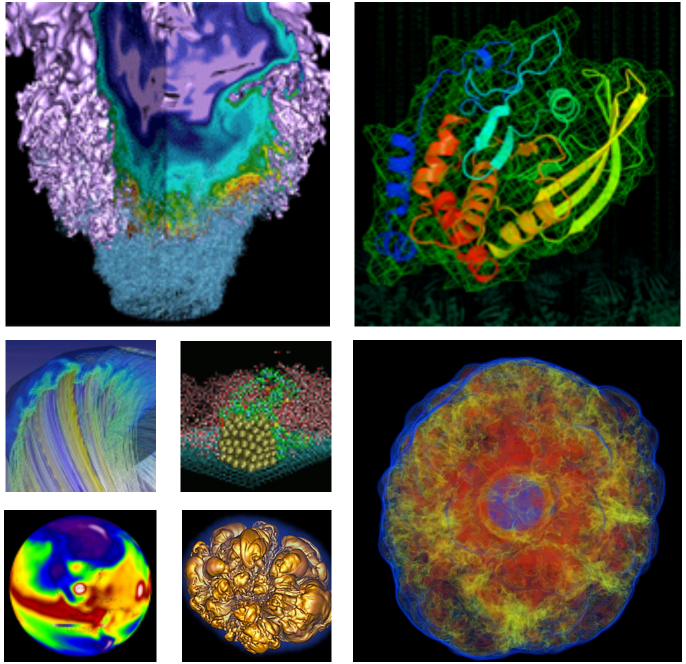
Productivity and High Performance, Can we have both?

An Exploration of Parallel-H5py from I/O Perspective



Jialin Liu

Data Analytics & Service Group

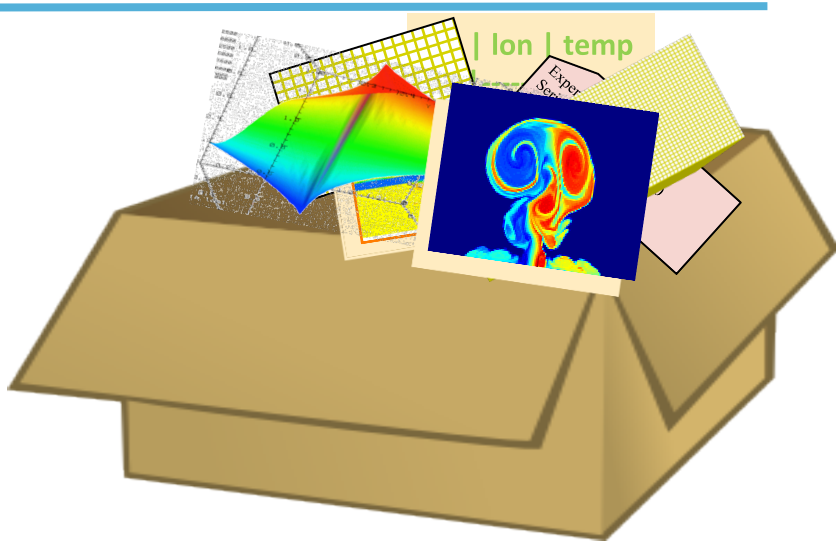
May 26, 2017

Outlines

- HDF5 and H5py
- Productivity
- H5py Internal
- Performance
- Case Studies
  - Warp
  - H5Boss

HDF5

- HDF5 are among the top 5 libraries at NERSC, 2015
  - 750+ unique users @NERSC, million of users worldwide
- 1987, NCSA&UIUC. NASA send HDF-EOS to 2.4 millions end users
- Hierarchical data organization
- Parallel I/O



Quincey Koziol

HDF5

HDF5 Datatype

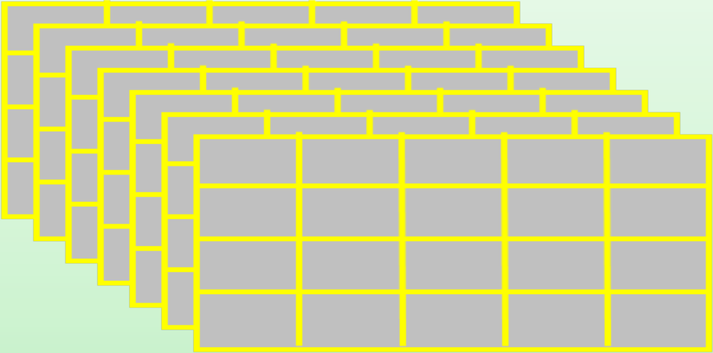
Integer: 32-bit, LE

HDF5 Dataspace

Rank	Dimensions

2	Dim[0] = 4
	Dim[1] = 5

Specifications for single data element and array dimensions



Multi-dimensional array of identically typed data elements

H5py

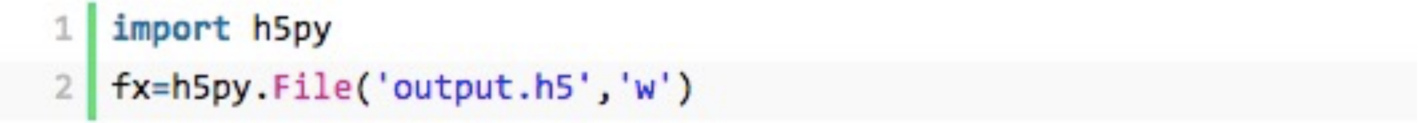
The h5py package is a Pythonic interface to the HDF5 binary data format.

- H5py provides easy-to-use high level interface, which allows you to store huge amounts of numerical data,
- Easily manipulate that data from NumPy.
- H5py uses straightforward NumPy and Python metaphors, like dictionary and NumPy array syntax.

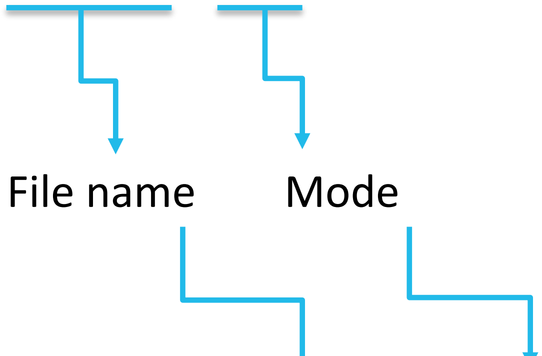
H5py: *a Productive HDF5 Interface*

Similar & Simpler Interface

- **Serial H5py**



```
import h5py
fx=h5py.File('output.h5','w')
```



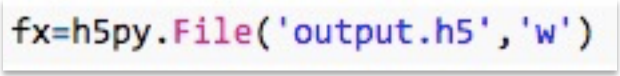
File name Mode

file_id = H5Fcreate(“output.h5”, H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);

Similar & Simpler Interface

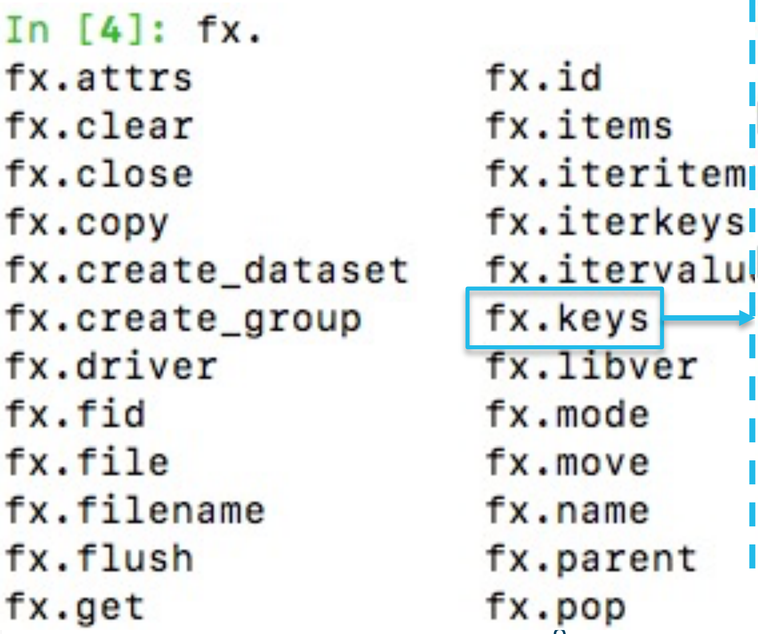
H5Py	HDF5
w- or x	H5F_ACC_EXCL
w	H5F_ACC_TRUNC
r	H5F_RDONLY
r+	H5F_ACC_RDWR
a (default)	*H5F_ACC_RDWR* *&H5F_ACC_EXCL*

