Productivity and High Performance, Can we have both?
An Exploration of Parallel-H5py from I/O Perspective

May 26, 2017

Jialin Liu
Data Analytics & Service Group
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 Datatype

- Integer: 32-bit, LE

HDF5 Dataspace

<table>
<thead>
<tr>
<th>Rank</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Dim[0] = 4</td>
</tr>
<tr>
<td></td>
<td>Dim[1] = 5</td>
</tr>
</tbody>
</table>

Specifications for single data element and array dimensions

Multi-dimensional array of identically typed data elements
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

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

```
file_id = H5Fcreate("output.h5", H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);
```
## Similar & Simpler Interface

<table>
<thead>
<tr>
<th>H5Py</th>
<th>HDF5</th>
</tr>
</thead>
<tbody>
<tr>
<td>w- or x</td>
<td>H5F_ACC_EXCL</td>
</tr>
<tr>
<td>w</td>
<td>H5F_ACC_TRUNC</td>
</tr>
<tr>
<td>r</td>
<td>H5F_RDONLY</td>
</tr>
<tr>
<td>r+</td>
<td>H5F_ACC_RDWR</td>
</tr>
<tr>
<td>a (default)</td>
<td>H5F_ACC_RDWR &amp; H5F_ACC_EXCL</td>
</tr>
</tbody>
</table>
Everything is Object

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

In [4]: fx.
   fx.attrs
   fx.clear
   fx.close
   fx.copy
   fx.create_dataset
   fx.create_group
   fx.driver
   fx.fid
   fx.file
   fx.filename
   fx.flush
   fx.get

fx.keys
```

```
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

Independent IO

Collective IO

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

Jean-Luc. Vay, Remi. Lehe, LBNL
Looks Like Numpy Arrays

Path to the dataset

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

- 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

- Error-detection
- Chunking
- Compression

Dataset Object

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()
timestart = MPI.Wtime()

f = h5py.File(filename, 'w', driver='mpio', comm=comm)

start = [0] * nproc
end = [0] * nproc
middle = [0] * nproc
myrank = comm.Get_rank()

status = comm.Bcast([start, end, middle, nproc], root=0)

# Local data set size calculation

start[mymark] = time.time()  # Start time for each process
local_data_size = local_data_size * nproc
local_data_size /= nproc
local_data_size = local_data_size
```

```python

# Initialize data set

data = np.zeros((10, 10, 10), dtype=np.float64)
data[0] = np.random.random()

# Broadcast data across processes

comm.Bcast([data], root=0)

# Perform computations

for i in range(10):
    for j in range(10):
        for k in range(10):
            data[i, j, k] = data[i, j, k] + 1

# Gather data from all processes

comm.Gather([data], [result], root=0)

# Print result

if comm.Get_rank() == 0:
    print(result)
```
Coding Efforts: Implicit IO

```python
import h5py

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

c = f['4857/55711/coadd']

c[3:10] = temp
```

- No Data IO
- Yes
- Yes, but partial
- Yes, but partial
Exploring Interactively on Notebook

```python
import h5py
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
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/dcoadd']()
df = pd.DataFrame(dcoadd)
df = df.cunsum()
plt.figure(); df.plot(logy=True, legend=False)
```

