run-time speed Boost histograms vs ROOT histograms

Bhawani Singh, Thomas Klemenz, Stefan Heckel

Quality Control

Technische Universität München

16.09.2021

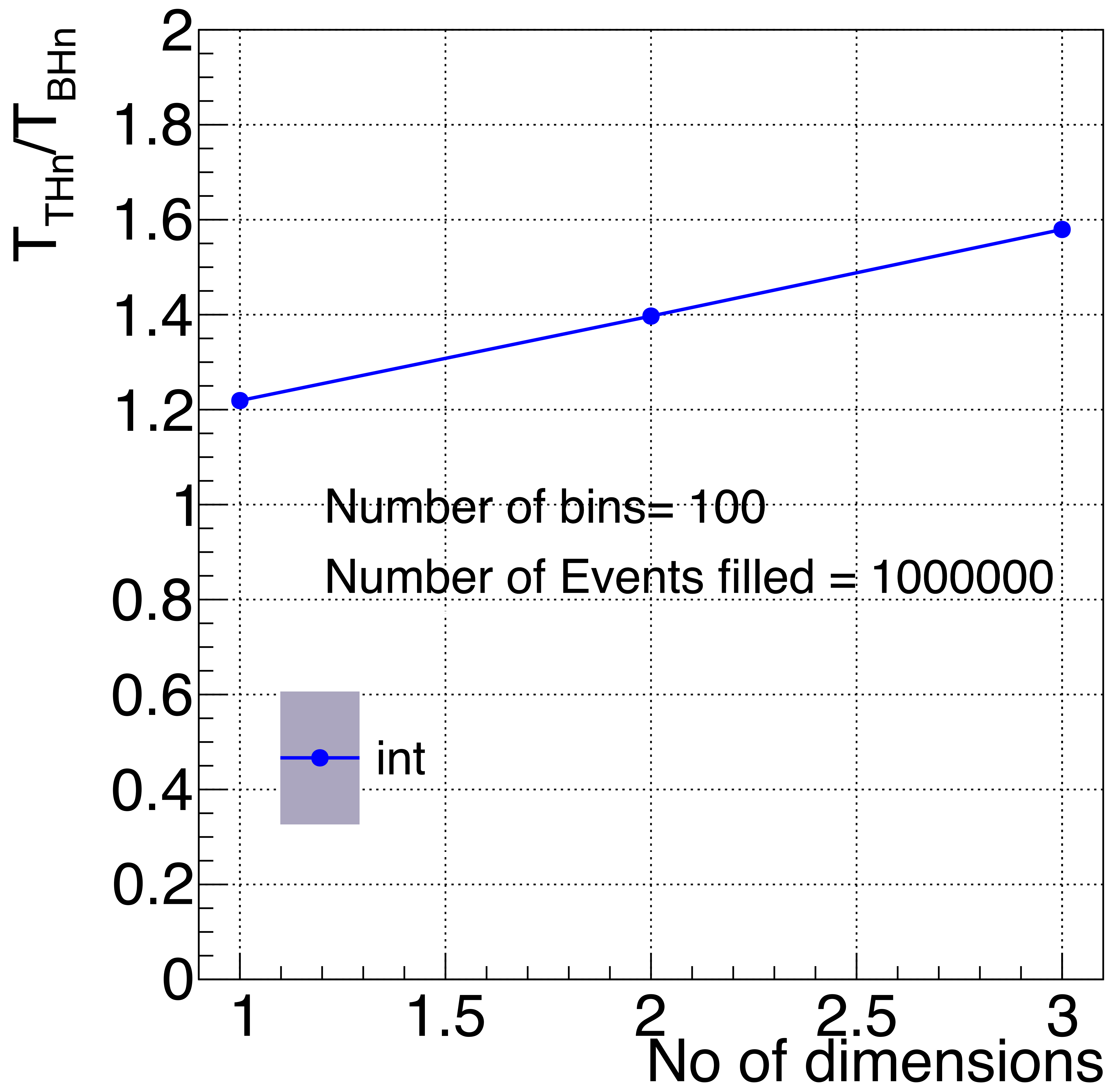
ALICE

Until now: boost library faster than ROOT