Everything is Object

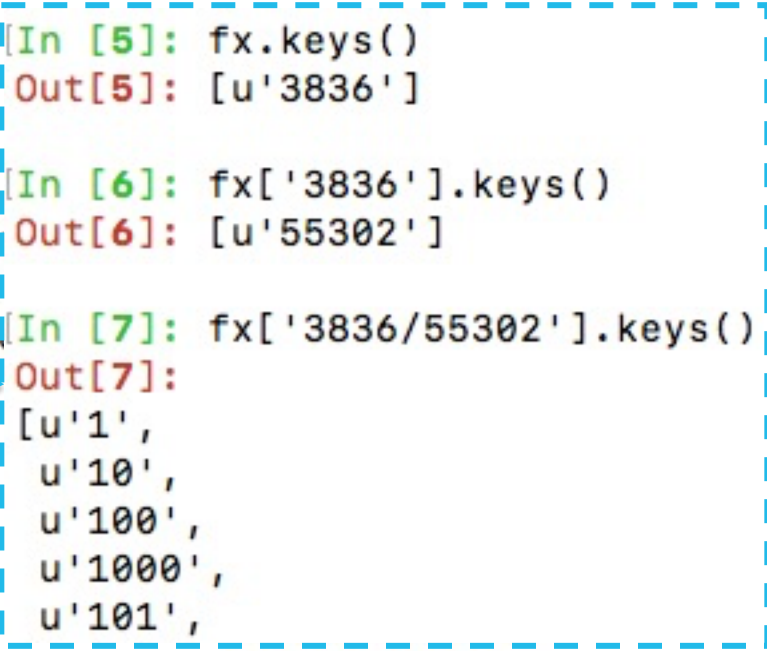


```
fx=h5py.File('output.h5','w')
```

File Object

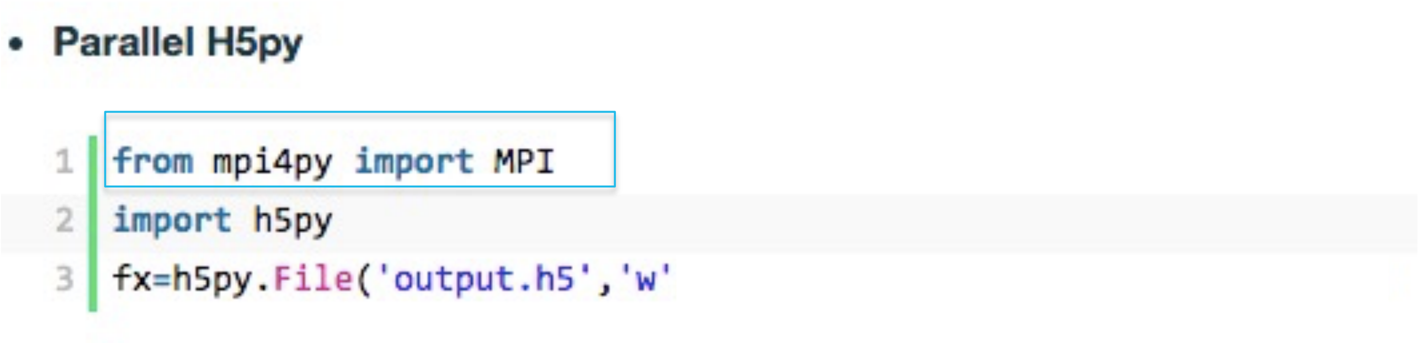


```
In [4]: fx.
fx.attrs                    fx.id
fx.clear                    fx.items
fx.close                    fx.iteritem
fx.copy                     fx.iterkeys
fx.create_dataset           fx.itervalu
fx.create_group             fx.keys
fx.driver                   fx.libver
fx.fid                      fx.mode
fx.file                     fx.move
fx.filename                 fx.name
fx.flush                    fx.parent
fx.get                      fx.pop
```



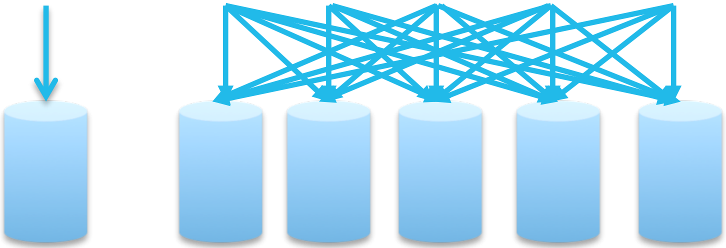
```
In [5]: fx.keys()
Out[5]: [u'3836']

In [6]: fx['3836'].keys()
Out[6]: [u'55302']

In [7]: fx['3836/55302'].keys()
Out[7]:
[u'1',
 u'10',
 u'100',
 u'1000',
 u'101',
```

One Line to Enable Parallel I/O

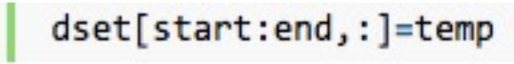


- **Parallel H5py**

```python
from mpi4py import MPI
import h5py
fx=h5py.File('output.h5','w'
```

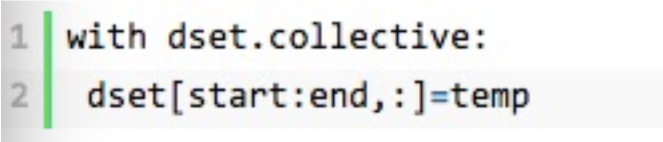


Two-Phase Collective IO, NERSC Contributions



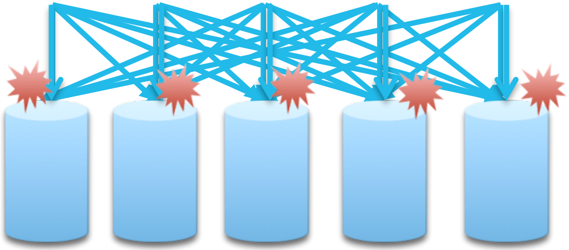
```
dset[start:end,:]=temp
```

Independent IO



```
with dset.collective:
 dset[start:end,:]=temp
```

Collective IO



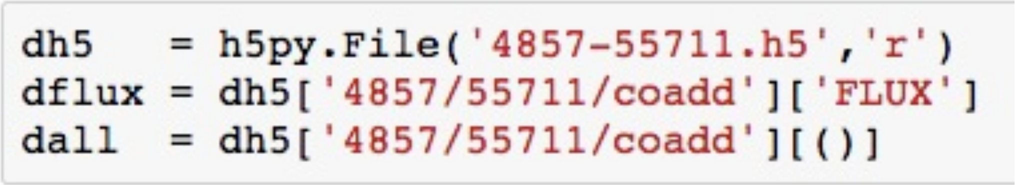
WARP

Jean-Luc. Vay, Remi. Lehe, LBNL

Collective IO

- Reduces the IO contention on server side
- Aggregates small IO into larger contiguous IO

Looks Like Numpy Arrays



```
dh5   = h5py.File('4857-55711.h5','r')
dflux = dh5['4857/55711/coadd']['FLUX']
dall  = dh5['4857/55711/coadd'][()]
```

Field Slicing

Path to the dataset

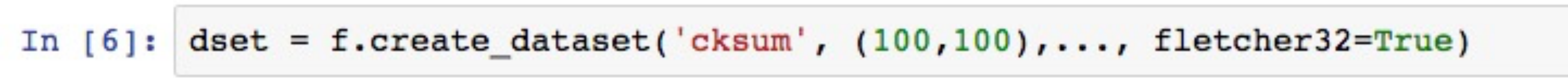
- **Indices**: anything that can be converted to a Python long
- **Slices** (i.e. [:] or [0:10])
- **Field names**, in the case of compound data
- At most one **Ellipsis** (...) object
- Limited **fancy slicing**, e.g., dset[1:6, [5,8,9]], *use with caution*

Beyond Numpy Arrays

Dataset Object

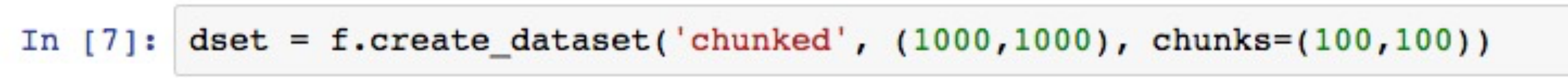
- ✧ Error-detection
- ✧ Chunking
- ✧ Compression

Checksum



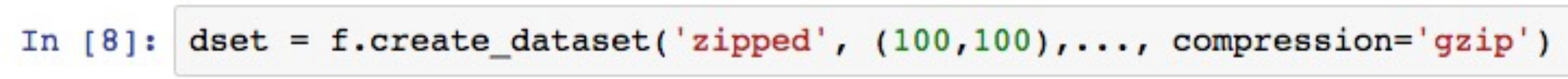
```
In [6]: dset = f.create_dataset('cksum', (100,100),..., fletcher32=True)
```

Chunking



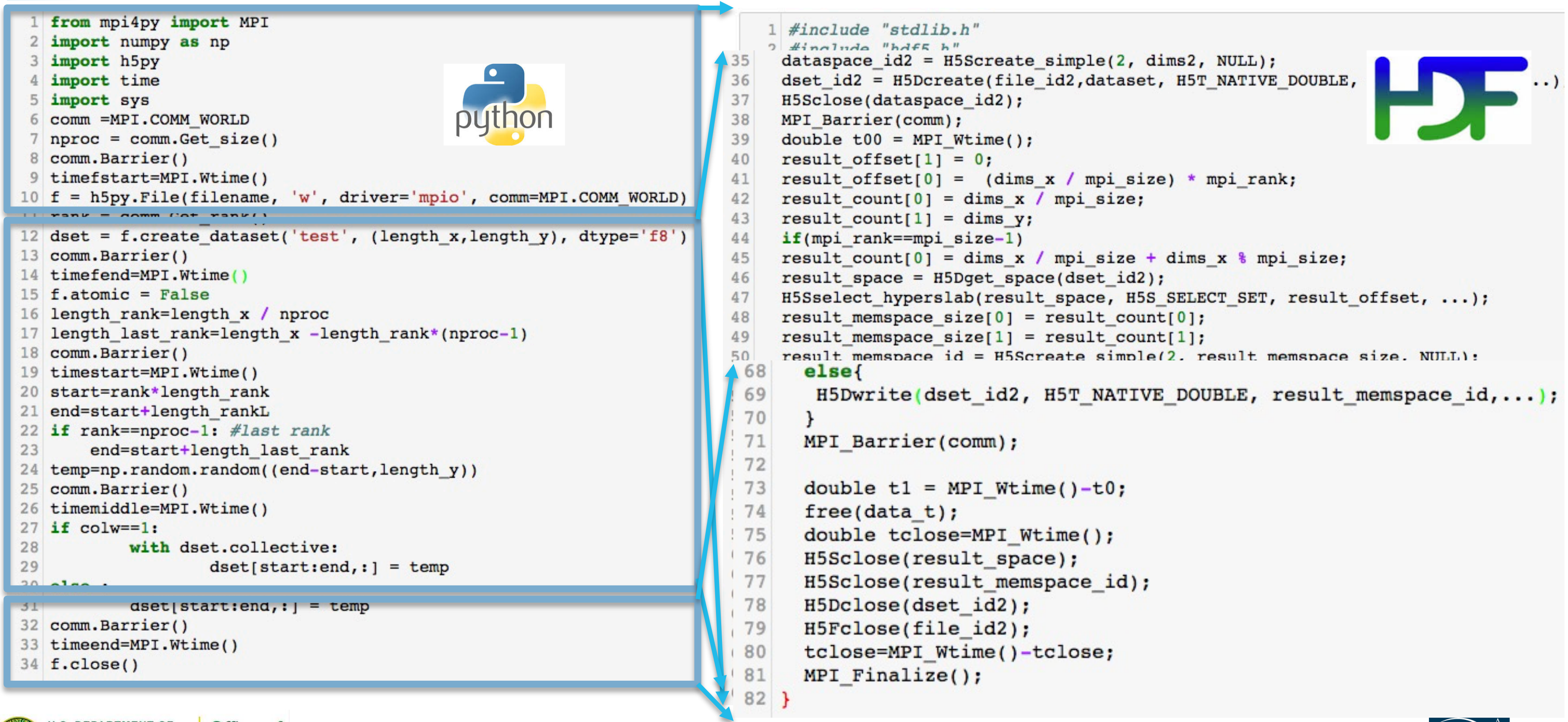
```
In [7]: dset = f.create_dataset('chunked', (1000,1000), chunks=(100,100))
```

