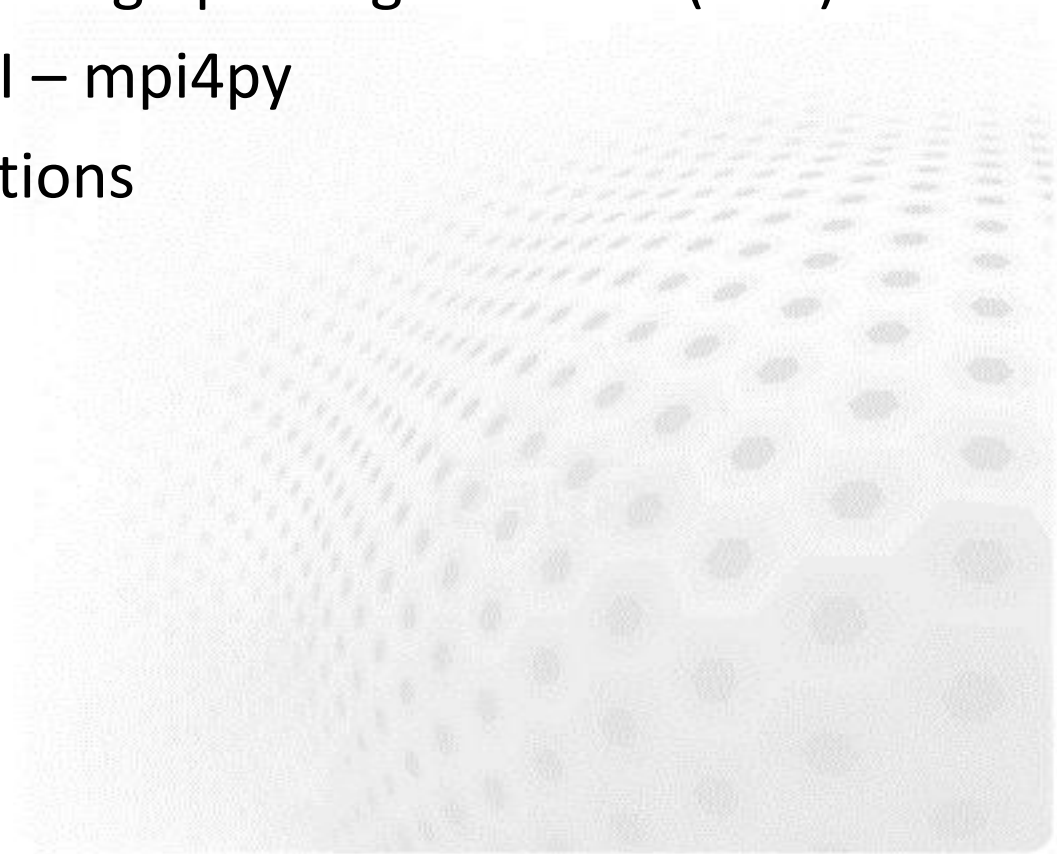


PARALLEL PROGRAMMING WITH PYTHON USING MPI4PY



Outline

- Brief introduction to message passing interface (MPI)
- Python interface to MPI – mpi4py
- Performance considerations

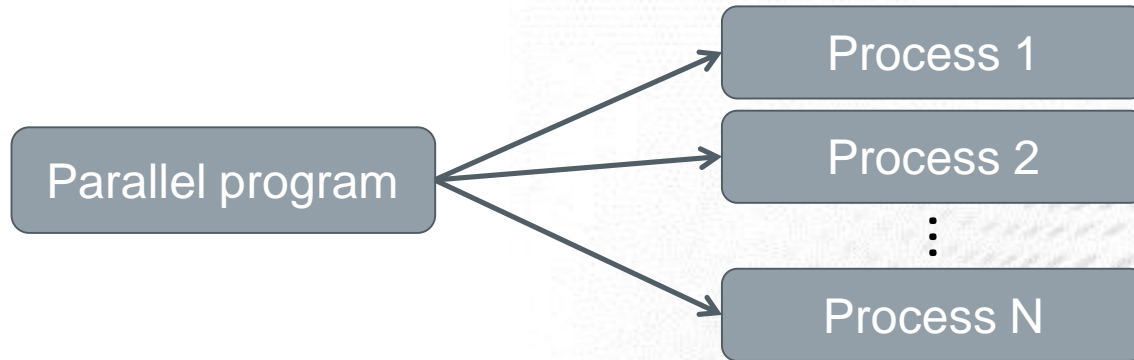


Message passing interface

- MPI is an application programming interface (API) for communication between separate processes
- The most widely used approach for distributed parallel computing
- MPI programs are portable and scalable
 - the same program can run on different types of computers, from PC's to supercomputers
- MPI is flexible and comprehensive
 - large (over 120 procedures)
 - concise (often only 6 procedures are needed)
- MPI standard defines C and Fortran interfaces
- **mpi4py** provides (an unofficial) Python interface