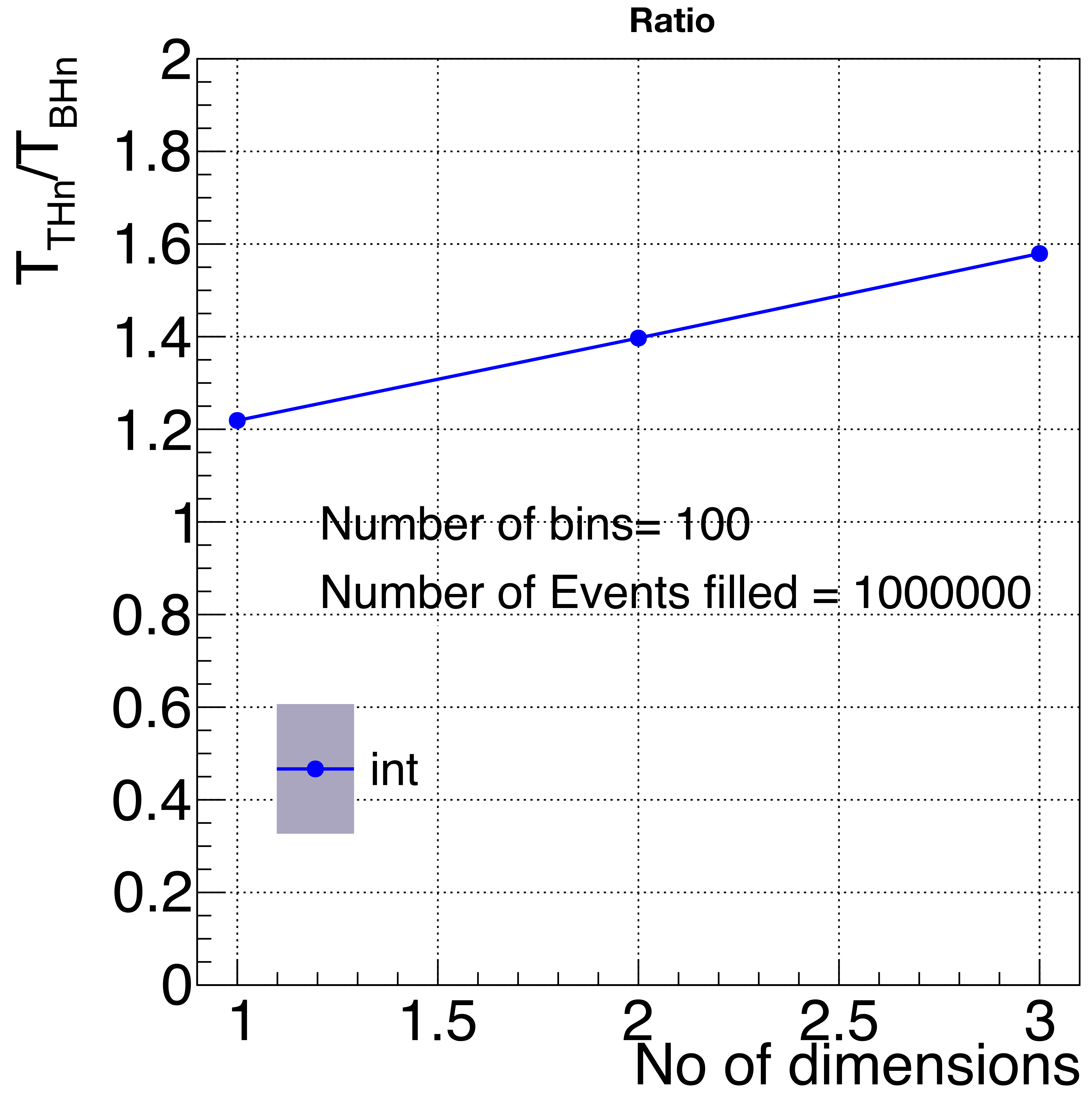
BHn vs TH1I,TH2I and TH3I

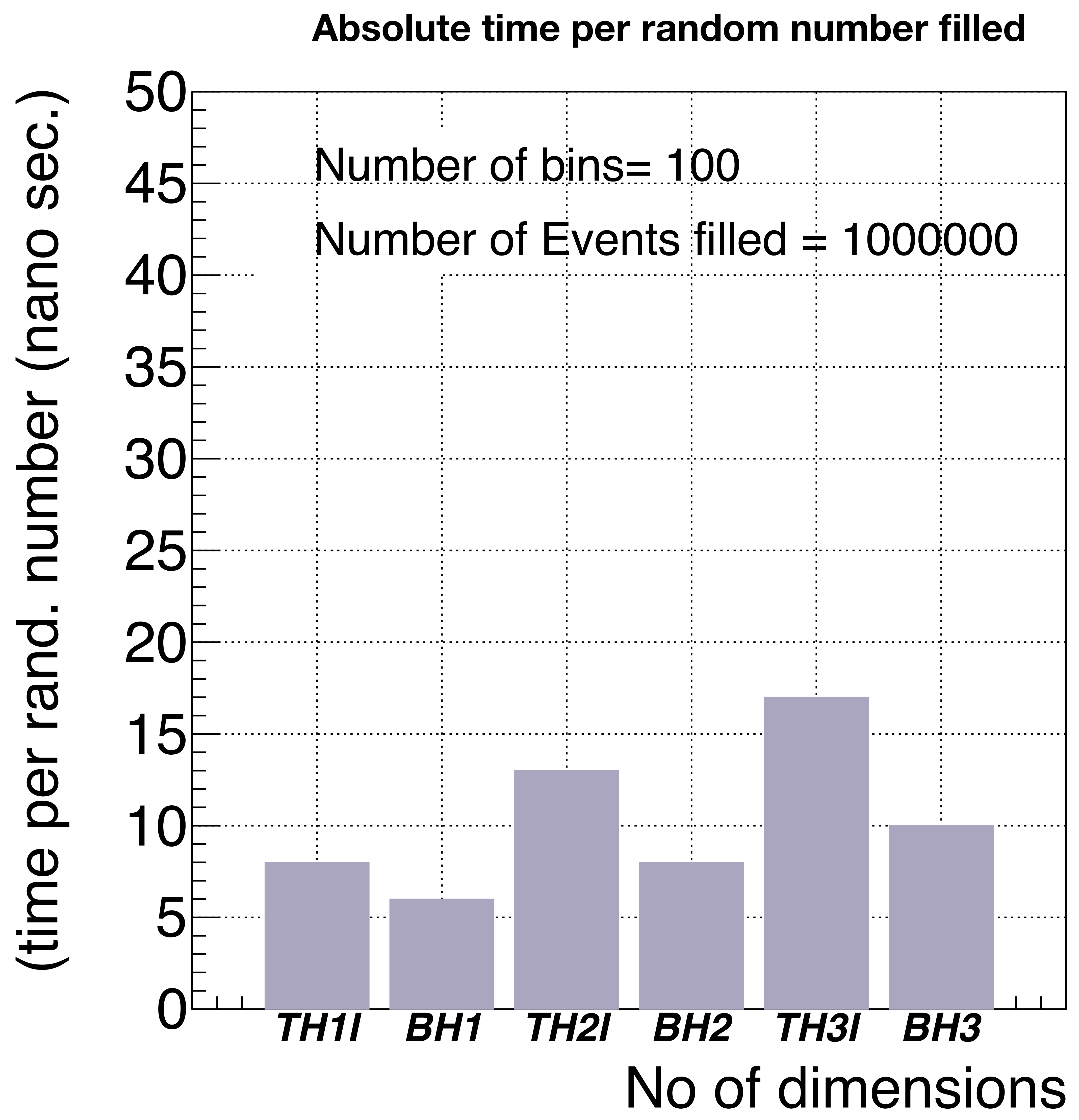
- Total Events filled = 1 **million** in each dimension
- # of bins in range: **100** in {0, 10}
- Uniform distribution of ints in {0,10}

