Point-to-Point Communication with MPI

Part I: Blocking Communication

Be Boulder.

University of Colorado **Boulder**

Outline

- Overview
- HDF5 File Creation/Reading
- Attributes
- Groups
- Subgroups

NOTE: HDF5 is VERY cumbersome to work with in Fortran and C++. These slides cover Python semantics. Examples in other languages are provided in the repository.

Useful HDF5 References

- General HDF5 (C++/Fortran):

  https://support.hdfgroup.org/HDF5/doc/index.html

- h5py (Python):

  http://www.h5py.org/

- h5 (R):

  https://cran.r-project.org/web/packages/hdf5r/index.html

Python Note

- Load the anaconda python 3 module

HDF5 Overview: Why HDF5

- Hierarchical Data Format (Your file is itself a mini file system)
- Standarized data structure
- Data + Metadata
- Portable
- Parallel (underpins parallel I/O layer in many HPC applications)
- Relatively easy to use

Getting Started with h5py

- First, make sure you can import the h5py and numpy modules

```
import h5py
import numpy as np
```

h5py: File Creation

- First, create a file object:

filename = 'test.hdf5'
f = h5py.File(filename, "w")   w = 'write'

- See HDF5/examples directory: create_hdf5.{f90,cpp,R,py}

- File objects and their methods provide a high-level interface for interacting with a file (open, close, flush, etc.)

- Data can be added to the file by creating HDF5 datasets associated with the file.

Research Computing
UNIVERSITY OF COLORADO **BOULDER**

Be Boulder.

h5py: Datasets

- Next, add a dataset to the file:

```
dname1="Integers"
ndata1= (100,)  note the comma!
dset1 = f.create_dataset(dname1,  ndata1, dtype='int32')
```

- Specify a name
- Specify dimensions (ndata1)
- ndata1 must be a tuple (you need the comma; otherwise it's cast as an 'int' and isn't iterable).
- Use numpy datatypes

h5py: Datasets

- Populate the dataset (datasets work like NumPy arrays):

```
dset1[:] = np.arange(1,101,dtype='int32')
```

- Datasets can be multidimensional

```
dname2='Reals'
ndata2=(2,2)
dset2 = f.create_dataset(dname2,ndata2,dtype='float64')
dset2[0,:] = np.array( [2.1, 3.0 ], dtype='float64')
dset2[1,:] = np.array( [55.0, -73.01 ], dtype='float64')
```

Attributes

- Small, named pieces of data directly attached to group and dataset objects

- Basically a small dictionary with scalar or NumPy array values

```
dset1.attrs['month']=7
dset1.attrs['year']=2017
```

- Once we're finished describing our data, close the file

```
f.close()        close the file
```

Be Boulder.

h5py: File Inquiry

- We can examine a file's contents at the command line using h5dump (a tool that is part of HDF5)

h5dump –n test.hdf5	display table of contents

h5dump –B test.hdf5	display values

h5dump -h	display additional options

H5py: Reading Data

- When reading a file, we can access our datasets by treating the file object as a dictionary (read_hdf5.{f90,cpp,R,py}):

```
import h5py
import numpy as np
f = h5py.File('test.hdf5', 'r')    r = "read"
integers = f['Integers']
reals = f['Reals']
print(integers[:])
f.close()
```

h5py: Going Deeper

- HDF5 files organized around:
  - Groups:
    - folder-like containers that hold datasets and other groups
    - Think "directories"
    - Subgroups → "subdirectories"
  - Datasets:
    - array-like collections of data

- General philosophy in Python:
  - Datasets act like NumPy arrays
  - Groups act like dictionaries

h5py: Going Deeper

- The file object:
  - Represents the "root" group of the file:
  - Analogous to "/" on a POSIX file system
  - Each group has a name attribute

```
import h5py
f = h5py.File('test.hdf5', 'r')
print(f.name)
f.close()
```

h5py: Going Deeper

- Creating groups (create_groups.{f90,cpp,R,py}):
  - Groups can be created within the "root" group of the file:

```
import h5py
f = h5py.File('new.hdf5', 'w')
ff1 = f.create_group("folder1")
ff2 = f.create_group("folder2")
f.close()
```

```
h5dump –n new.hdf5      display table of contents
```

h5py: Going Deeper

- We can open a file and add data to existing groups (modify_groups.{f90,R,cpp,py}):

```
import h5py
f = h5py.File('new.hdf5', 'r+')        r+ = "read/write"
ff1 = f["folder1"]
npts=10
ndata=(npts,)
dname='data_range_1'
dset1=ff1.create_dataset(dname,ndata,dtype='int32')
dset1[:]=np.arange(1,npts+1,dtype='int32')
f.close()
```

```
h5dump –n new.hdf5     display table of contents
```

Research Computing
UNIVERSITY OF COLORADO BOULDER

Be Boulder.

h5py: Subgroups

- We can create groups within groups (and add data; subgroups.{f90,cpp,R,py}):

```
import h5py
f = h5py.File('new.hdf5', 'r+')          r+ = "read/write"
ff1 = f["folder1"]
fsub = ff1.create_group("subfolder1")
ndata=(10,)
dname2='data_set_2'
dset2=fsub.create_dataset(dname,ndata,dtype='int32')
dset2[:]=np.arange(21,31,dtype='int32')
f.close()
```

```
h5dump –n new.hdf5      display table of contents
```

h5py: Subgroups

- (Python) We can create a full path of subgroups at once:

```
import h5py
f = h5py.File('new.hdf5', 'r+')        r+ = "read/write"
ff2 = f["folder2"]
sfstring='subfolder2/subfolder3/subfolder4'
fsub2 = ff2.create_group(sfstring)
f.close()
```

```
h5dump –n new.hdf5     display table of contents
```