Execution model in MPI

- Parallel program is launched as set of **independent, identical processes**



- All the processes contain the same program code and instructions
- Processes can reside in different nodes or even in different computers
- The way to launch parallel program is implementation dependent
 - mpirun, mpiexec, aprun, poe, ...
- When using Python, one launches N Python interpreters
 - mpirun -np 32 python parallel_script.py

MPI Concepts

- rank: id number given to process
 - it is possible to query for rank
 - processes can perform different tasks based on their rank

```
mpi.py
```

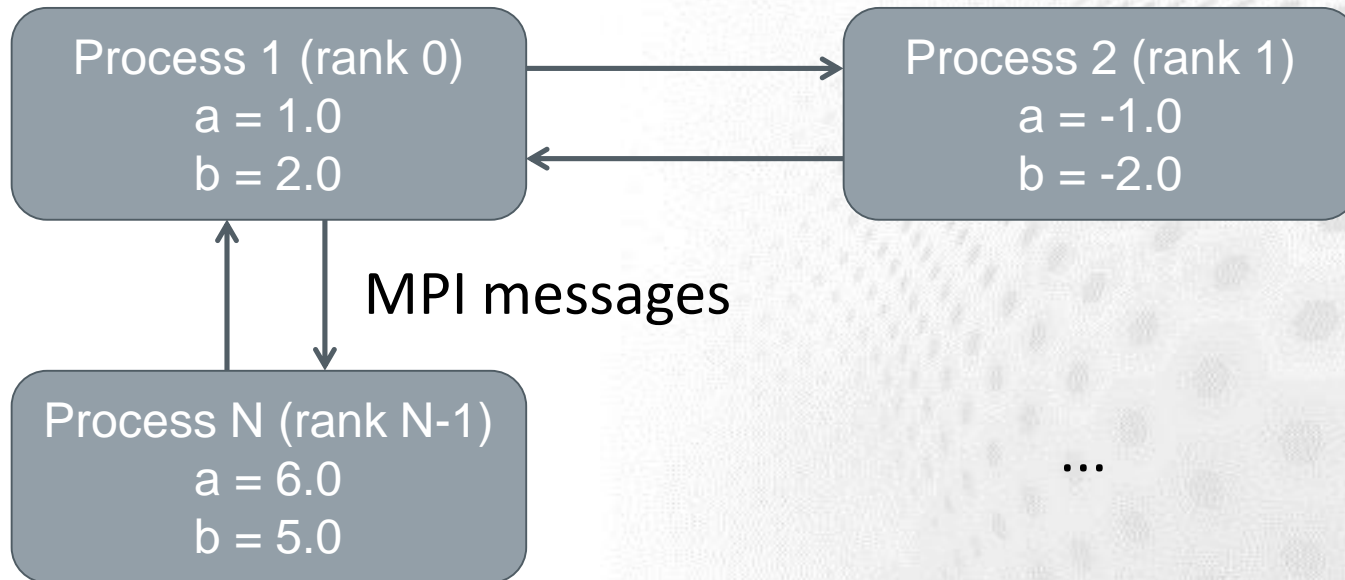
```
if (rank == 0):  
    # do something  
elif (rank == 1):  
    # do something else  
else:  
    # all other processes do something different
```

MPI Concepts

- Communicator: group containing process
 - in mpi4py the basic object whose methods are called
 - **MPI_COMM_WORLD** contains all the process (MPI.COMM_WORLD in mpi4py)

Data model

- All variables and data structures are local to the process
- Processes can exchange data by sending and receiving messages



Using mpi4py

- Basic methods of communicator object
 - Get_size()
Number of processes in communicator
 - Get_rank()
rank of this process

```
mpi.py
from mpi4py import MPI

comm = MPI.COMM_WORLD # communicator object containing all processes
size = comm.Get_size()
rank = comm.Get_rank()

print "I am rank %d in group of %d processes" % (rank, size)
```


Sending and receiving data

➤ Sending and receiving a dictionary

```
mpi.py
from mpi4py import MPI

comm = MPI.COMM_WORLD # communicator object containing all processes
rank = comm.Get_rank()

if rank == 0:
    data = {'a': 7, 'b': 3.14}
    comm.send(data, dest=1, tag=11)
elif rank == 1:
    data = comm.recv(source=0, tag=11)
```