Boost histograms
Faster



BHn vs TH1I,TH2I and TH3I

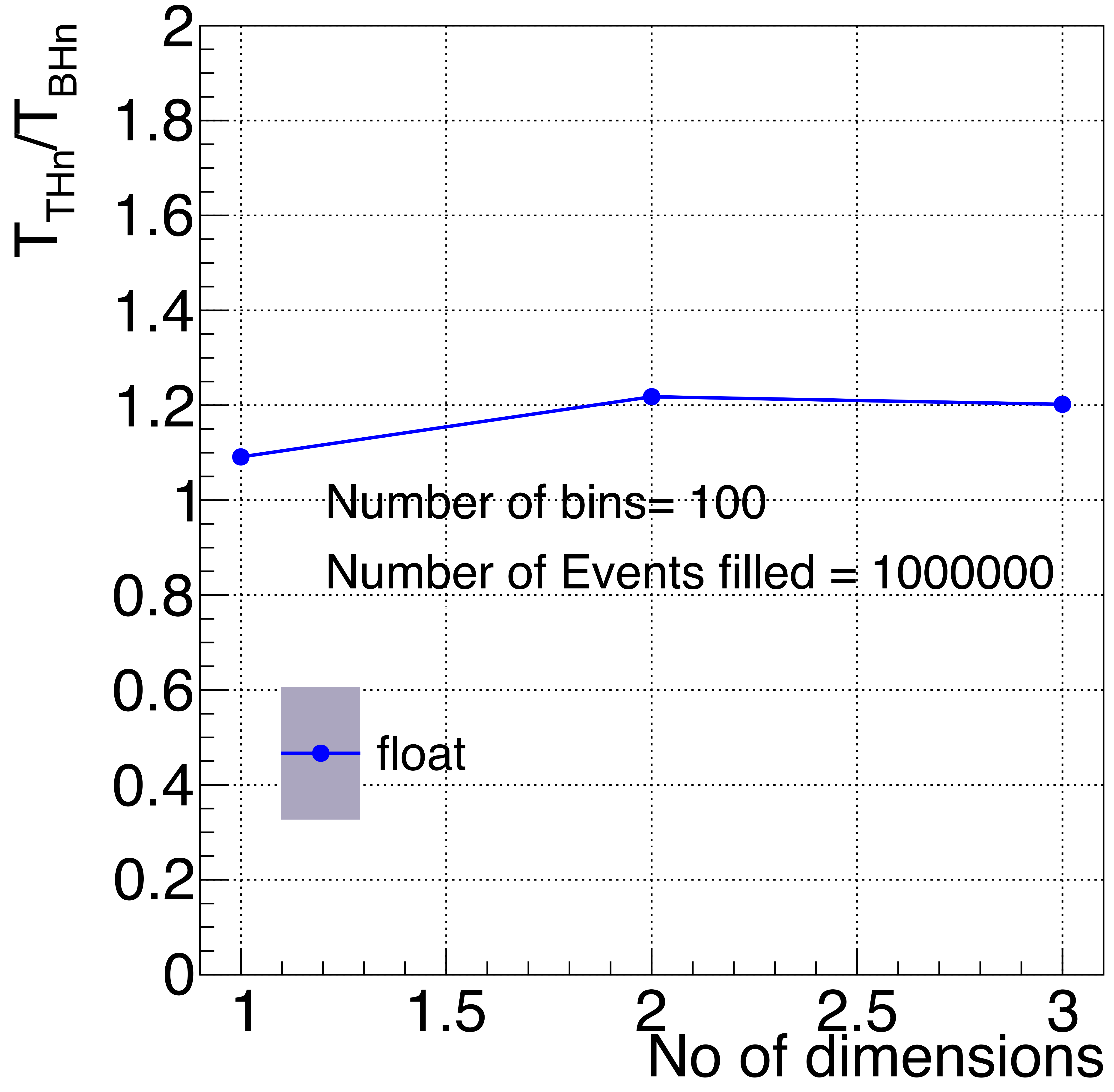




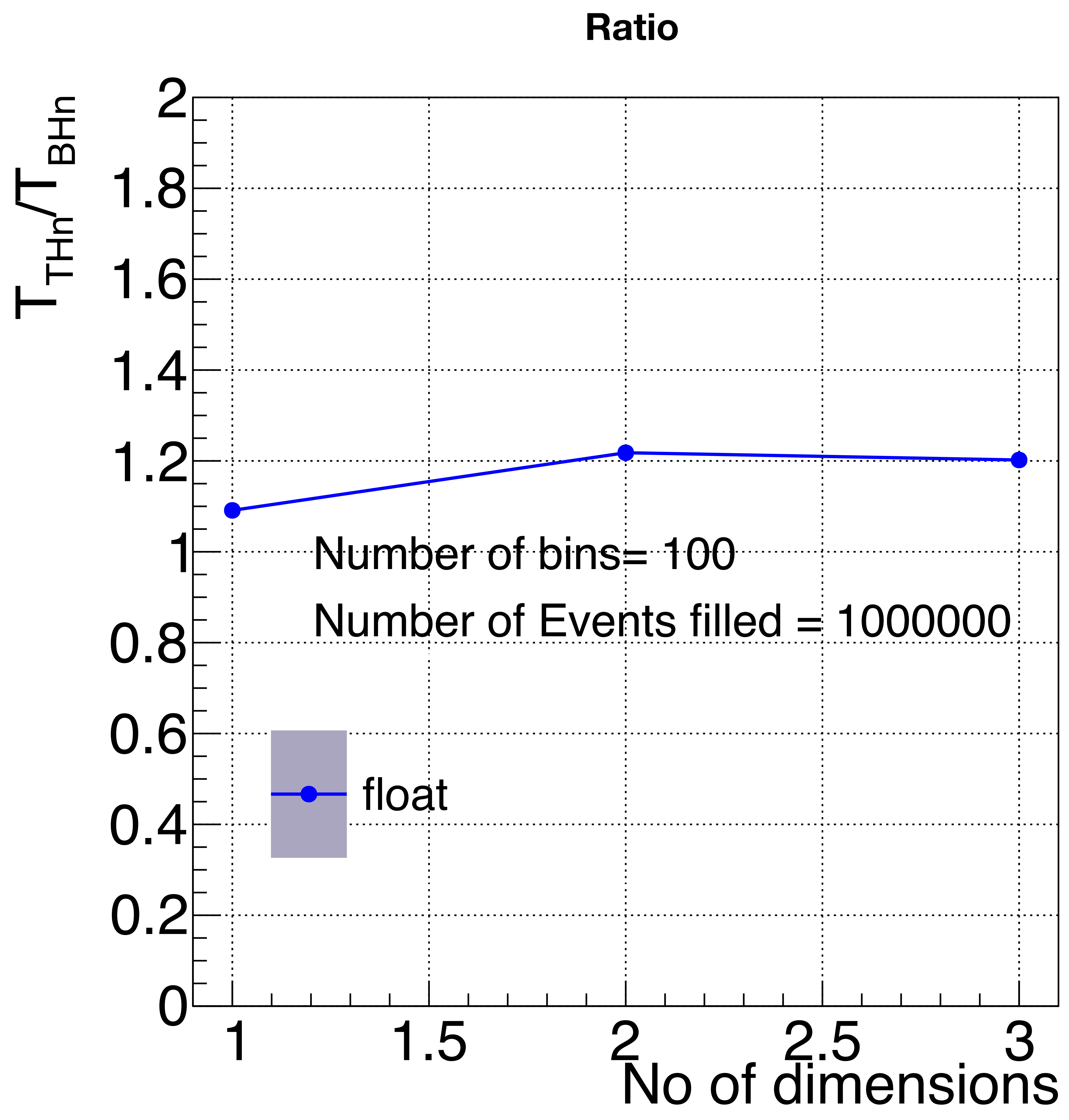
BHn vs TH1F, TH2F and TH3F

