boost-histogram and hist

Henry Schreiner

April 15, 2019
Histograms in Python
Current state of histograms in Python

Core library: numpy
- Historically slow
- No histogram object
- Plotting is separate

Other libraries
- Narrow focus: speed, plotting, or language
- Many are abandoned
- Poor design, backends, distribution
What is needed?

Histograms in Python

Design
- A histogram should be an object
- Manipulation and plotting should be easy

Performance
- Fast single threaded filling
- Multithreaded filling (since it’s 2019)

Flexibility
- Axes options: sparse, growing, labels
- Storage: integers, weights, errors...

Distribution
- Easy to use anywhere, pip or conda
- Should have wheels, be easy to build, etc.
Future of histograms in Python

Core histogramming libraries

Universal adaptor

Front ends (plotting, etc)

boost-histogram

ROOT

Aghast

hist

mpl-hep

physt

others

Henry Schreiner

April 15, 2019
Boost::Histogram (C++14)
Intro to Boost::Histogram

- Multidimensional templated header-only histogram library: [boostorg/histogram](https://boostorg/histogram)
- Designed by Hans Dembinski, inspired by ROOT, GSL, and histbook

### Histogram
- Axes
- Storages
- Accumulators

### Axes types
- Regular, Circular
- Variable
- Integer
- Category

### Diagram
- Regular axis
- Storage (Static Dynamic)
- Optional underflow
- Optional overflow
- Regular axis with log transform
- Accumulator int, double, unlimited, ...

---

Henry Schreiner

April 15, 2019
Intro to Boost::Histogram

- Multidimensional templated header-only histogram library: /boostorg/histogram
- Designed by Hans Dembinski, inspired by ROOT, GSL, and histbook
boost-histogram (Python)
Intro to the Python bindings

• Boost::Histogram developed with Python in mind
• Original bindings based on Boost::Python
  ▶ Hard to build and distribute
  ▶ Somewhat limited
• New bindings: /scikit-hep/boost-histogram
  ▶ 0-dependency build (C++14 only)
  ▶ State-of-the-art PyBind11
Design

- Supports Python 2.7 and 3.4+
- 260+ unit tests run on Azure on Linux, macOS, and Windows
- Up to 16 axes supported (may go up or down)
- 1D, 2D, and ND histograms all have the same interface

Tries to stay close to the original Boost::Histogram where possible.

**C++**
```cpp
#include <boost/histogram.hpp>
namespace bh = boost::histogram;

auto hist = bh::make_histogram(
    bh::axis::regular<>{2, 0, 1, "x"},
    bh::axis::regular<>{4, 0, 1, "y"});

hist(.2, .3);
```

**Python**
```python
import boost.histogram as bh

hist = bh.make_histogram(
    bh.axis.regular(2, 0, 1, metadata="x"),
    bh.axis.regular(4, 0, 1, metadata="y"))

hist(.2, .3)
```
Design: Manipulations

**Combine** two histograms
```
hist1 + hist2
```

**Scale** a histogram
```
hist * 2.0
```

**Project** a 3D histogram to 2D
```
hist.project(0,1) # select axis
```

**Sum** a histogram contents
```
hist.sum()
```

**Access** an axis
```
axis0 = hist.axis(0)
axis0.edges() # The edges array
axis0.bin(1) # The bin accessors
```

**Fill** 2D histogram with values or arrays
```
hist(x, y)
```

**Fill** copies in 4 threads, then merge
```
hist.fill_threaded(4, x, y)
```

**Fill** in 4 threads (atomic storage only)
```
hist.fill_atomic(4, x, y)
```

**Convert** to Numpy, 0-copy
```
hist.view()
# Or
np.asarray(hist)
```
Flexibility: Axis

- `bh.axis.regular`
  - `bh.axis.regular_uoflow`
  - `bh.axis.regular_noflow`
  - `bh.axis.regular_growth`
- `bh.axis.circular`
- `bh.axis.regular_log`
- `bh.axis.regular_sqrt`
- `bh.axis.regular_pow`
- `bh.axis.integer`
- `bh.axis.integer_noflow`
- `bh.axis.integer_growth`
- `bh.axis.variable`
- `bh.axis.category_int`
- `bh.axis.category_int_growth`
Flexibility: Storage types

- `bh.storage.int`
- `bh.storage.double`
- `bh.storage.unlimited (WIP)`
- `bh.storage.atomic_int`
- `bh.storage.weight (WIP)`
- `bh.storage.profile (WIP, needs sampled fill)`
- `bh.storage.weighted_profile (WIP, needs sampled fill)`
The following measurements are with:

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Details</th>
</tr>
</thead>
</table>
| **1D**    | 100 regular bins  
            | 10,000,000 entries |
| **2D**    | 100x100 regular bins  
            | 1,000,000 entries |

See my [histogram performance post](#) for measurements of other libraries.
### Performance: macOS, dual core, 1D

