if any) with `/usr/bin/time`.

Optional (if you finish early): Vectorize the slow numpy code

If you are done with the auc example, try this:

- Copy the file `diff2d_slow_numpy.py` to `diff2d_numpy.py`.
- Replace the indexed loops with whole-array vector operations.
- Measure the speed-up (if any) with `/usr/bin/time`.

3. Hands-on 3: Parallelize Area-under-the-curve

- Use the `numexpr` package to parallelize the `auc_vector.py` code.
- Measure the speed-up using `/usr/bin/time` with up to 16 threads.

4. Hands-on 4: MPI area under the curve

The exercise is for you to parallelize `auc_vector.py` with the `mpi4py` package:

- Take your numpy-vectorized code in `auc_vector.py`.
- Divide the computation over MPI tasks.
- Measure speed-up for up to 8 processes.