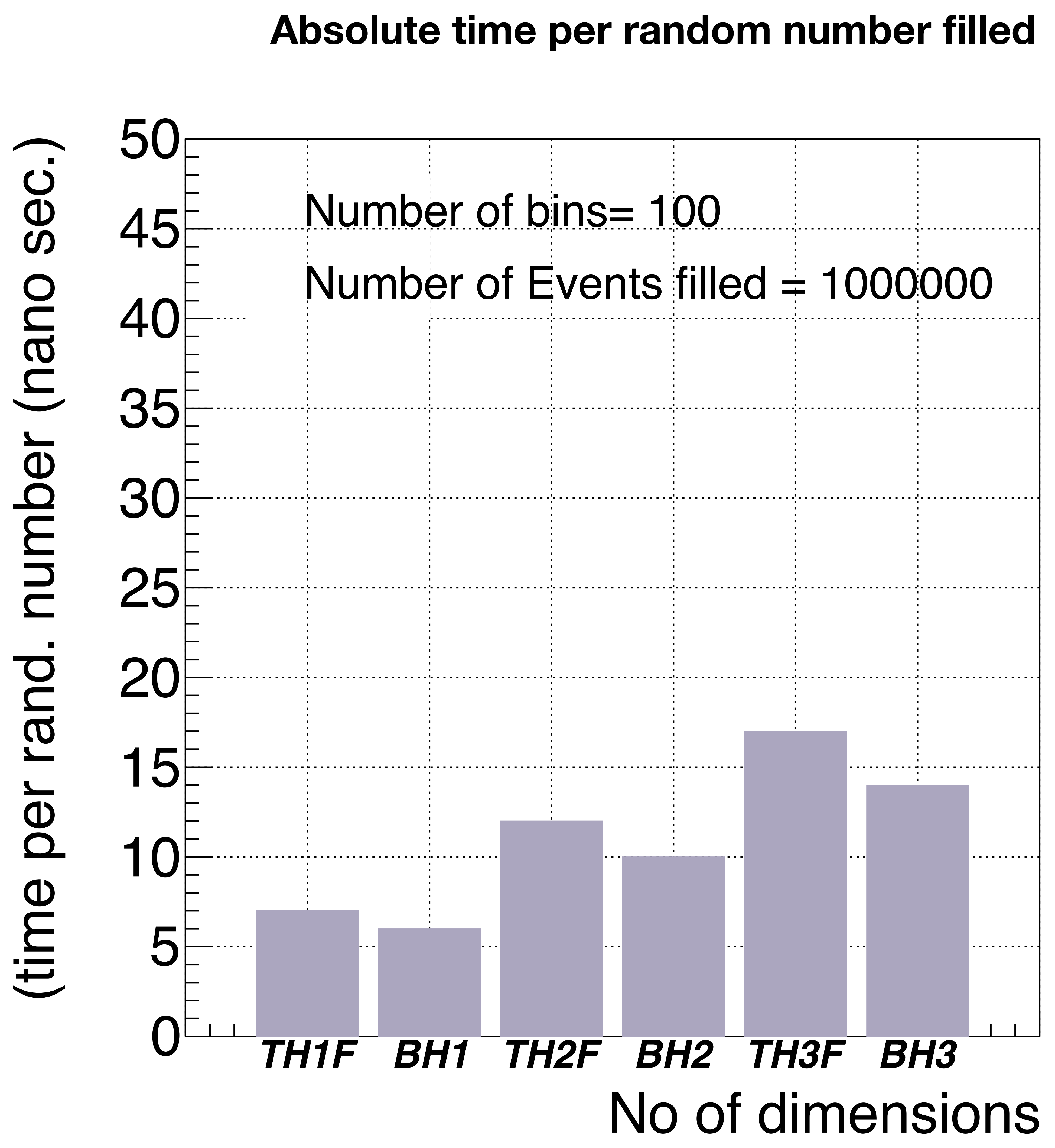
- Total Events filled = 1 **million** in each dimension
- # of bins in range: **100** in {0, 1.0}
- Uniform distribution of floats in {0,1.0}