Compression



```
In [8]: dset = f.create_dataset('zipped', (100,100),..., compression='gzip')
```

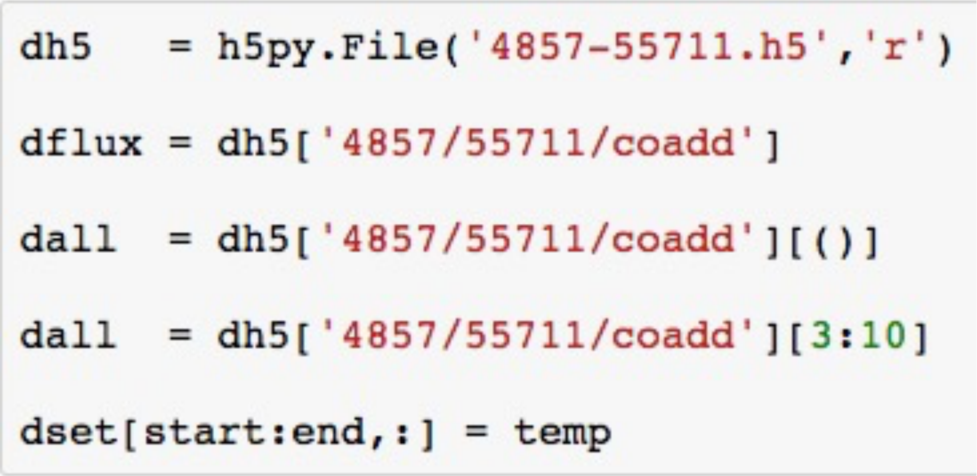
Coding Efforts



```python
from mpi4py import MPI
import numpy as np
import h5py
import time
import sys
comm =MPI.COMM_WORLD
nproc = comm.Get_size()
comm.Barrier()
timefstart=MPI.Wtime()
f = h5py.File(filename, 'w', driver='mpio', comm=MPI.COMM_WORLD)
rank = comm.Get_rank()
dset = f.create_dataset('test', (length_x,length_y), dtype='f8')
comm.Barrier()
timefend=MPI.Wtime()
f.atomic = False
length_rank=length_x / nproc
length_last_rank=length_x -length_rank*(nproc-1)
comm.Barrier()
timestart=MPI.Wtime()
start=rank*length_rank
end=start+length_rankL
if rank==nproc-1: #last rank
    end=start+length_last_rank
temp=np.random.random((end-start,length_y))
comm.Barrier()
timemiddle=MPI.Wtime()
if colw==1:
        with dset.collective:
                dset[start:end,:] = temp
else :
        dset[start:end,:] = temp
comm.Barrier()
timeend=MPI.Wtime()
f.close()
```

```c
#include "stdlib.h"
#include "hdf5.h"
dataspace_id2 = H5Screate_simple(2, dims2, NULL);
dset_id2 = H5Dcreate(file_id2,dataset, H5T_NATIVE_DOUBLE, ..)
H5Sclose(dataspace_id2);
MPI_Barrier(comm);
double t00 = MPI_Wtime();
result_offset[1] = 0;
result_offset[0] =  (dims_x / mpi_size) * mpi_rank;
result_count[0] = dims_x / mpi_size;
result_count[1] = dims_y;
if(mpi_rank==mpi_size-1)
result_count[0] = dims_x / mpi_size + dims_x % mpi_size;
result_space = H5Dget_space(dset_id2);
H5Sselect_hyperslab(result_space, H5S_SELECT_SET, result_offset, ...);
result_memspace_size[0] = result_count[0];
result_memspace_size[1] = result_count[1];
result_memspace_id = H5Screate_simple(2, result_memspace_size, NULL);
  else{
   H5Dwrite(dset_id2, H5T_NATIVE_DOUBLE, result_memspace_id,...);
  }
  MPI_Barrier(comm);

  double t1 = MPI_Wtime()-t0;
  free(data_t);
  double tclose=MPI_Wtime();
  H5Sclose(result_space);
  H5Sclose(result_memspace_id);
  H5Dclose(dset_id2);
  H5Fclose(file_id2);
  tclose=MPI_Wtime()-tclose;
  MPI_Finalize();
}
```

Coding Efforts: Implicit IO



```
dh5   = h5py.File('4857-55711.h5','r')

dflux = dh5['4857/55711/coadd']

dall  = dh5['4857/55711/coadd'][()]

dall  = dh5['4857/55711/coadd'][3:10]

dset[start:end,:] = temp
```

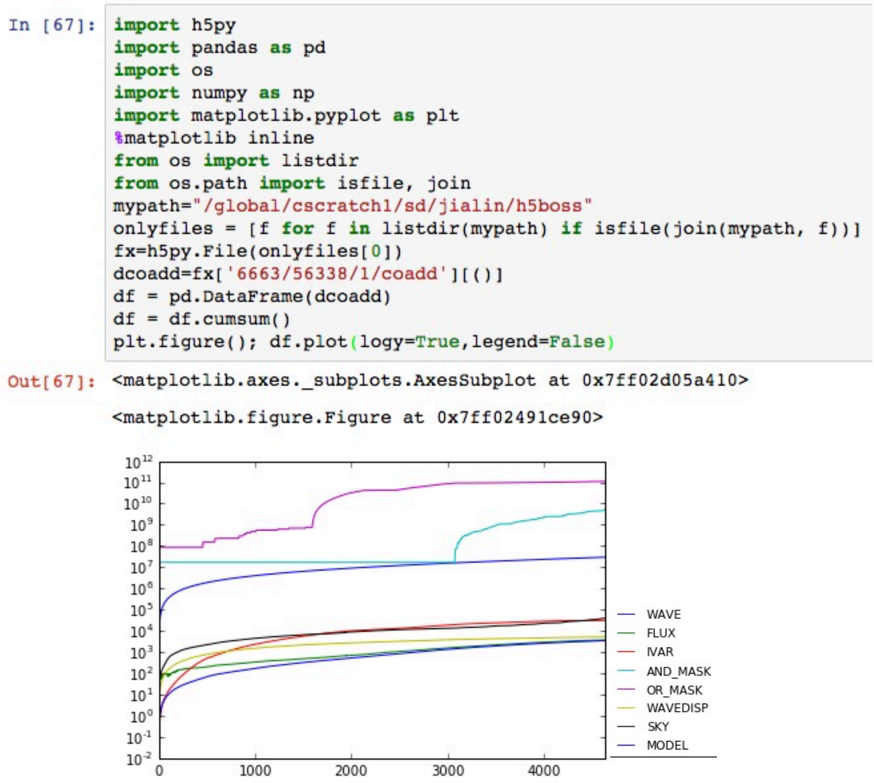
No Data IO

Yes

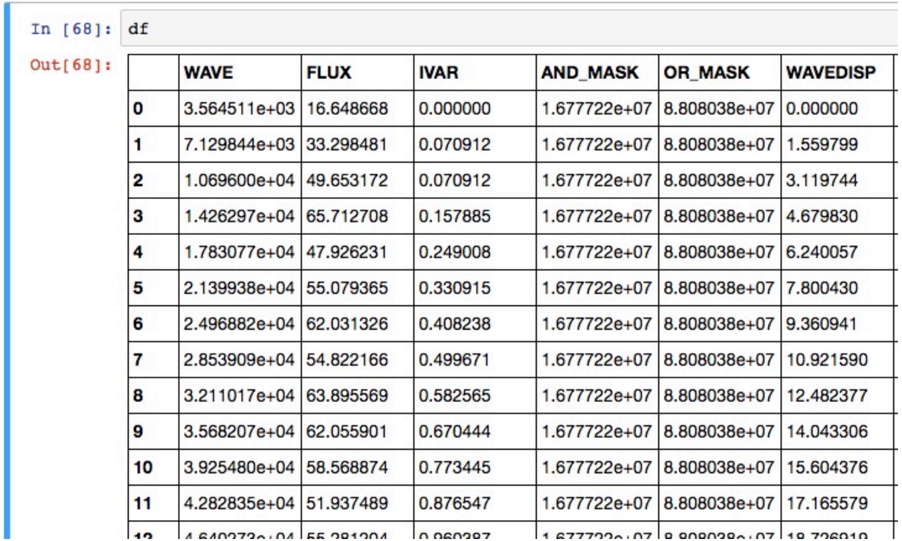
Yes, but partial

Yes, but partial

Exploring Interactively on Notebook



```
In [67]: import h5py
         import pandas as pd
         import os
         import numpy as np
         import matplotlib.pyplot as plt
         %matplotlib inline
         from os import listdir
         from os.path import isfile, join
         mypath="/global/cscratch1/sd/jialin/h5boss"
         onlyfiles = [f for f in listdir(mypath) if isfile(join(mypath, f))]
         fx=h5py.File(onlyfiles[0])
         dcoadd=fx['6663/56338/1/coadd'][()]
         df = pd.DataFrame(dcoadd)
         df = df.cumsum()
         plt.figure(); df.plot(logy=True,legend=False)

Out[67]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff02d05a410>

         <matplotlib.figure.Figure at 0x7ff02491ce90>
```