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Fill</th>
<th>Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numpy</td>
<td>uint64</td>
<td></td>
<td>149.4 ms</td>
<td>1x</td>
</tr>
<tr>
<td>Any</td>
<td>int</td>
<td></td>
<td>236 ms</td>
<td>0.63x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>1</td>
<td>86.23 ms</td>
<td>1.7x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint 1</td>
<td></td>
<td>132 ms</td>
<td>1.1x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint 2</td>
<td></td>
<td>168.2 ms</td>
<td>0.89x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint 4</td>
<td></td>
<td>143.6 ms</td>
<td>1x</td>
</tr>
<tr>
<td>Regular</td>
<td>int 1</td>
<td></td>
<td>84.75 ms</td>
<td>1.8x</td>
</tr>
<tr>
<td>Regular</td>
<td>int 2</td>
<td></td>
<td>51.6 ms</td>
<td>2.9x</td>
</tr>
<tr>
<td>Regular</td>
<td>int 4</td>
<td></td>
<td>42.39 ms</td>
<td>3.5x</td>
</tr>
</tbody>
</table>
### Performance: CentOS7, 24 core, 1D (anaconda)

**boost-histogram** (Python)

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Fill</th>
<th>Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numpy</td>
<td>uint64</td>
<td></td>
<td>121 ms</td>
<td>1x</td>
</tr>
<tr>
<td>Any</td>
<td>int</td>
<td></td>
<td>261.5 ms</td>
<td>0.46x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td></td>
<td>142.2 ms</td>
<td>0.85x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>1</td>
<td>319.1 ms</td>
<td>0.38x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>48</td>
<td>272.9 ms</td>
<td>0.44x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>1</td>
<td>243.4 ms</td>
<td>0.5x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>6</td>
<td>94.76 ms</td>
<td>1.3x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>12</td>
<td>71.38 ms</td>
<td>1.7x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>24</td>
<td>52.26 ms</td>
<td>2.3x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>48</td>
<td>43.01 ms</td>
<td>2.8x</td>
</tr>
</tbody>
</table>

April 15, 2019

Henry Schreiner
### Performance: KNL, 64 core, 1D (anaconda)

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Fill</th>
<th>Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numpy</td>
<td>uint64</td>
<td></td>
<td>716.9 ms</td>
<td>1x</td>
</tr>
<tr>
<td>Any</td>
<td>int</td>
<td></td>
<td>1418 ms</td>
<td>0.51x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td></td>
<td>824 ms</td>
<td>0.87x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>1</td>
<td>871.7 ms</td>
<td>0.82x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>4</td>
<td>437.1 ms</td>
<td>1.6x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>64</td>
<td>198.8 ms</td>
<td>3.6x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>128</td>
<td>186.8 ms</td>
<td>3.8x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>256</td>
<td>195.2 ms</td>
<td>3.7x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>1</td>
<td>796.9 ms</td>
<td>0.9x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>2</td>
<td>430.6 ms</td>
<td>1.7x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>4</td>
<td>247.6 ms</td>
<td>2.9x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>64</td>
<td>88.77 ms</td>
<td>8.1x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>128</td>
<td>98.08 ms</td>
<td>7.3x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>256</td>
<td>112.2 ms</td>
<td>6.4x</td>
</tr>
</tbody>
</table>
## Performance: macOS, dual core, 2D

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Fill</th>
<th>Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numpy</td>
<td>uint64</td>
<td></td>
<td>121.1 ms</td>
<td>1x</td>
</tr>
<tr>
<td>Any</td>
<td>int</td>
<td></td>
<td>37.12 ms</td>
<td>3.3x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td></td>
<td>18.5 ms</td>
<td>6.5x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint 1</td>
<td></td>
<td>20.21 ms</td>
<td>6x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint 2</td>
<td></td>
<td>14.17 ms</td>
<td>8.5x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint 4</td>
<td></td>
<td>10.23 ms</td>
<td>12x</td>
</tr>
<tr>
<td>Regular</td>
<td>int 1</td>
<td></td>
<td>17.86 ms</td>
<td>6.8x</td>
</tr>
<tr>
<td>Regular</td>
<td>int 2</td>
<td></td>
<td>9.41 ms</td>
<td>13x</td>
</tr>
<tr>
<td>Regular</td>
<td>int 4</td>
<td></td>
<td>6.854 ms</td>
<td>18x</td>
</tr>
</tbody>
</table>
Performance: CentOS7, 24 core, 2D (anaconda)

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Fill</th>
<th>Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numpy</td>
<td>uint64</td>
<td></td>
<td>87.27 ms</td>
<td>1x</td>
</tr>
<tr>
<td>Any</td>
<td>int</td>
<td></td>
<td>41.42 ms</td>
<td>2.1x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td></td>
<td>21.67 ms</td>
<td>4x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>1</td>
<td>38.61 ms</td>
<td>2.3x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>6</td>
<td>19.89 ms</td>
<td>4.4x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>24</td>
<td>9.556 ms</td>
<td>9.1x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>48</td>
<td>8.518 ms</td>
<td>10x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>1</td>
<td>36.5 ms</td>
<td>2.4x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>6</td>
<td>8.976 ms</td>
<td>9.7x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>12</td>
<td>5.318 ms</td>
<td>16x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>24</td>
<td>4.388 ms</td>
<td>20x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>48</td>
<td>5.839 ms</td>
<td>15x</td>
</tr>
</tbody>
</table>
## Performance: KNL, 64 core, 2D (anaconda)