Boost histograms Faster



BHn vs TH1F,TH2F and TH3F

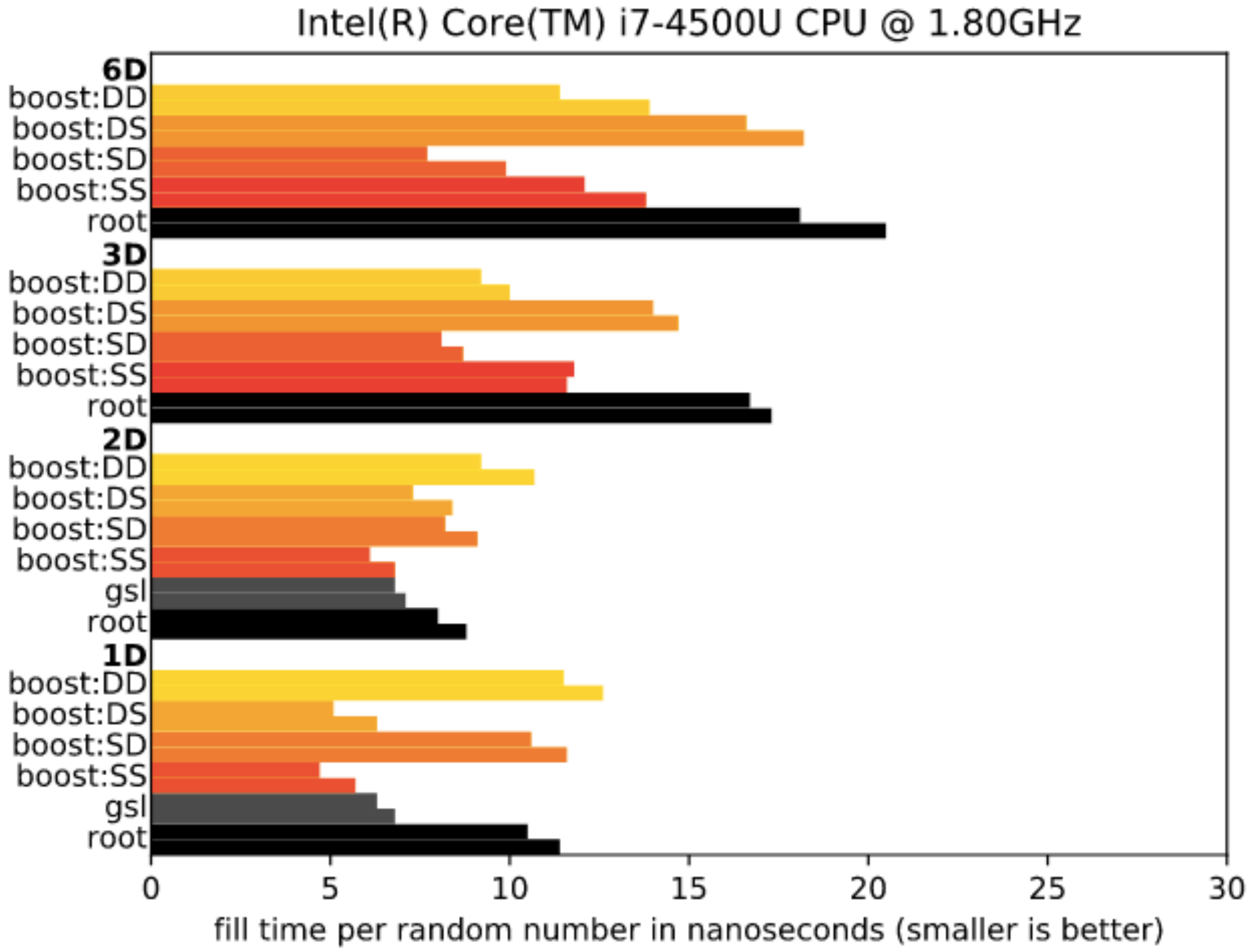




Benchmark check done by authors

- Boost.Histogram are compared with histogram classes from root library
- ROOT classes (`TH1I` for 1D, `TH2I` for 2D, `TH3I` for 3D and `THnI` for 6D)
- Boost classes
  - boost:SS Histogram with std::tuple<axis::regular<>> and std::vector<int>
  - boost:SD Histogram with std::tuple<axis::regular<>> with boost::histogram::unlimited_storage
  - boost:DS Histogram with std::vector<axis::variant<axis::regular<>>> with std::vector<int>
  - boost:DD Histogram with std::vector<axis::variant<axis::regular<>>> with boost::histogram::unlimited_storage

