

Learning Objectives

The learning objectives in the tutorial are

- Construct and *Pickle Python* objects
- Describe and measure the performance impact "Pickling" has on MPI communication
- Apply the differing MPI-routines based on the first letter correctly

Tools

- mpi4py
- Anaconda

Contents

Working with Anaconda 1: Tutorial (10 min)	1
Sending a Custom Object 2: Tutorial (10 min)	2
Benchmarking the "Pickling" Impact 3: Tutorial (10 min)	3

Working with Anaconda 1: Tutorial (10 min)

Anaconda can be used to create your own, dedicated python environments. *Anaconda* is already installed on the *SCC* and can be immediately used by you.

Steps

1. Load the `py-mpi4py` module
2. Install `uv` via `curl -LsSf https://astral.sh/uv/install.sh | sh`
3. Run `uv add 'numpy'` to add Numpy
4. If you want to run your job use for instance `uv run mpirun -n 2 python your-script.py`
5. Use this line inside your batch script
6. Submit a simple python script with *sbatch*, which only imports *numpy* and *pandas*. You can print a *Done!* afterwards to have some output on your *stdout*

Hints

- In your run/batch script, you also need to load the *py-mpi4py* module
- You can load a module with:

```
1 module load py-mpi4py
```

- to be able to install python modules with *uv* use:

```
1 uv add
```

- You can also edit the `pyproject.toml` file by hand

Sending a Custom Object 2: Tutorial (10 min)

In the lecture we have seen two different groups of communication based on the performed "pickling", remember, it depended on the capitalized first character (send vs. Send). In this tutorial we want to write a short ping-pong benchmark to measure the effects "pickling" have on the overall communication performance.

Steps

1. Load the module *py-mpi4py* `$ module load py-mpi4py`
2. Create a "rather complicated" class which has an internal state, e.g. some variables are set to a certain value
3. Define a function which modifies the state (variable) of an object of that class
4. Define a function to print the state of an object of that class
5. The process with *rank == 0* should create one instance
6. Send the object to process with *rank == 1*
7. Print the state of the object on the process *rank == 1*.
8. Change the state of the object in process *rank == 1* and send it to process *rank == 0*
9. Print the state of the object in process *rank == 0*

Hints

- You can build your custom class with something like:

```
1 class MyClass:
2
3     def __init__(self):
4         self.my_str = 'Initialized'
5
6     def print_state(self):
7         print(self.my_str)
8
9     def change_state(self, new_str):
10        self.my_str = new_str
```

- You can initialize your MPI with:

```
1 from mpi4py import MPI
2
3 comm = MPI.COMM_WORLD
4 size = comm.Get_size()
5 rank = comm.Get_rank()
```

- You can check the rank of a process with

```
1 if rank == 0:
2     # do stuff
```

- You can send and receive messages with:

```
1 if rank == 0:
2     data = {'a': 1, 'b': 2, 'c': 'test string'}
3     comm.send(data, dest=1, tag=11)
4 elif rank == 1:
5     data = comm.recv(source=0, tag=11)
6     print(data)
```

Benchmarking the "Pickling" Impact 3: Tutorial (10 min)

Steps

1. Initialize a *numpy* array with 1000000 elements
2. Send this array from process 0 to process 1 using the lower case mpi function
3. Measure the time this communication needs. Feel free to average over multiple runs
4. Repeat the previous steps with the upper case MPI functions
5. Describe and explain the difference

Hints

- You can initialize a numpy array with

```
1 data = numpy.arange(1000000, dtype=numpy.float)
```

- You can measure the time a code snippet needs with:

```
1 import time
2
3 start = time.time()
4 print("hello")
5 end = time.time()
6 print(end - start)
```

Further Reading

- <https://docs.python.org/3/tutorial/classes.html>
- <https://mpi4py.readthedocs.io/en/stable/index.html>