Sending and receiving data

- Arbitrary Python objects can be communicated with the **send** and **receive** methods of communicator
- **send(data, dest, tag)**
 - **data** Python object to send
 - **dest** destination rank
 - **tag** id given to the message
- **recv(source, tag)**
 - **source** source rank
 - **tag** id given to the message
 - data is provided as return value
- Destination and source ranks as well as tags have to match

Communicating NumPy arrays

- Arbitrary Python objects are converted to byte streams when sending
- Byte stream is converted back to Python object when receiving
- Conversions give overhead to communication
- (Contiguous) NumPy arrays can be communicated with very little overhead with upper case methods:
- **Send(data, dest, tag)**
- **Recv(data, source, tag)**
 - Note the difference in receiving: the data array has to exist in the time of call

Communicating NumPy arrays

➤ Sending and receiving a NumPy array

```
mpi.py
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

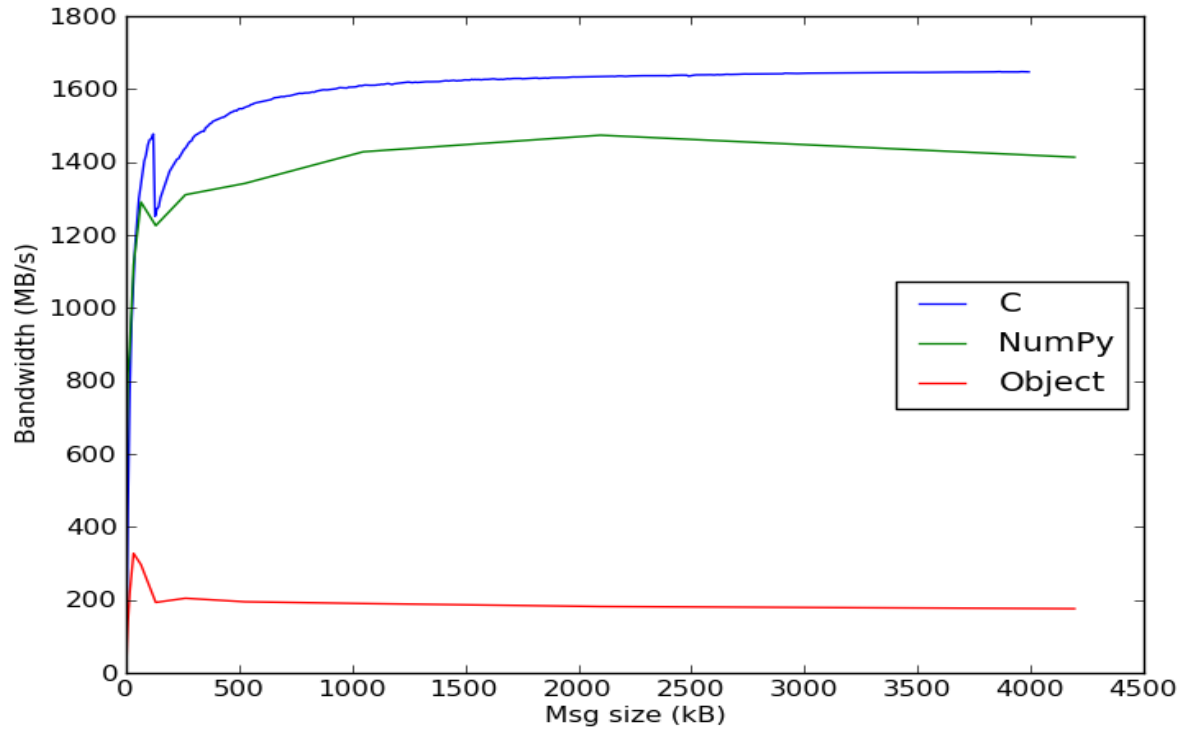
if rank == 0:
    data = numpy.arange(100, dtype=numpy.float)
    comm.Send(data, dest=1, tag=13)
elif rank == 1:
    data = numpy.empty(100, dtype=numpy.float)
    comm.Recv(data, source=0, tag=13)
```

➤ Note the difference between upper/lower case!

- send/recv: general Python objects, slow
- Send/Recv: continuous arrays, fast

mpi4py performance

➤ Ping-pong test



Summary

- mpi4py provides Python interface to MPI
- MPI calls via communicator object
- Possible to communicate arbitrary Python objects
- NumPy arrays can be communicated with nearly same speed as from C/Fortran