Fill performance



https://www.boost.org/doc/libs/1_70_0/libs/histogram/doc/html/histogram/benchmarks.html

now: boost library slows down when increasing number histograms in the macro

Strange behaviour of clock due to optimisation

- TimeFunction( ){ :

```
        Create 1Dim BHn object;

        Create TH1F object;
        start clock_BHn;
        fill (){
        boosthisto(val);
        }
        stop clock_BHn;
        start clock_THn;
        fill (){
        TH1thisto(val);
        }
        stop clock_THn
        return ratio of time;
    }
    Anotherfunction(){
    do something in the code: where you can have another histogram/BHn
    }
```

```
call in main(){
TimeFunction( );
}
```

Strange behaviour of clock due to optimisation

- TimeFunction( ){ :

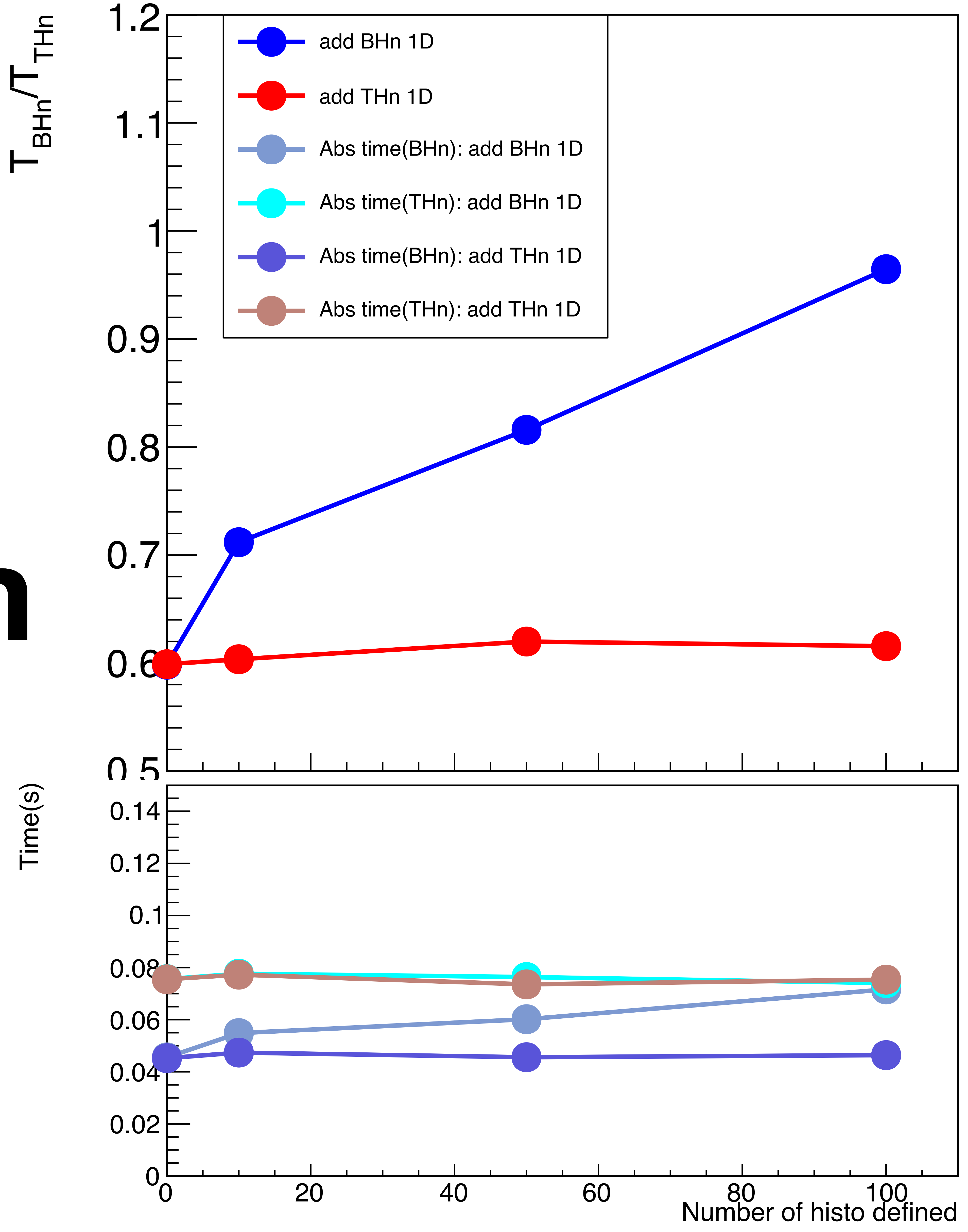
```
        Create 1Dim BHn object;
        Create TH1F object;
        Add more BHn or THn but don't fill, this changes the clock results

        start clock_BHn;
        fill (){
        boosthisto(val);
        }
        stop clock_BHn;
        start clock_THn;
        fill (){
        TH1thisto(val);
        }
        stop clock_THn
        return ratio of time;
    }
    Anotherfunction(){
    do something in the code: where you can have another histogram/BHn
    }

  call in main(){
  TimeFunction( );
  }
```

Add more boost histograms/root histogram

- Total Events filled = 10 **million** in each dimension
- # of bins in range: **100** in {0, 10}
- Uniform distribution of ints in {0,1.0}