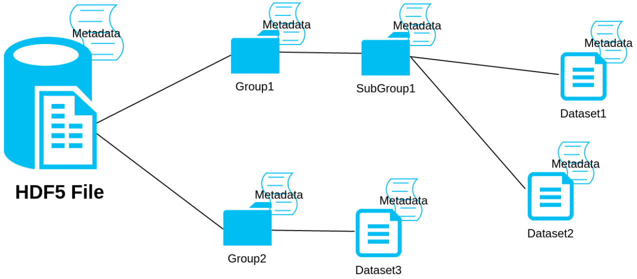


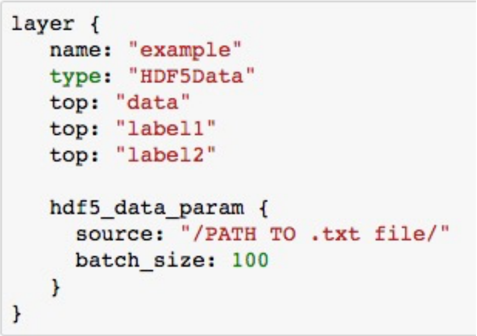
```
In [68]: df

Out[68]:
```

	WAVE	FLUX	IVAR	AND_MASK	OR_MASK	WAVEDISP
0	3.564511e+03	16.648668	0.000000	1.677722e+07	8.808038e+07	0.000000
1	7.129844e+03	33.298481	0.070912	1.677722e+07	8.808038e+07	1.559799
2	1.069600e+04	49.653172	0.070912	1.677722e+07	8.808038e+07	3.119744
3	1.426297e+04	65.712708	0.157885	1.677722e+07	8.808038e+07	4.679830
4	1.783077e+04	47.926231	0.249008	1.677722e+07	8.808038e+07	6.240057
5	2.139938e+04	55.079365	0.330915	1.677722e+07	8.808038e+07	7.800430
6	2.496882e+04	62.031326	0.408238	1.677722e+07	8.808038e+07	9.360941
7	2.853909e+04	54.822166	0.499671	1.677722e+07	8.808038e+07	10.921590
8	3.211017e+04	63.895569	0.582565	1.677722e+07	8.808038e+07	12.482377
9	3.568207e+04	62.055901	0.670444	1.677722e+07	8.808038e+07	14.043306
10	3.925480e+04	58.568874	0.773445	1.677722e+07	8.808038e+07	15.604376
11	4.282835e+04	51.937489	0.876547	1.677722e+07	8.808038e+07	17.165579
12	4.640273e+04	55.281204	0.960387	1.677722e+07	8.808038e+07	18.726919

https://ipython.nersc.gov

Learning the Data Easily





```
layer {
   name: "example"
   type: "HDF5Data"
   top: "data"
   top: "label1"
   top: "label2"

   hdf5_data_param {
     source: "/PATH TO .txt file/"
     batch_size: 100
   }
}
```

HDF5 Data Layer in Caffe

module load deeplearning

Productivity --> Performance?

- H5py: Productivity
  - Similar/Simpler Interface
  - Everything is Object
  - One Line to Parallel I/O
  - Beyond Numpy
  - Productive Coding
  - Seamlessly Importable in Notebook, etc
- H5py: Performance
  - ?
  - ?

Challenging: More than Single IO Layer

python

h5py

mpi4py

numpy

HDF

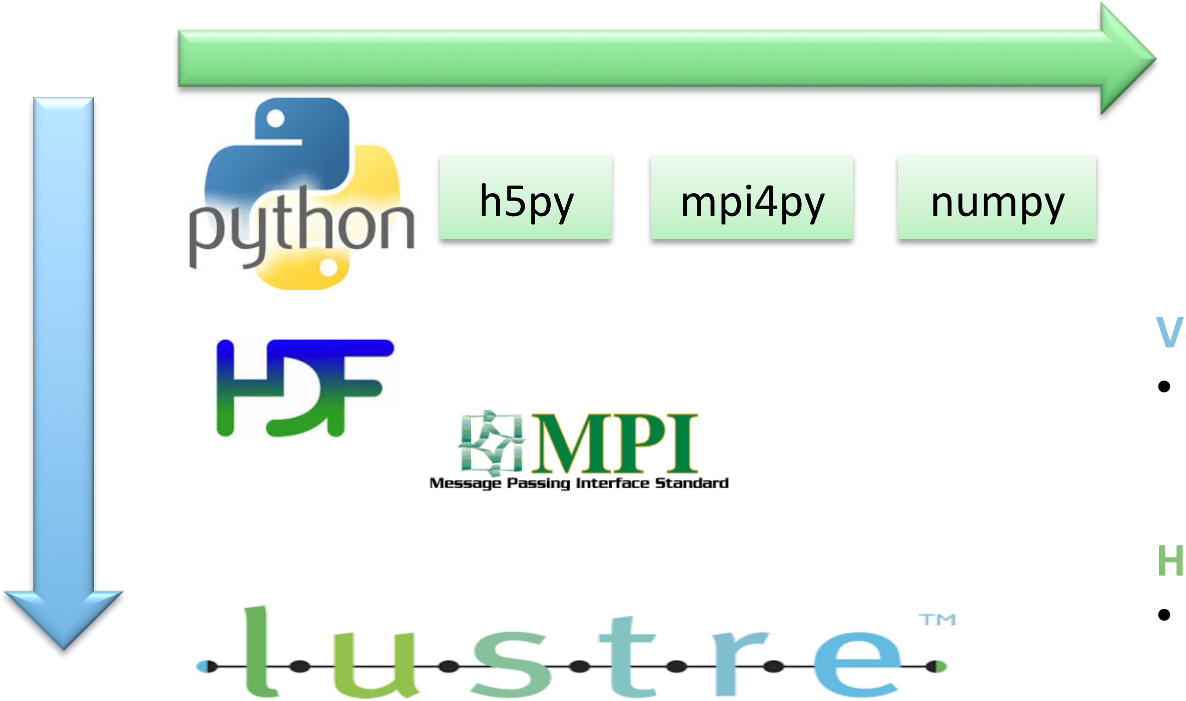
MPI
Message Passing Interface Standard

lustre™

"H5py performance is slow, parallel IO is not as good as serial IO"

DENIED

Views of Performance



h5py

mpi4py

numpy

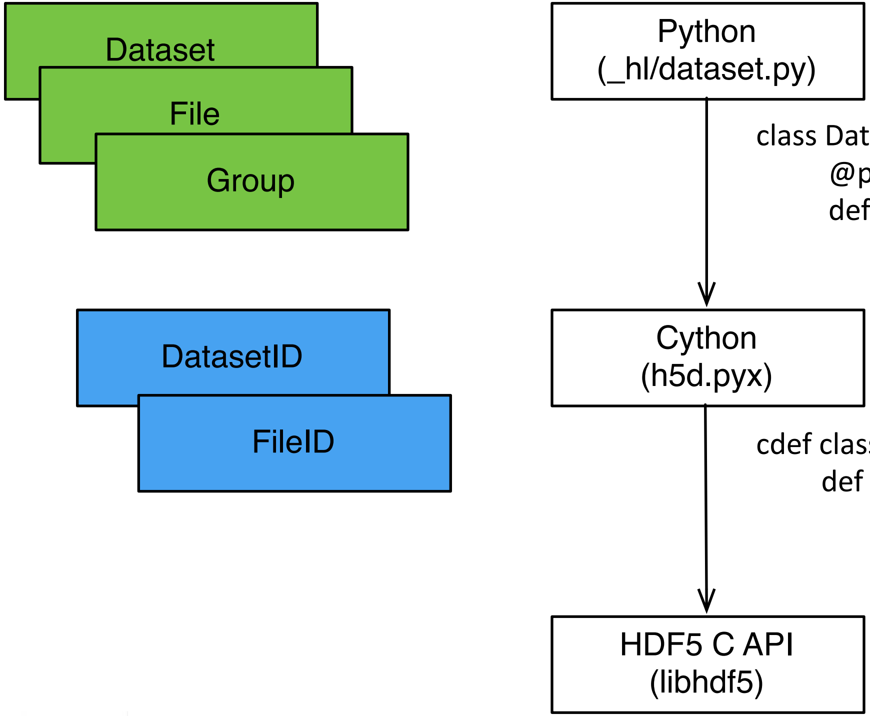
Vertical View:

- Performance penalty of python layer. e.g., H5py, Cython

Horizontal View:

- Scalability. e.g., mpi4py, srun

H5py Implementation (Vertical View)



```
class Dataset(HLObject):
        @property
        def storagesize(self):
              return self.id.get_storage_size()
```

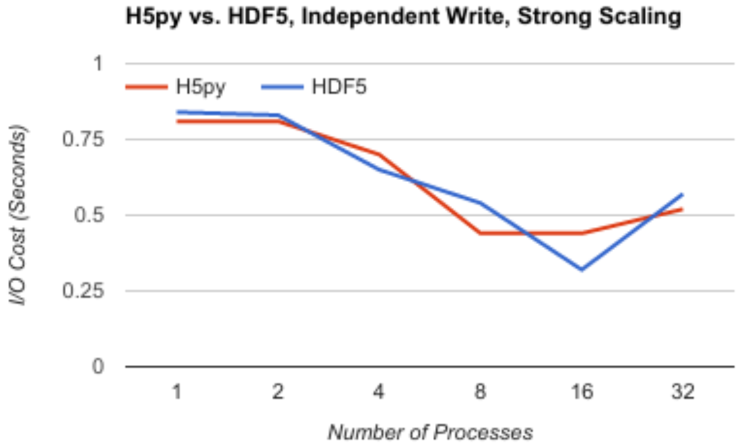
```
cdef class DatasetID(ObjectID):
        def get_storage_size(self):
              return H5Dget_storage_size(self.id)
```

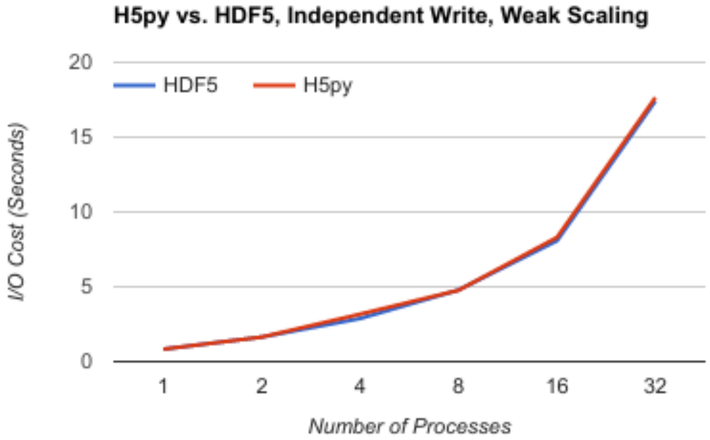
```
hsize_t H5Dget_storage_size(hid_t dset_id)
```

H5py Metadata Performance

Operation	H5py	HDF5	Details	Ratio
1K File Creation (s)	4.7	3.0	Create a file then close the file	63.8%
1K Object Scanning (s)	4.5	2.7	Open a group then scan all objects: group, dataset, link, etc	60.0%

