
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

Task 1: Working with Anaconda (10 min)

Anaconda can be used to create your own, dedicated python environments. *Anaconda* is already installed on the *SCC* cluster (as miniforge) and can be immediately used by you. See our HPC Documentation (click for link) for more.

Steps

1. Load the miniforge/conda module: `module load miniforge3`
2. Create an environment in a specific location: `conda create --prefix ./mpi4py`
3. Activate it: `source activate ./mpy4pi` NOTE: Do NOT use the usual `conda activate myenvironment`.
4. `conda install mpi4py numpy`
5. Make sure you are using the right mpirun/python with `which mpirun` and `which python3`.

Task 2: Sending a Custom Object (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*
10. How does MPI ensure that our steps take place in the correct order? (hint: why is it improtant to pair send's and recv's?) Compare and contrast with OpenMP.

Hints

- 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 can 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:

```python
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)
```

Task 3: Benchmarking the "Pickling" Impact (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 in another script with the upper case MPI functions. Pay attention to the syntax differences!
5. (If you want to do both in a single script, you'll need to sync the ranks with a comm.barrier() after the first transfer, otherwise you will get inconsistent results)
6. Describe and explain the difference

Hints

- You can initialize a numpy array with

```python
data = numpy.arange(1000000, dtype=numpy.float64)
```

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

```python
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