MPI4PY

- The Problem
- Step-by-step guide
  - Make a directory for all your virtualenv projects (optional)
  - Decide on your compiler/MPI and know where you are
  - Run virtualenv and install mpi4py
  - Installing for another combination
  - Using mpi4py
  - Launching MPI jobs
  - Notes for Deneb
    - MPI4PY with Intel Infiniband/Omnipath
    - Different architectures and the GPU nodes

This article explains how to install mpi4py in your home directory on the cluster in such a way that it is possible to use different clusters and/or compilers and MPI libraries.

The Problem

mpi4py needs to be built for a specific combination of the following:

- Compiler
- MPI flavour
- CPU type
- Interconnect
- Python version

This means that if installed with pip install --user mpi4py it will be tied to the exact combination of the aforementioned requirements used at install time.

In SCITAS clusters there is an environment variable set that captures the CPU type and the Interconnect: $SYS_TYPE.

The solution to this is to use the Python virtualenv package which allows multiple python environments to co-exist.

Step-by-step guide

Make a directory for all your virtualenv projects (optional)

```
$ cd
$ mkdir virtualenv
```

Decide on your compiler/MPI and know where you are

For each combination we need to create a directory and so we choose the following naming convention:

$SYS_TYPE_compiler_MPI

e.g.

```
$ cd virtualenv
$ mkdir ${SYS_TYPE}_gcc_mvapich
$ ls
x86_E5v4_Mellanox_gcc_mvapich
```

Note that for Intel there is only one MPI available so ${SYS_TYPE}_intel will suffice as a name.

${SYS_TYPE} identifies the hardware type and looks something like x86_E5v4_Mellanox

Optionally, if one intends to use multiple Python versions you might want to include the version of Python, for example, you can append: _py3.

Run virtualenv and install mpi4py

First check that the correct modules have been loaded!
$ module load gcc mvapich2 python
$ module list
CurrentlyLoaded Modules:
1) gcc/7.4.0 2) mvapich2/2.3.1 3) python/3.7.3

Now create the virtualenv by pointing to use the appropriate directory and python version (2 or 3)

$ virtualenv -p python3 --system-site-packages virtualenv/$(SYS_TYPE)_gcc_mvapich
Running virtualenv with interpreter /ssoft/spack/humagne/v1/opt/spack/linux-rhel7-x86_E5v4_Mellanox/gcc-7.4.0/python-3.7.3-5lm3virkrg4nq4tjhx76dgqy7zbt4kfam/bin/python3
Using base prefix '/ssoft/spack/humagne/v1/opt/spack/linux-rhel7-x86_E5v4_Mellanox/gcc-7.4.0/python-3.7.3-5lm3virkrg4nq4tjhx76dgqy7zbt4kfam'
New python executable in /home/user/virtualenv/x86_E5v4_Mellanox_gcc_mvapich/bin/python3
Also creating executable in /home/user/virtualenv/x86_E5v4_Mellanox_gcc_mvapich/bin/python
Installing setuptools, pip, wheel...
done

And change to the newly created virtual environment

$ source virtualenv/$(SYS_TYPE)_gcc_mvapich/bin/activate
(x86_E5v4_Mellanox_gcc_mvapich) [user@cluster ]$

Note that the prompt is changed as a reminder that the virtualenv is active.

Now we can install mpi4py using the "--no-cache-dir" option to make sure that it always get rebuilt correctly

(x86_E5v4_Mellanox_gcc_mvapich) [user@cluster]$ pip install --no-cache-dir mpi4py
Collecting mpi4py
  Downloading mpi4py-3.0.3.tar.gz (1.4 MB)
  (...)
Successfully built mpi4py
Installing collected packages: mpi4py
Successfully installed mpi4py-3.0.3

To leave the virtual environment simply type deactivate.

(x86_E5v4_Mellanox_gcc_mvapich) [user@cluster -]$ deactivate
[user@cluster -]$ 

Installing for another combination

We can repeat the above process for as many permutations as we need:
Using mpi4py

When you want to use mpi4py you now need to load the appropriate virtual environment as well as loading the corresponding modules:

```
$ module load intel intel-mpi python
$ source virtualenv/${SYS_TYPE}_intel/bin/activate
{x86_E5v4_Mellanox_intel} [user@cluster]$ pip install --no-cache-dir mpi4py
Collecting mpi4py
Installing collected packages: mpi4py
Successfully installed mpi4py-3.0.3
```

Note how virtualenv knows which python is associated with the environment so simply typing `python` is sufficient.

The same applies for batch scripts - just source the virtualenv after loading your modules.

Launching MPI jobs

As with traditional MPI jobs you need to use `srun` to correctly launch the jobs:

```
srun python mympicode.py
```

Failure to use `srun` will result in only one rank being launched.

Notes for Deneb

**MPI4PY with Intel Infiniband/Omnipath**

By default MPI4PY is not fully compatible with the Interconnect on the Deneb cluster due to the use of some MPI 3.0 calls which are not supported. The usual symptom is that communications between ranks will block or fail.

Please see the following page for the details: https://software.intel.com/en-us/articles/python-mpi4py-on-intel-true-scale-and-omni-path-clusters

The solution is to set the following variable just after importing MPI4PY:

```
```
A more explicit example is:

```python
import mpi4py
mpi4py.rc.recv_mprobe = False
from mpi4py import MPI

comm = MPI.COMM_WORLD
..
```

When using Intel MPI (module load intel-mpi) it is also possible to work around the issue by changing the fabric protocol via "export I_MPI_FABRICS=shm:ofa". This is not possible for MVAPICH2 nor OpenMPI.

**Different architectures and the GPU nodes**

Deneb is a heterogeneous cluster with the following $SYS_TYPE:

- x86_E5v2_IntelIB
- x86_E5v3_IntelIB
- x86_E5v2_Mellanox_GPU

The first two are cross compatible for mpi4py but if using the GPU nodes you should change to the appropriate $SYS_TYPE before configuring mpi4py.

```
$ slmodules --s x86_E5v2_Mellanox_GPU -v
[INFO] S+L release: stable
[INFO] S+L systype: x86_E5v2_Mellanox_GPU
[INFO] S+L engaged!
$ echo $SYS_TYPE
x86_E5v2_Mellanox_GPU
```