H5py vs. HDF5 Single Node Independent I/O





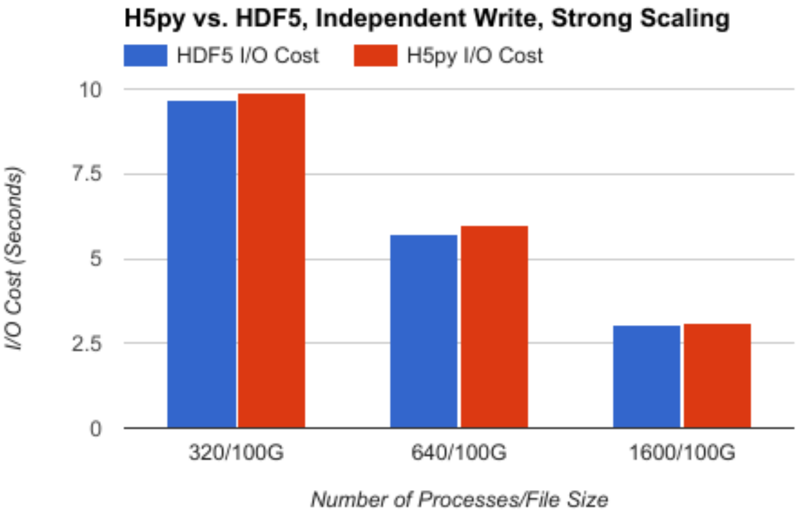
Strong Scaling, 800MB

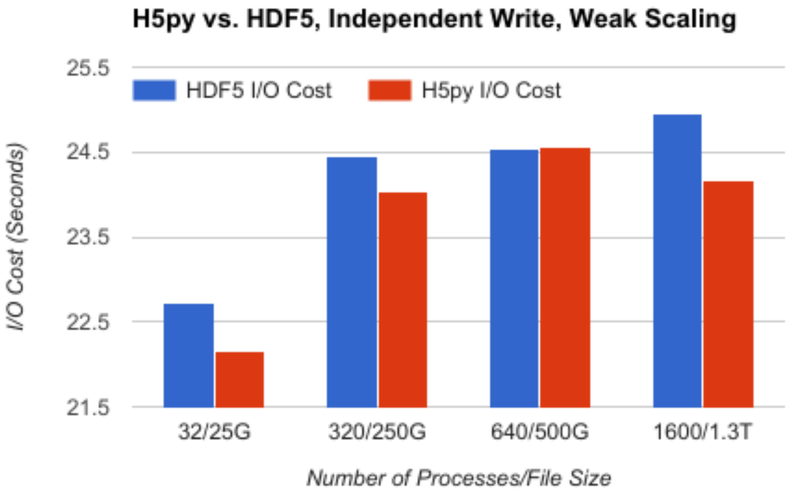
Weak Scaling, 800MB/Process

100%

97.8%

H5py vs. HDF5 Multi-node Independent I/O





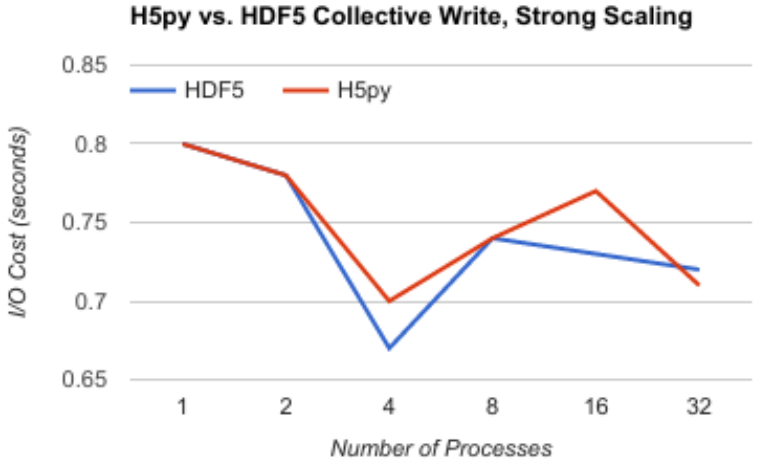
Strong Scaling

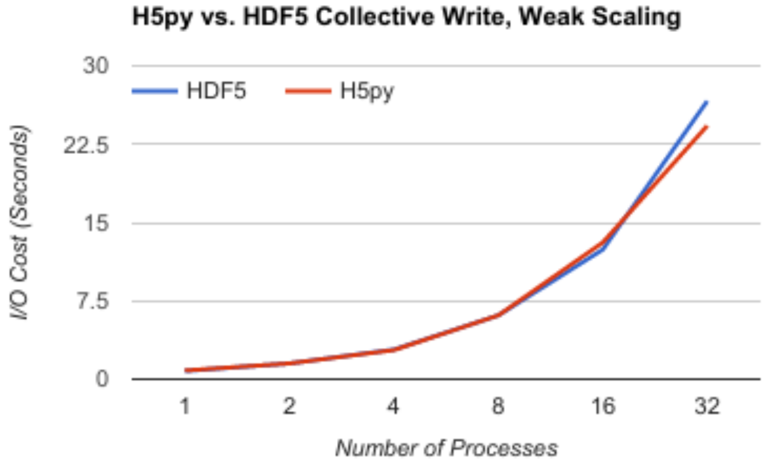
Weak Scaling

97.1%

100%

H5py vs. HDF5 Single Node Collective I/O





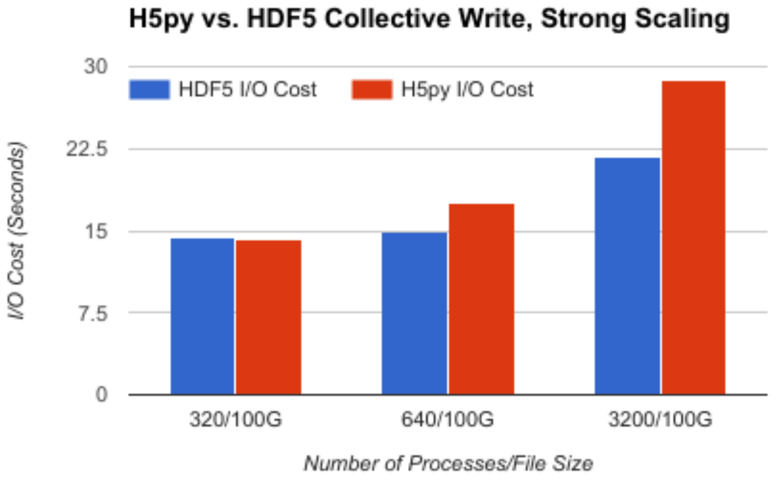
Strong Scaling, 800MB

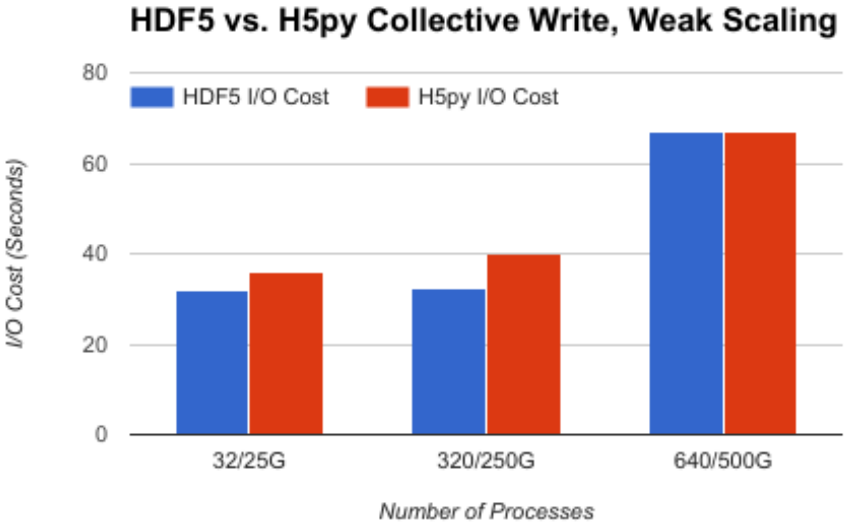
98.6%

Weak Scaling, 800MB/Process

100%

H5py vs. HDF5 Multi-node Collective I/O





Strong Scaling

84%, 101%, 75%
AVG: 87%

Weak Scaling

88%, 81%, 99%
AVG: 90%

H5py vs. HDF5 Performance

H5Py Performance / HDF5 Performance

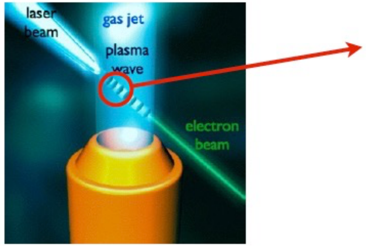
		Single Node	Multi-nodes
Metadata	1k File Creation	63.8%	
	1k Object Scanning	60.0%	
Independent I/O	Weak Scaling	97.8%	100%
	Strong Scaling	100%	97.1%
Collective I/O	Weak Scaling	100%	90%
	Strong Scaling	98.6%	87%

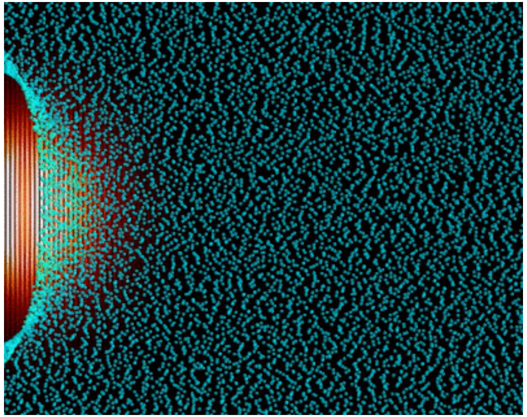
Case Study I: Warp

- Particle-in-cell simulation codes
- Alex Friedman, David Grote, 1980s
- LBNL, LLNL, and PPPL
- Broad variety of integrated physics models and extensive diagnostics
- Laser-wakefield

Physics of laser-wakefield

- The laser pulse pushes away the electrons of the gas
- This creates an accelerating structure