```
<table>
<thead>
<tr>
<th></th>
<th>FLUX</th>
<th>IVAR</th>
<th>AND_MASK</th>
<th>OR_MASK</th>
<th>WAVEDISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.554511e+03</td>
<td>16.648668</td>
<td>0.000000</td>
<td>1.67772e+07</td>
<td>8.608036e+07</td>
</tr>
<tr>
<td>1</td>
<td>7.129844e+03</td>
<td>33.296481</td>
<td>0.070912</td>
<td>1.67772e+07</td>
<td>8.608036e+07</td>
</tr>
<tr>
<td>2</td>
<td>1.069600e+04</td>
<td>49.653172</td>
<td>0.070912</td>
<td>1.67772e+07</td>
<td>8.608036e+07</td>
</tr>
<tr>
<td>3</td>
<td>1.426297e+04</td>
<td>65.712708</td>
<td>0.157885</td>
<td>1.67772e+07</td>
<td>8.608038e+07</td>
</tr>
<tr>
<td>4</td>
<td>1.783077e+04</td>
<td>47.926231</td>
<td>0.249008</td>
<td>1.67772e+07</td>
<td>8.608038e+07</td>
</tr>
<tr>
<td>5</td>
<td>2.139638e+04</td>
<td>55.079355</td>
<td>0.330915</td>
<td>1.67772e+07</td>
<td>8.608036e+07</td>
</tr>
<tr>
<td>6</td>
<td>2.496828e+04</td>
<td>62.031326</td>
<td>0.408238</td>
<td>1.67772e+07</td>
<td>8.608036e+07</td>
</tr>
<tr>
<td>7</td>
<td>2.853909e+04</td>
<td>54.822166</td>
<td>0.499761</td>
<td>1.67772e+07</td>
<td>8.608036e+07</td>
</tr>
<tr>
<td>8</td>
<td>3.211017e+04</td>
<td>63.695556</td>
<td>0.582555</td>
<td>1.67772e+07</td>
<td>8.608036e+07</td>
</tr>
<tr>
<td>9</td>
<td>3.568207e+04</td>
<td>62.056901</td>
<td>0.670444</td>
<td>1.67772e+07</td>
<td>8.608036e+07</td>
</tr>
<tr>
<td>10</td>
<td>3.925480e+04</td>
<td>58.568874</td>
<td>0.773446</td>
<td>1.67772e+07</td>
<td>8.608038e+07</td>
</tr>
<tr>
<td>11</td>
<td>4.282835e+04</td>
<td>51.837489</td>
<td>0.876547</td>
<td>1.67772e+07</td>
<td>8.608036e+07</td>
</tr>
<tr>
<td>12</td>
<td>4.640973e+04</td>
<td>55.314024</td>
<td>0.980937</td>
<td>1.67772e+07</td>
<td>8.608036e+07</td>
</tr>
</tbody>
</table>
```

https://ipython.nersc.gov
Learning the Data Easily

HDF5 Data Layer in Caffe

module load deplearning
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

“H5py performance is slow, parallel IO is not as good as serial IO”
Views of Performance

Vertical View:
• Performance penalty of python layer. e.g., H5py, Cython

Horizontal View:
• Scalability. e.g., mpi4py, srun
**H5py Implementation (Vertical View)**

```
H5Dget_storage_size(hid_t dset_id)
```

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

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

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

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

```
Pyhton
(_hl/dataset.py)
class Dataset(HLObject):
    @property
def storagesize(self):
        return self.id.get_storage_size()

Cython
(h5d.pyx)
cdef class DatasetID(ObjectID):
    def get_storage_size(self):
        return H5Dget_storage_size(self.id)

