Max-Planck-Institut für Kernphysik

C++11 histogram library for Boost

Hans Dembinski[1]
[1]MPIK Heidelberg, Germany

ROOT Users‘ workshop, Sarajevo, Sep 2018

(boost).histogram in a nutshell

- Multi-dimensional histogram in C++
  - Python bindings included, based on boost.python
    - Numpy & pickle support
    - Faster than numpy.histogram for N-dim >= 2
  - Header-only without Python bindings
  - Feature set based on ROOT histograms & GSL histograms
- Source: https://github.com/hdembinski/histogram
- Docs: http://hdembinski.github.io/histogram/doc/html
- Thoroughly unit-tested: line coverage **99.94 %**
- Selected features
  - Automatic memory efficient handling of bin counters
  - No-overflow guarantee
  - Supports many axis types, e.g. a circular axis
  - Supports weighted increments
  - **Faster** than other libs in benchmarks
    (depends on configuration, compiler & machine)
- Submitted for Boost review: Sep 17-27

Why Boost C++ libraries?

- Free peer-reviewed portable C++ source libraries
- Greatly extends functionality of C++ stdlib
- Popular in science and industry
- Often first step towards C++ standardization
- Existing sub-libraries with statistics tools
  - Accumulators
  - Math
  - Random (superset of std::random)

- Histogram: basic data structure in statistics software
  - Provide standard solution to stop reinvention of the wheel
  - Solution must be useful for everyone
  - Solution must be customizable and fast (policy-based design)

Policy-based design

a variant of
static polymorphism

histogram<*typename* **Axes**, *typename* **Storage**>

- Host class
- Defines public n-dimensional interface
- Converts n-dimensional index to internal consecutive counter address

Options for **Axes = Sequence of axis types**

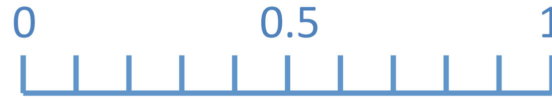
- Static sequence **std::tuple<...>**
  - When number and axis types are known at compile-time
  - Very fast execution speed
- Dynamic sequence **std::vector<axis::any<...>>**
  - When number and axis types are only known at run-time
  - Reduced code execution speed (about a factor 2)
  - Python-bindings require this

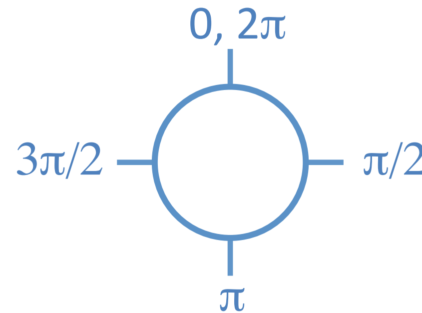
Options for **Storage**

- Static counters **array_storage<T>**
  - Full control over counter type, but...
  - Choice of T may not be safe/efficient and difficult to predict before seeing the data
- Dynamic counters **adaptive_storage**
  - Cannot overflow
  - Adaptive memory consumption
  - Runtime cost over-compensated by better utilization of CPU cache
- *Add your own, e.g. mmap'd file, stack-based buffer, ...*

Built-in axis types

- regular
  regular<>(10, 0, 1)
  regular<transform::log>(10, 1, 1e5)
  ...





- circular
  circular<>(4)

- variable
  variable<>({0.1, 0.3, 0.9}) : 2 bins [0.1, 0.3], [0.3, 0.9]

- integer
  integer<>(3, 6) : 3 bins [3,4], [4,5], [5,6]

- category
  category<std::string>({“red”, “blue”}) : 2 bins “red” and “blue”

Add your own!