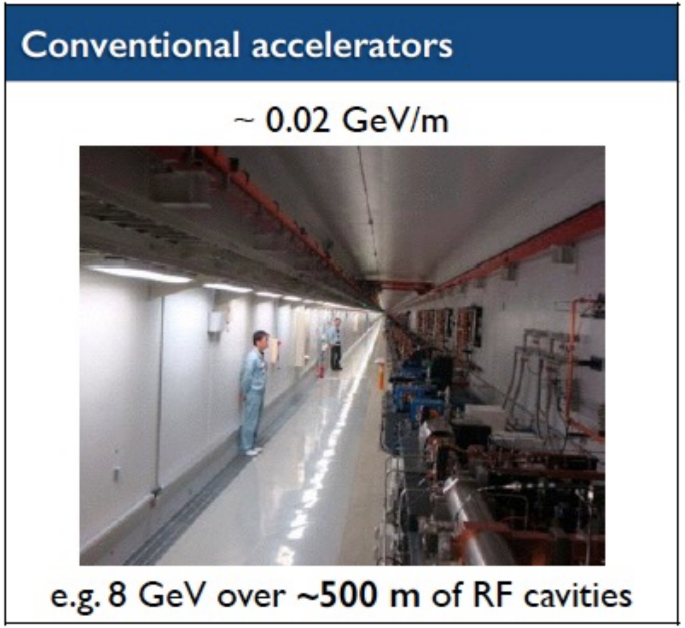




Remi Lehe, LBNL, GTC 2017

Case Study I: Warp

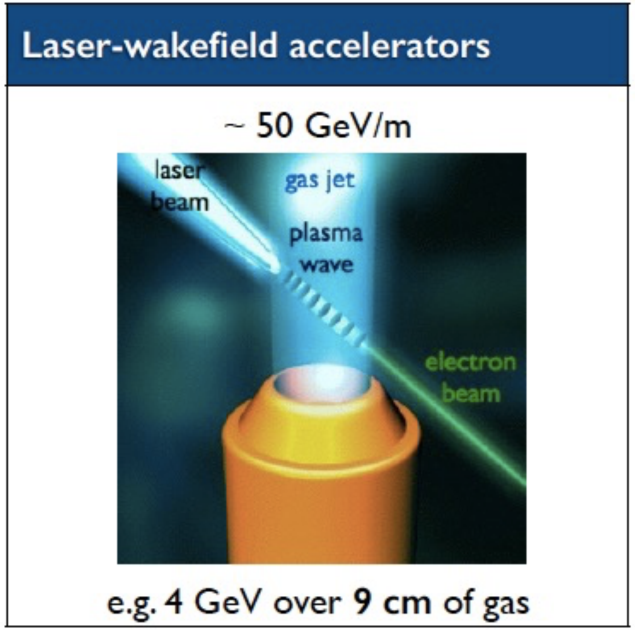
- ✧ Laser-wakefield 500 meters → 9 cm



Conventional accelerators

~ 0.02 GeV/m

e.g. 8 GeV over ~500 m of RF cavities



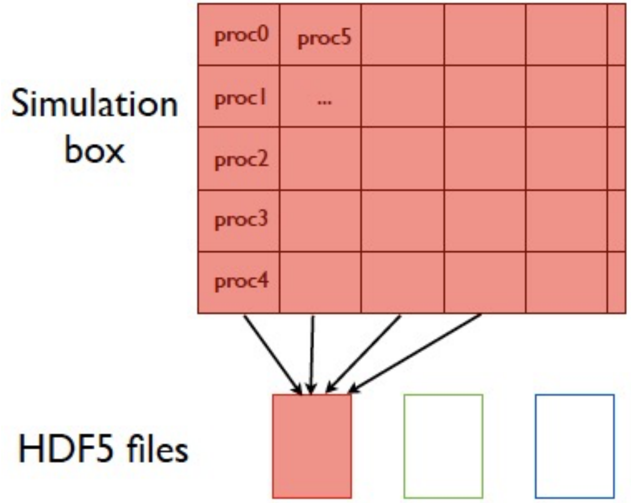
Laser-wakefield accelerators

~ 50 GeV/m

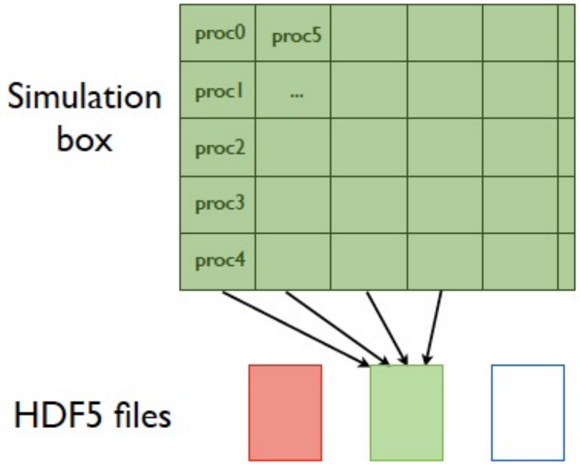
e.g. 4 GeV over 9 cm of gas

Remi Lehe, LBNL, GTC 2017

Case Study I: Warp IO with H5py



iteration 0



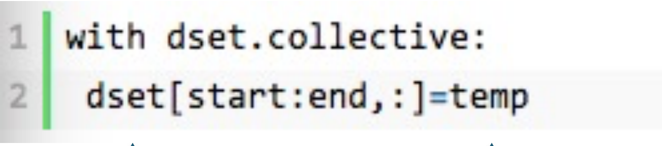
iteration 1

Perfom
~100 iterations of
the solver

Case Study I: Warp IO with H5py

- 172 - 600 MB per file
- With *parallel_output = False*, the simulation finished in less than 20 min, so it took less than 20 min to write all these files.
- With *parallel_output = True*, the simulation only had time to write the first 2 files (out of 80!)

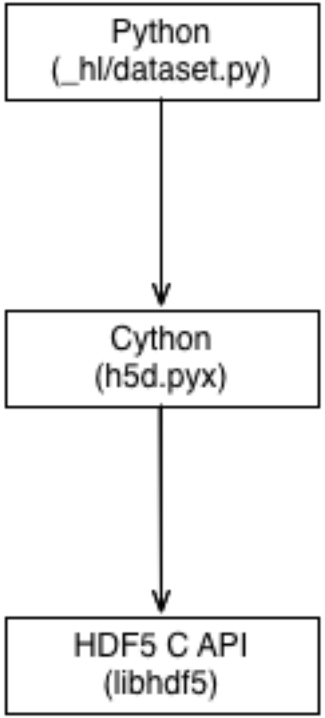
Case Study I: Warp IO with H5py



```
with dset.collective:
  dset[start:end,:]=temp
```

32 bit

64 bit

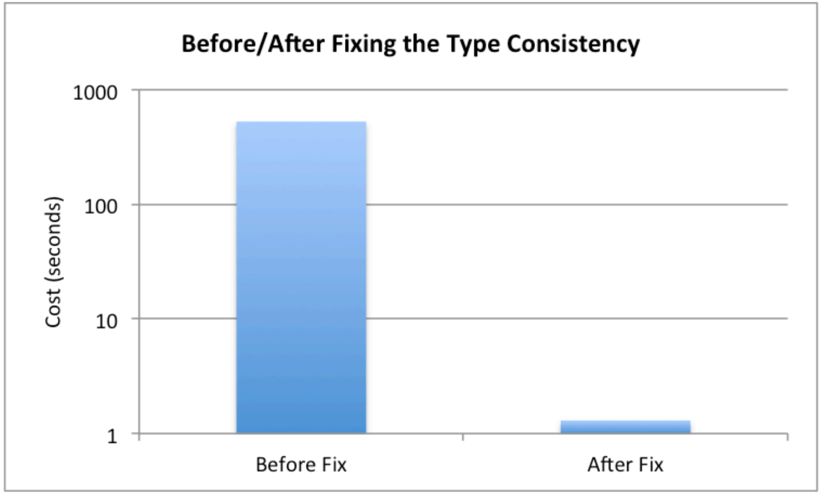


“Let there be Parallel I/O”

“Sorry, you broke my rules”

Case Study I: Warp IO with H5py

- Numpy array in Warp is using 64 bits
- H5py dataset in Warp is created with 32 bits float, *dtype='f'*
- HDF5 internally checks the type consistency
- Refuses to use collective I/O in case of inconsistency

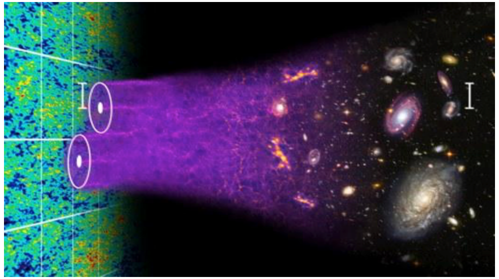


Alex Sim, CRD/LBNL

- Before Fix: 527 seconds
- After Fix: 1.3 seconds

Case Study II: H5Boss

- BOSS Baryon Oscillation Spectroscopic Survey – from SDSS
- Perform typical randomly generated query to extract small amount of stars/ galaxies from millions
- Run on final release of SDSS-III complete BOSS dataset
- **H5Boss**: A H5py based python package for:
  - Reformatting Fits to HDF5 files
  - Querying/Subsetting Fiber datasets



Baryon acoustic oscillations in early universe, still can be seen in survey like **BOSS**, (courtesy of Chris Blake and Sam Moorfield)

Jialin Liu, Debbie Bard, Quincey Koziol, Stephen Bailey, Prabhat, "H5Boss: A HDF5 based Python Package for BOSS Spectroscopic Survey Data", *In Submission*