**boost-histogram (Python)**

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Fill</th>
<th>Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numpy</td>
<td>uint64</td>
<td></td>
<td>439.5 ms</td>
<td>1x</td>
</tr>
<tr>
<td>Any</td>
<td>int</td>
<td></td>
<td>250.6 ms</td>
<td>1.8x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td></td>
<td>135.6 ms</td>
<td>3.2x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>1</td>
<td>142.2 ms</td>
<td>3.1x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>4</td>
<td>52.71 ms</td>
<td>8.3x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>32</td>
<td>12.05 ms</td>
<td>36x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>64</td>
<td>16.5 ms</td>
<td>27x</td>
</tr>
<tr>
<td>Regular</td>
<td>aint</td>
<td>256</td>
<td>43.93 ms</td>
<td>10x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>1</td>
<td>141.1 ms</td>
<td>3.1x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>2</td>
<td>70.78 ms</td>
<td>6.2x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>4</td>
<td>36.11 ms</td>
<td>12x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>64</td>
<td>18.93 ms</td>
<td>23x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>128</td>
<td>36.09 ms</td>
<td>12x</td>
</tr>
<tr>
<td>Regular</td>
<td>int</td>
<td>256</td>
<td>55.64 ms</td>
<td>7.9x</td>
</tr>
</tbody>
</table>
Performance: Summary

boost-histogram (Python)

<table>
<thead>
<tr>
<th>System</th>
<th>1D max speedup</th>
<th>2D max speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>macOS 1 core</td>
<td>1.7 x</td>
<td>6.5 x</td>
</tr>
<tr>
<td>macOS 2 core</td>
<td>3.5 x</td>
<td>18 x</td>
</tr>
<tr>
<td>Linux 1 core</td>
<td>0.85 x</td>
<td>4 x</td>
</tr>
<tr>
<td>Linux 24 core</td>
<td>2.8 x</td>
<td>20 x</td>
</tr>
<tr>
<td>KNL 1 core</td>
<td>0.87 x</td>
<td>3.2 x</td>
</tr>
<tr>
<td>KNL 64 core</td>
<td>8.1 x</td>
<td>36 x</td>
</tr>
</tbody>
</table>

- Note that Numpy 1D is well optimized (last few versions)
- Anaconda versions may provide a few more optimizations to Numpy
- Mixing axes types in boost-histogram can reduce performance by 2-3x
Distribution

- We must provide excellent distribution.
  - If anyone writes `pip install boost-histogram` and it fails, we have failed.
- Docker ManyLinux1 GCC 8.3: 

Wheels

- manylinux1 32, 64 bit (ready)
- manylinux2010 64 bit (planned)
- macOS 10.9+ (wip)
- Windows 32, 64 bit, Python 3.6+ (wip)
  - Is Python 2.7 Windows needed?

Source

- SDist (ready)
- Build directly from GitHub (done)

Conda

- conda package (planned, easy)

```bash
python -m pip install \
  git+https://github.com/scikit-hep/boost-histogram.git@develop
```
Plans

- Add shortcuts for axis types, fill out axis types
- Allow view access into unlimited storage histograms
- Add `from_numpy` and numpy style shortcut(s)
- Filling
  - Samples
  - Weights
  - Non-numerical fill (if possible)
- Add profile, weighted_profile histograms
- Add reduce operations
- Release to PyPI
- Add some docs and read the docs support

First alpha

Release planned this week
Bikeshedding (API discussion)

Let’s discuss API! (On GitHub issues or gitter)

- Download: pip install boost-histogram (WIP)
- Use: import boost.histogram as bh
- Create: hist = 
  bh.histogram(bh.axis.regular(12,0,1))
- Fill: hist(values)
- Access values, convert to numpy, etc.

Documentation

- The documentation will also need useful examples, feel free to contribute!
hist
hist is the ‘wrapper’ piece that does plotting and interacts with the rest of the ecosystem.

**Plans**

- Easy plotting adaptors (mpl-hep)
- Serialization formats (ROOT, HDF5)
- Auto-multithreading
- Statistical functions (Like TEfficiency)
- Multihistograms (HistBook)
- Interaction with fitters (ZFit, GooFit, etc)
- Bayesian Blocks algorithm from SciKit-HEP
- Command line histograms for stream of numbers

**Call for contributions**

- What do you need?
- What do you want?
- What would you like?

Join in the development! This should combine the best features of other packages.
Questions?
• Supported by IRIS-HEP, NSF OAC-1836650