HDF5 C API
(libhdf5)
```
## H5py Metadata Performance

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

**H5py vs. HDF5, Independent Write, Strong Scaling**

<table>
<thead>
<tr>
<th>Number of Processes/File Size</th>
<th>I/O Cost (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>320/100G</td>
<td>H5py I/O Cost: 9.5</td>
</tr>
<tr>
<td></td>
<td>HDF5 I/O Cost: 10</td>
</tr>
<tr>
<td>640/100G</td>
<td>H5py I/O Cost: 7.5</td>
</tr>
<tr>
<td></td>
<td>HDF5 I/O Cost: 7</td>
</tr>
<tr>
<td>1600/100G</td>
<td>H5py I/O Cost: 6.5</td>
</tr>
<tr>
<td></td>
<td>HDF5 I/O Cost: 6</td>
</tr>
</tbody>
</table>

**H5py vs. HDF5, Independent Write, Weak Scaling**

<table>
<thead>
<tr>
<th>Number of Processes/File Size</th>
<th>I/O Cost (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32/25G</td>
<td>H5py I/O Cost: 2.5</td>
</tr>
<tr>
<td></td>
<td>HDF5 I/O Cost: 2.3</td>
</tr>
<tr>
<td>320/250G</td>
<td>H5py I/O Cost: 2.5</td>
</tr>
<tr>
<td></td>
<td>HDF5 I/O Cost: 2.3</td>
</tr>
<tr>
<td>640/500G</td>
<td>H5py I/O Cost: 2.5</td>
</tr>
<tr>
<td></td>
<td>HDF5 I/O Cost: 2.3</td>
</tr>
<tr>
<td>1600/1.3T</td>
<td>H5py I/O Cost: 2.5</td>
</tr>
<tr>
<td></td>
<td>HDF5 I/O Cost: 2.3</td>
</tr>
</tbody>
</table>

**Strong Scaling**

97.1%

**Weak Scaling**

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

<table>
<thead>
<tr>
<th></th>
<th>Single Node</th>
<th>Multi-nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Metadata</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1k File Creation</td>
<td>63.8%</td>
<td></td>
</tr>
<tr>
<td>1k Object Scanning</td>
<td>60.0%</td>
<td></td>
</tr>
<tr>
<td><strong>Independent I/O</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weak Scaling</td>
<td>97.8%</td>
<td>100%</td>
</tr>
<tr>
<td>Strong Scaling</td>
<td>100%</td>
<td>97.1%</td>
</tr>
<tr>
<td><strong>Collective I/O</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weak Scaling</td>
<td>100%</td>
<td>90%</td>
</tr>
<tr>
<td>Strong Scaling</td>
<td>98.6%</td>
<td>87%</td>
</tr>
</tbody>
</table>
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

Remi Lehe, LBNL, GTC 2017
Case Study I: Warp

- Laser-wakefield

500 meters $\rightarrow$ 9 cm

**Conventional accelerators**

$\sim 0.02$ GeV/m

e.g. 8 GeV over $\sim 500$ m of RF cavities

**Laser-wakefield accelerators**

$\sim 50$ GeV/m

e.g. 4 GeV over 9 cm of gas

Remi Lehe, LBNL, GTC 2017
Case Study I: Warp IO with H5py

Simulation box

<table>
<thead>
<tr>
<th>proc0</th>
<th>proc5</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc1</td>
<td>...</td>
</tr>
<tr>
<td>proc2</td>
<td></td>
</tr>
<tr>
<td>proc3</td>
<td></td>
</tr>
<tr>
<td>proc4</td>
<td></td>
</tr>
</tbody>
</table>

HDF5 files

iteration 0

Perform
~100 iterations of
the solver

Simulation box

<table>
<thead>
<tr>
<th>proc0</th>
<th>proc5</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc1</td>
<td>...</td>
</tr>
<tr>
<td>proc2</td>
<td></td>
</tr>
<tr>
<td>proc3</td>
<td></td>
</tr>
<tr>
<td>proc4</td>
<td></td>
</tr>
</tbody>
</table>

HDF5 files

iteration 1
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

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

32 bit

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

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:
  - Reformating 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*
User `cat1` is looking for 1 million fibers

<table>
<thead>
<tr>
<th>plate</th>
<th>mjd</th>
<th>fiber</th>
</tr>
</thead>
<tbody>
<tr>
<td>4322</td>
<td>55532</td>
<td>32</td>
</tr>
<tr>
<td>5892</td>
<td>55598</td>
<td>25</td>
</tr>
<tr>
<td>6298</td>
<td>55133</td>
<td>11</td>
</tr>
</tbody>
</table>

Case Study II: H5Boss IO

- Read
- Converted Fits Files in H5
- Write
- Subset File
- Data Analysis
- Post Analysis
- Share

H5boss.subset

Query
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

Write
Subset File

e.g., Groups, Datasets

Post Analysis
Share
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
   - Sender
     - Pickling
       - Dispatching
       - Allocation
       - Packing
   - Receiver
     - Unpickling
       - Allocation
       - Translation
       - Unpacking
Re-design the key, value pair to be buffer like

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

<table>
<thead>
<tr>
<th>K:</th>
<th>V:</th>
</tr>
</thead>
<tbody>
<tr>
<td>b['3666/55159/599/coadd']</td>
<td>(((((WAVE, '&lt;f4'), (FLUX, '&lt;f4'), (IVAR, '&lt;f4'), (AND_MAKS, '&lt;i4'), (OR_MASK, '&lt;i4'), ('WAVEDISP', '&lt;f4'), (SKY, '&lt;f4'), (MODEL, '&lt;f4'))), (4619,), '/global/cscratch1/sd/jialin/h5boss/3666-55159.hdf5')</td>
</tr>
</tbody>
</table>

Key: Path to HDF5 dataset
Value: shape

<table>
<thead>
<tr>
<th>K:</th>
<th>V:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(str) &quot;3666/55159/599/coadd&quot;</td>
<td>(int) &quot;4619&quot;</td>
</tr>
</tbody>
</table>
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
    - ...

Ø H5py: Productivity
Ø H5py: Performance
1. Optimal HDF5 file creation

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

Choose the most modern format [optional]
2. Use low-level API in H5py

```python
1. space = h5py.h5s.create_simple((100,))
2. plist = h5py.h5p.create(h5py.h5p.DATASET_CREATE)
3. 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

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
High Performance H5py with Sample Codes

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

Thanks