DTU Compute
Department of Applied Mathematics and Computer Science

DTU

High-Performance Computing

02616

Large-scale Modelling

recap week 4

`mpi4py`

MPI

- How `Recv/Send` are used
  - `Recv` is a *maximum* buffer size!
- The difference between `send/Send` (prefer `Send`)
  - `Send/Recv` is for pre-allocated numpy buffers
  - `send/recv` is for Python objects (slow and lots of overhead)
- How to use the buffer argument to control amount of data sent/received
  - `comm.Send(buf, …) # auto dtype + count`
  - `comm.Send([buf, 1], …) # auto dtype`
  - `comm.Send([buf, MPI.INT], …) # auto count`
  - `comm.Send([buf, 1, MPI.INT], …) # explicit`
- How to use non-blocking communications

```
req = comm.Isend(buf, …)
# do something else than writing to buf
req.Wait() # buf is now sent!
```

  - Remember to *always* post a `Wait/Test` for all requests

MPI

- How to use `Status` for getting data

```
comm.Recv(…, status=status)
count = status.Get_count(MPI.DOUBLE)
tag = status.tag
source = status.source
```

- How to use *collectives*

```
#rank=0: [0, 1] → [0, 1]
#rank=1:        → [0, 1]
comm.Bcast(buf, root=0)
#rank=0: [0, 1] → [0]
#rank=1:        → [1]
comm.Scatter(buf_send, buf_recv, root=0)
#rank=0: [0] → [0, 1]
#rank=1: [1] →
comm.Gather(buf_send, buf_recv, root=0)
#rank=0: [0, 4] → [0, 3]
#rank=1: [1, 3] →
comm.Reduce(buf_send, buf_recv, op=MPI.MIN,
root=0)
```