Boost histograms
Faster but slows down as increasing number of BHn in the macro

1 Dim

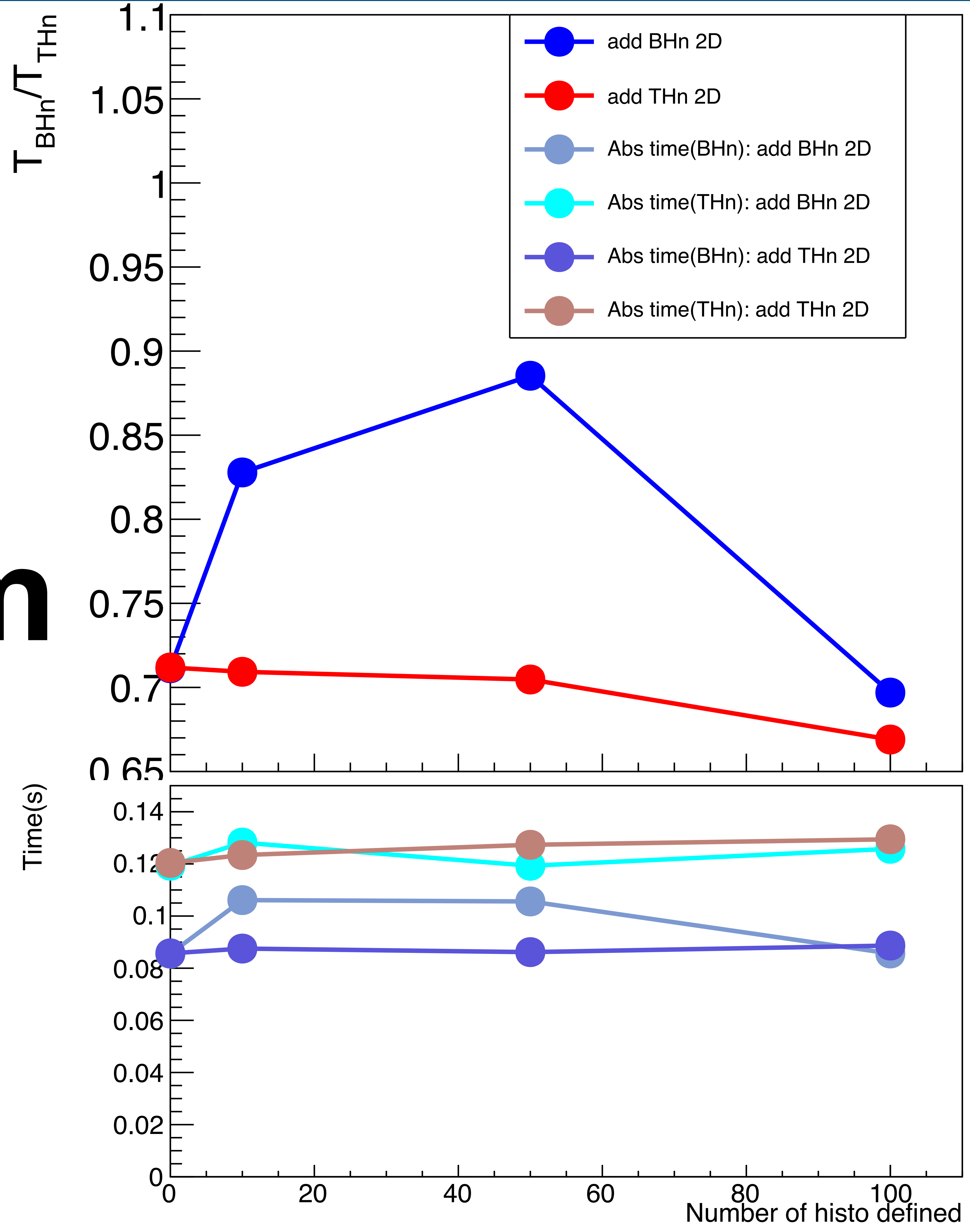


Add more boost histograms/root histogram

- Total Events filled = 10 **million** in each dimension
- # of bins in range: **100** in {0, 10}
- Uniform distribution of ints in {0,1.0}