C++ example

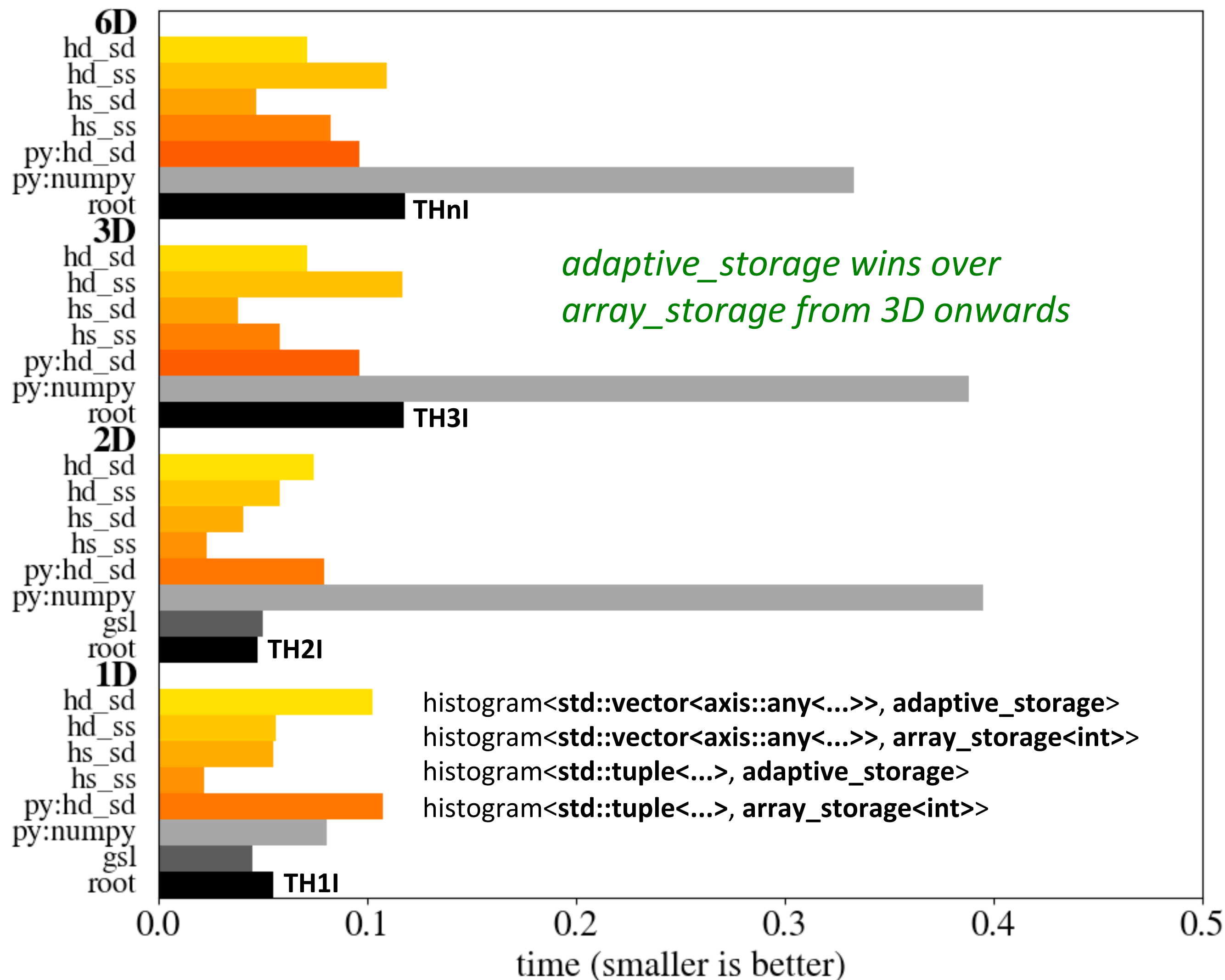
```
#include <boost/histogram.hpp> // all-included-header

int main() {
  namespace bh = boost::histogram;
  using namespace bh::literals; // enables _c suffix

  auto h = bh::make_static_histogram( bh::axis::regular<>(6, -1.0, 2.0, "x") );

  auto data = { -0.4, 1.1, 0.3, 1.7 };
  std::for_each(data.begin(), data.end(), h);

  for (auto it = h.begin(); it != h.end(); ++it) {
    const auto bin = it.bin(0_c);
    std::cout << "bin " << it.idx(0) << " x in [" << bin.lower() << ", " << bin.upper() << "): ”
                << it->value() << " +/- " << std::sqrt(it->variance()) << std::endl;
  }
}
/* program output: (note that under- and overflow bins appear at the end)
bin 0 x in [-1.0, -0.5): 0 +/- 0
...
bin 5 x in [ 1.5, 2.0): 1 +/- 1
bin 6 x in [ 2.0, inf): 0 +/- 0
bin -1 x in [-inf, -1): 0 +/- 0 */
```

Python example

```
import histogram as bh
import numpy as np
h = bh.histogram(
   bh.axis.regular(10, 0.0, 5.0, "radius", uoflow=False),
   bh.axis.circular(4, 0.0, 2 * np.pi, "phi")
)
x = np.random.randn(1000) # generate x
y = np.random.randn(1000) # generate y
radius = (x ** 2 + y ** 2) ** 0.5
phi = np.arctan2(y, x)

h(radius, phi)

count_matrix = np.asarray(h) # access histogram counts (no copy)

print(count_matrix)
# program output:
# [[37 26 33 37]
#  [60 69 76 62]
#  ...
#  [ 0 1 0 0]
#  [ 0 0 0 0]]
```

Benchmarks

2.9 GHz Macbook Pro, 1 million bins placed along 1, 2, 3, and 6 dimensions



Summary and Outlook

- (boost).histogram
  - Header-only C++11, only Boost as dependency
  - Python module with numpy support as shared library
  - Source: https://github.com/hdembinski/histogram
  - Docs: http://hdembinski.github.io/histogram/doc/html
  - Many parallels with ROOT::Experimental::RHist
  - Official Boost review Sep 17 to 27
  - Teamed up with Jim Pivarski to integrate histbook features https://github.com/scikit-hep/histbook
    - Compute histograms and profiles in a unified way
    - More options for parallelization