Case Study II: H5Boss IO

User `cat1` is looking for 1 million fibers

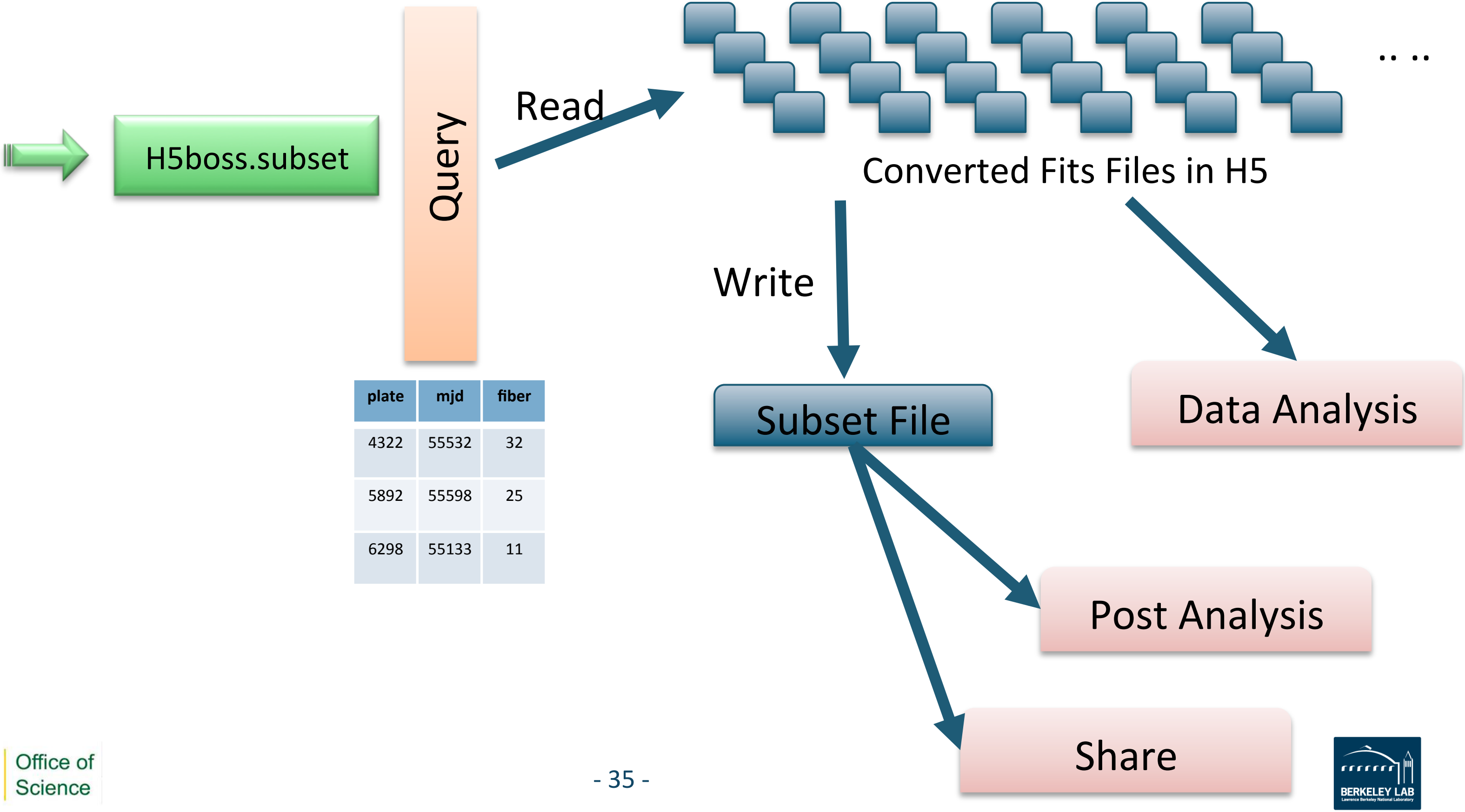
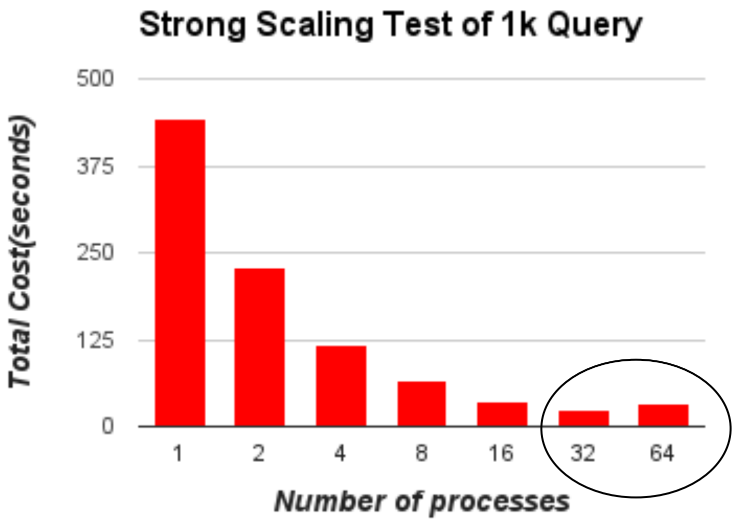


plate	mjd	fiber
4322	55532	32
5892	55598	25
6298	55133	11

Case Study II: H5Boss IO



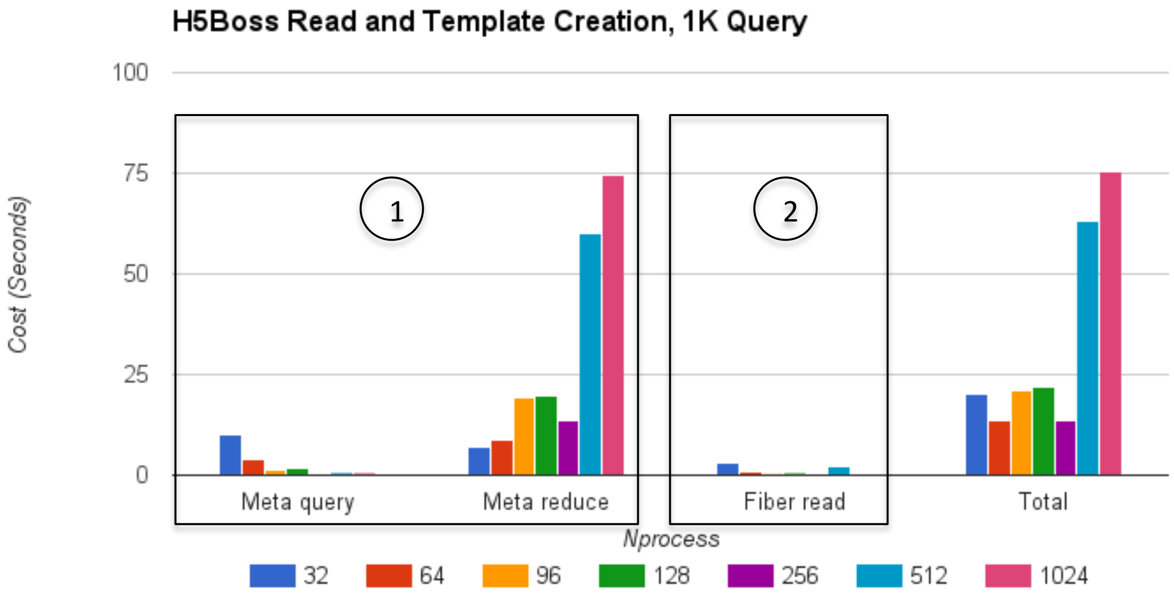
Before optimization, with 1k query, strong scaling:

- Scalable on single node
- Not scalable on multiple nodes

Case Study II: H5Boss IO

1 Query

2 Read



From 1 node to 32 nodes

Case Study II: H5Boss IO

Each Process: Query

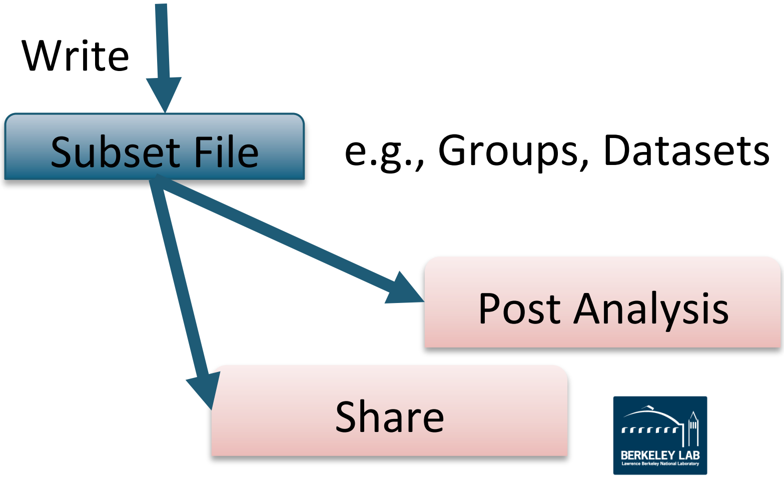
- 🟢 1. Files open
- 🟢 2. Plate/mjd/fiber scanning/searching
- 🟢 3. Key-value construction, needed for creating the shared file

All Processes: Communication

- 🔴 4. All to all reduction to form a global shared k-v list

Checklist:

1. The overhead in allreduce
2. The k,v structure



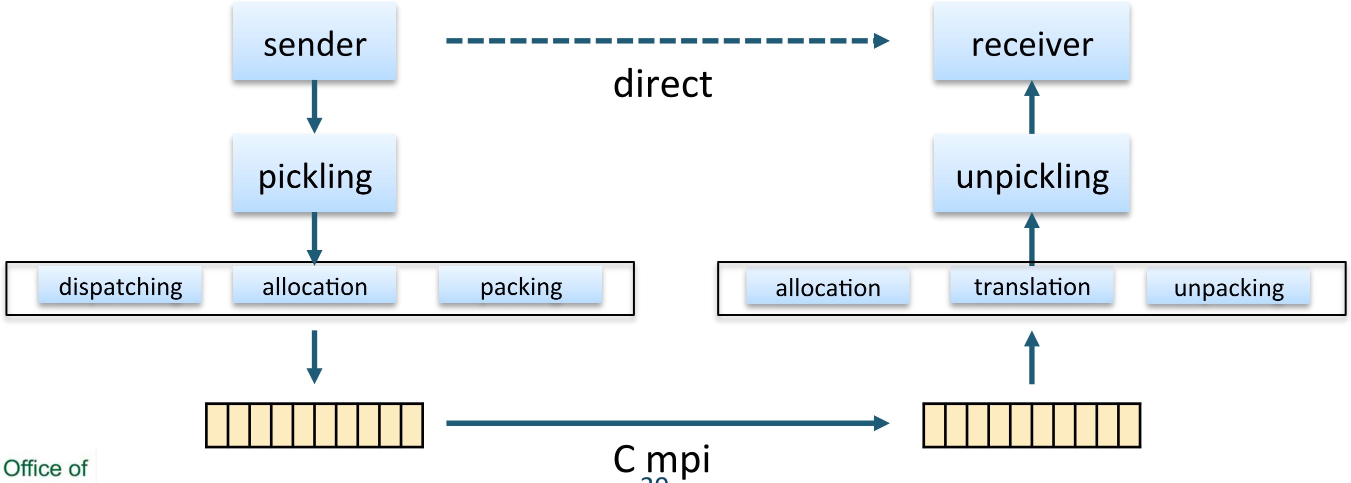
Case Study II: H5Boss IO

Why mpi4py’s allreduce could be an issue?

