
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

Sending a Custom Object 1: Tutorial (30 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 *openmpi* `$ module load openmpi`
2. Install *mpi4py* into your *conda* environment
   `$ pip install mpi4py`
3. Create a "rather complicated" class which has an internal state, e.g. some variables are set to a certain value
4. Define a function which modifies the state (variable) of an object of that class
5. Define a function to print the state of an object of that class
6. The process with *rank == 0* should create one instance
7. Send the object to process with *rank == 1*
8. Print the state of the object on the process *rank == 1*.

9. Change the state of the object in process *rank == 1* and send it to process *rank == 0*
10. Print the state of the object in process *rank == 0*

Hints

- If you get an error installing *mpi4py* with *pip* you are likely having a wrong *Python* version. Please install *Python 3.7* using:
  `$ conda install python=3.7`
- The difference between *conda istall* and *pip install* is that *conda* pulls prebuilt packages, whereas *pip* is building a package locally
- You can built your custom class with something like:

```
class MyClass:

  def __init__(self):
    self.my_str = 'Initialized'

  def print_state(self):
    print(self.my_str)

  def change_state(self, new_str):
    self.my_str = new_str
```

- You ca initialize your MPI with:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
```

- You can check the rank of a process with

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

- You can send and receive messages with:

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

Benchmarking the "Pickling" Impact 2: Tutorial (30 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

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

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

```
import time

start = time.time()
print("hello")
end = time.time()
print(end - start)
```

Further Reading

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