1. Allreduce vs allreduce

- Lowercase: generic Python objects, send(), recv(), etc
- Upper-case: buffer like object, Send(), Recv(), etc

2. pickling & unpickling



Case Study II: H5Boss IO

- **Re-design the key, value pair to be buffer like**

Key: Path to HDF5 dataset
Value: (type, shape, path to file)

K:
- b[‘3666/55159/599/coadd’]

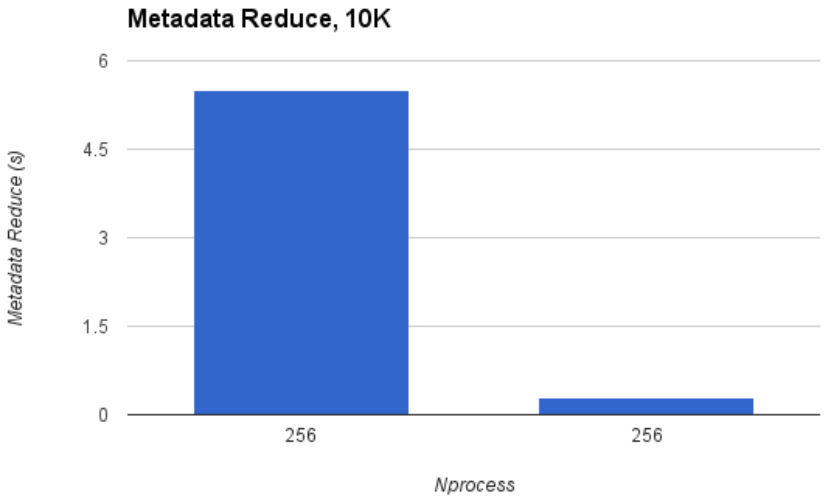
V:
- (([((WAVE, ‘<f4’), (FLUX, ‘<f4’), (IVAR, ‘<f4’), (AND_MAKS, ‘<i4’), (OR_MASK, ‘<i4’), (‘WAVEDISP’, ‘<f4’), (SKY, ‘<f4’), (MODEL,‘<f4’)]),
- (4619,)
- ‘/global/cscratch1/sd/jialin/h5boss/3666-55159.hdf5’)

Key: Path to HDF5 dataset
Value: shape

K: (str) “3666/55159/599/coadd”
V: (int) “4619”

Case Study II: H5Boss IO

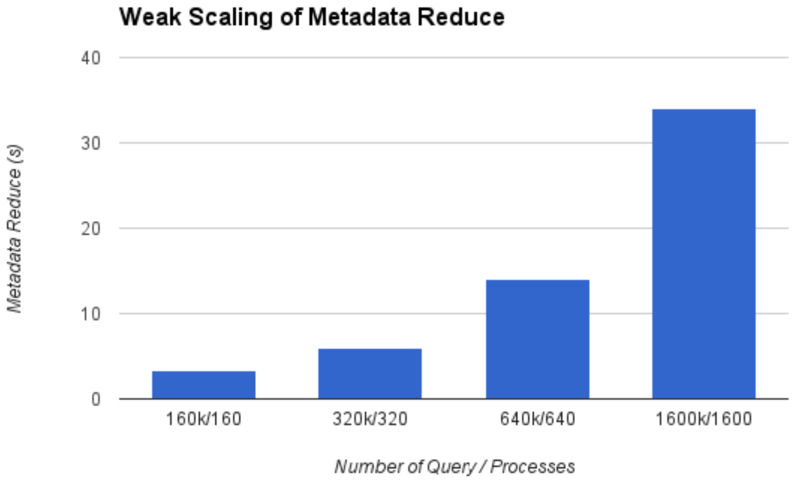
With optimized (k,v) structure, 19X faster



Case Study II: H5Boss IO

With optimized (k,v) structure

Weak scaling to 1.6 million fiber query and 1600 processes

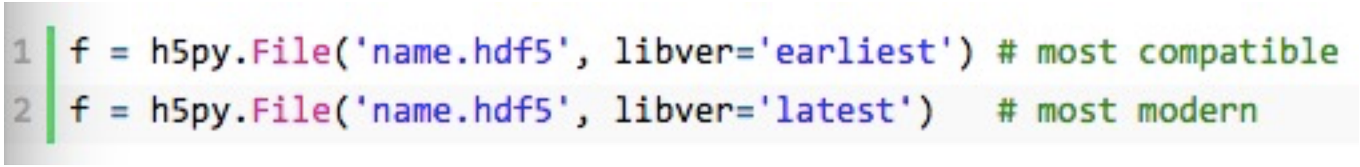


Productivity --> Performance

- H5py: Productivity
  - Similar/simpler interface
  - ....
  - Seamlessly importable in notebook, …
- H5py: Performance
  - H5py often reaches 90% of HDF5 performance in benchmarking
  - In practice, case by case:
    - Type consistency
    - Object vs. Buffer
    - …

H5py other Best Practice

1. Optimal HDF5 file creation



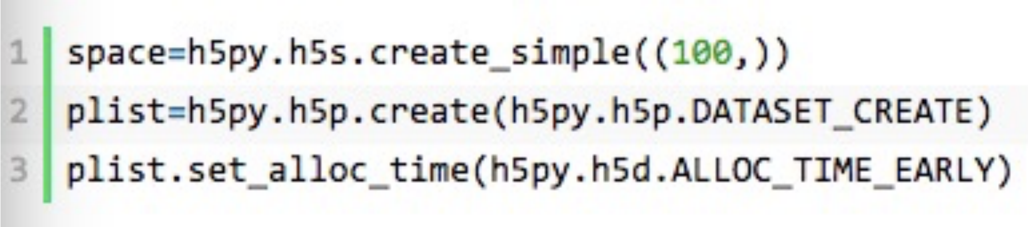
```
f = h5py.File('name.hdf5', libver='earliest') # most compatible
f = h5py.File('name.hdf5', libver='latest')   # most modern
```

2.25X

Choose the most modern format [optional]

H5py other Best Practice

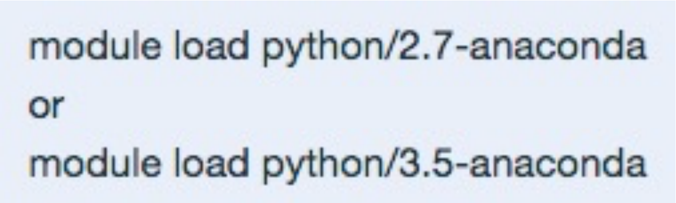
2. Use low-level API in H5py



```
space=h5py.h5s.create_simple((100,))
plist=h5py.h5p.create(h5py.h5p.DATASET_CREATE)
plist.set_alloc_time(h5py.h5d.ALLOC_TIME_EARLY)
```

Get closer to the HDF5 C library, fine tuning

H5py at NERSC



```
module load python/2.7-anaconda
or
module load python/3.5-anaconda
```

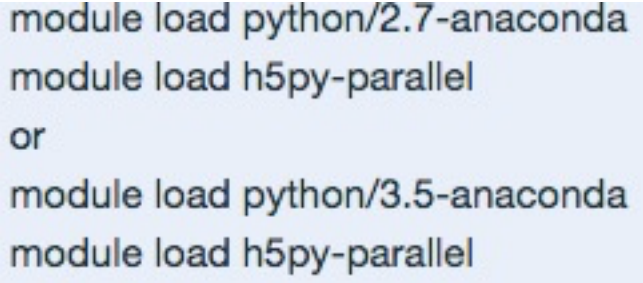
Serial H5py

Anaconda includes h5py package

- H5py 2.6.0
- Built-in hdf5 library, 1.8.17
- Easy use with other packages
- *No parallel support*

Works on both Edison and Cori

H5py at NERSC



```
module load python/2.7-anaconda
module load h5py-parallel
or
module load python/3.5-anaconda
module load h5py-parallel
```

H5py-parallel @ NERSC

- H5py 2.6.0
- Compiled with cray-hdf5-parallel/1.8.16
- No conflict with anaconda's serial h5py
  - Import h5py (perfectly fine)
  - Can use together with anaconda
- Up to date features

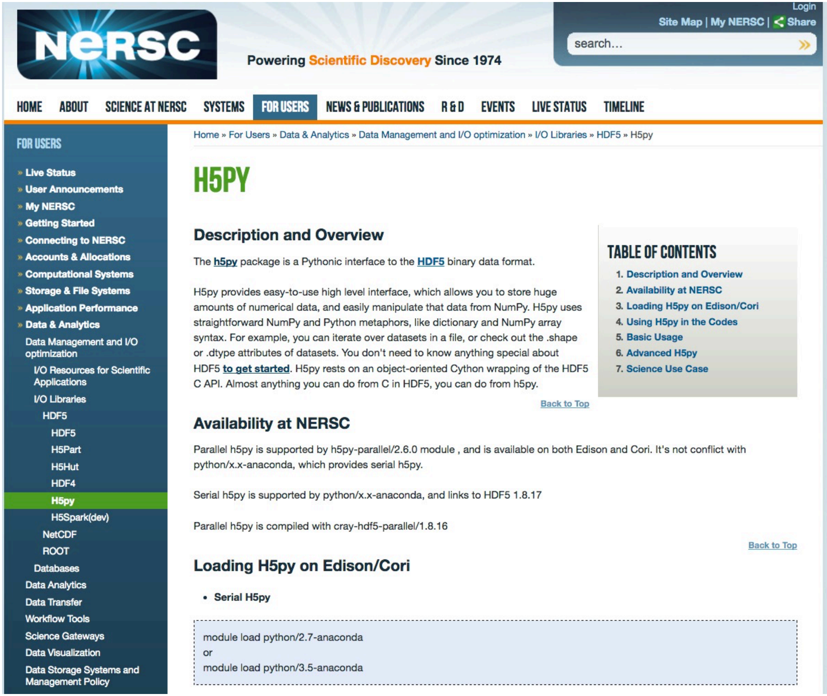
Rollin Thomas

H5py at NERSC

High Performance H5py with Sample Codes

http://www.nersc.gov/users/data-analytics/data-management/i-o-libraries/hdf5-2/h5py/

Thanks



H5PY

Description and Overview

The h5py package is a Pythonic interface to the HDF5 binary data format.

H5py provides easy-to-use high level interface, which allows you to store huge amounts of numerical data, and easily manipulate that data from NumPy. H5py uses straightforward NumPy and Python metaphors, like dictionary and NumPy array syntax. For example, you can iterate over datasets in a file, or check out the .shape or .dtype attributes of datasets. You don't need to know anything special about HDF5 to get started. H5py rests on an object-oriented Cython wrapping of the HDF5 C API. Almost anything you can do from C in HDF5, you can do from h5py.

Back to Top

Availability at NERSC

Parallel h5py is supported by h5py-parallel/2.6.0 module , and is available on both Edison and Cori. It's not conflict with python/x.x-anaconda, which provides serial h5py.

Serial h5py is supported by python/x.x-anaconda, and links to HDF5 1.8.17

Parallel h5py is compiled with cray-hdf5-parallel/1.8.16

Back to Top

Loading H5py on Edison/Cori

- **Serial H5py**

```
module load python/2.7-anaconda
or
module load python/3.5-anaconda
```