Boost histograms
Faster but slows down as increasing number of BHn in the macro

2 Dim

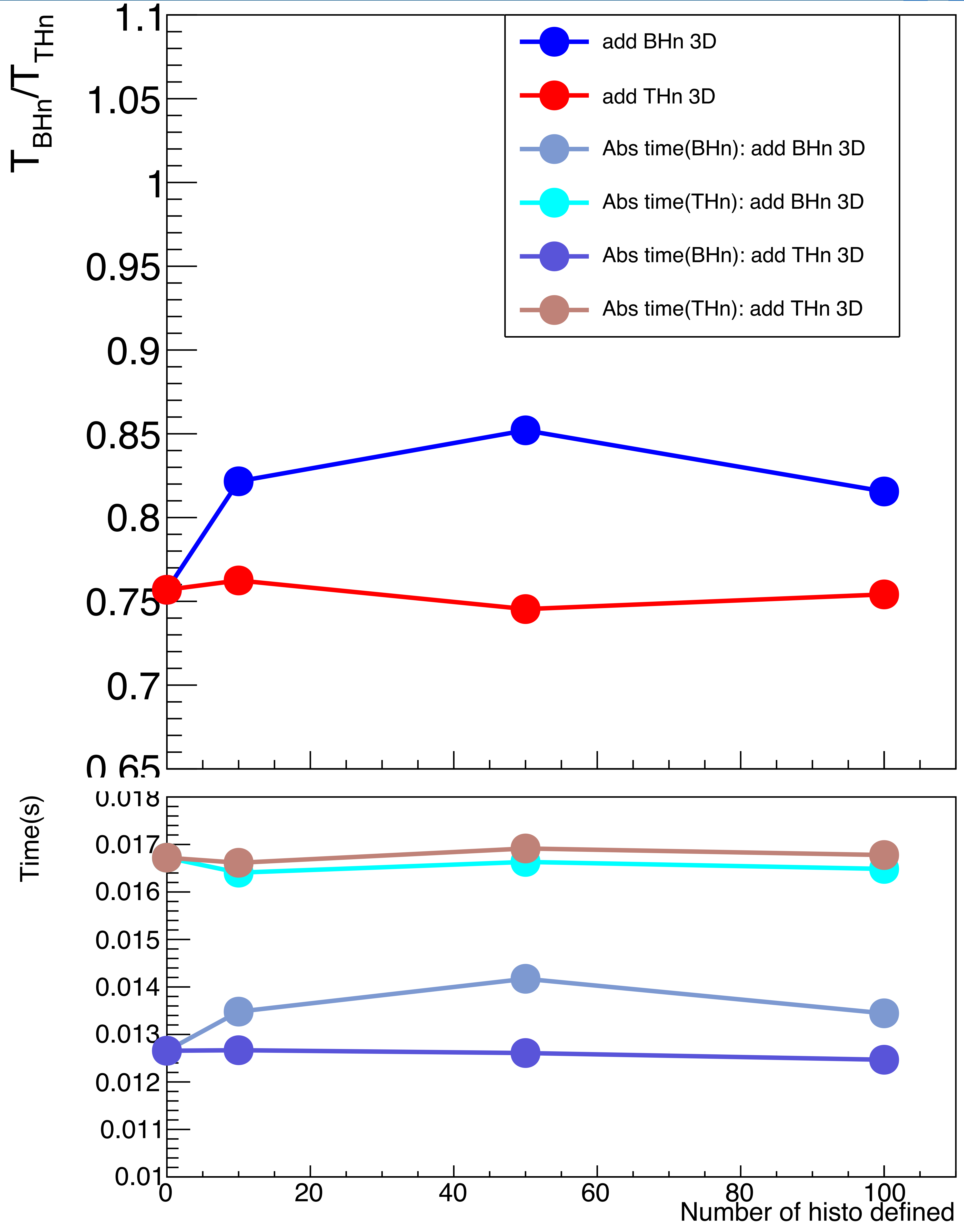


Add more boost histograms/root histogram

- Total Events filled = 10 **million** in each dimension
- # of bins in range: **100** in {0, 10}
- Uniform distribution of ints in {0,1.0}

Boost histograms
Faster but slows down as increasing number of BHn in the macro

3 Dim



Additional slides

Intro

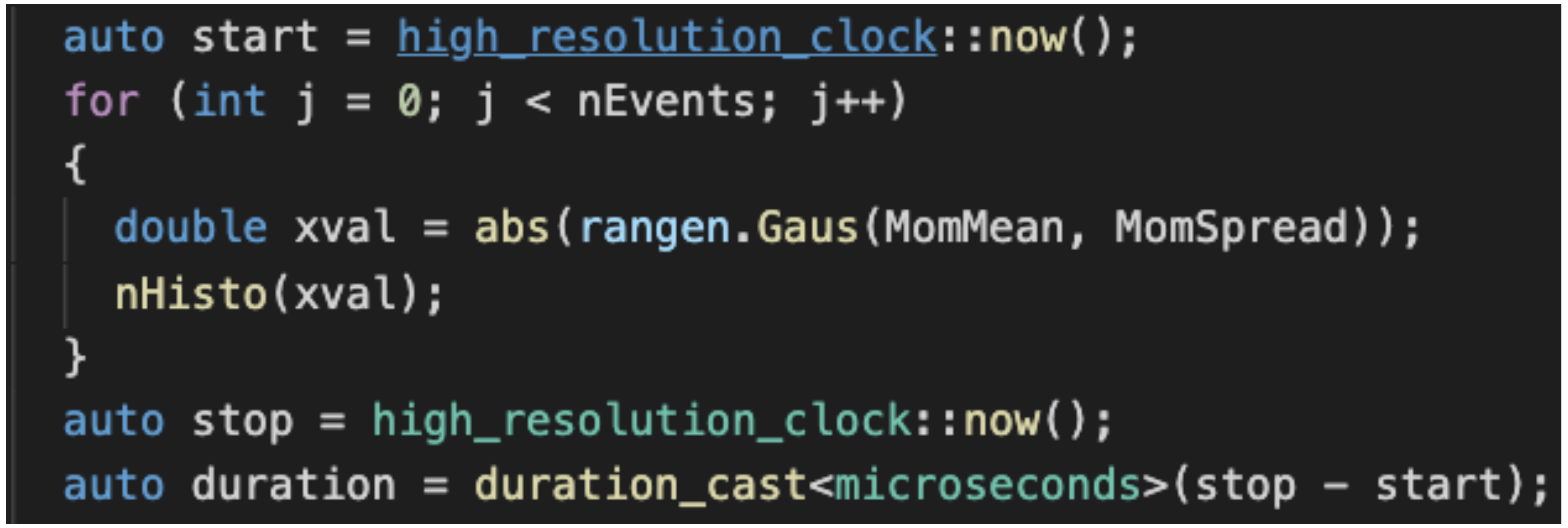
- Check run-time with :

```
Create BHn object;
start1 clock( );
loop(events){
   BHn_histo->Fill(x,y… ) //call operator::accepts n args
}
stop1 clock( );
```

Exactly same way for TH1D,TH2D and TH3D classes

```
Create THn object;
start2 clock( );
loop(events){
 THn_histo->Fill(x,y… ) //call operator::accepts  n args
}
stop2 clock( );
```

Example

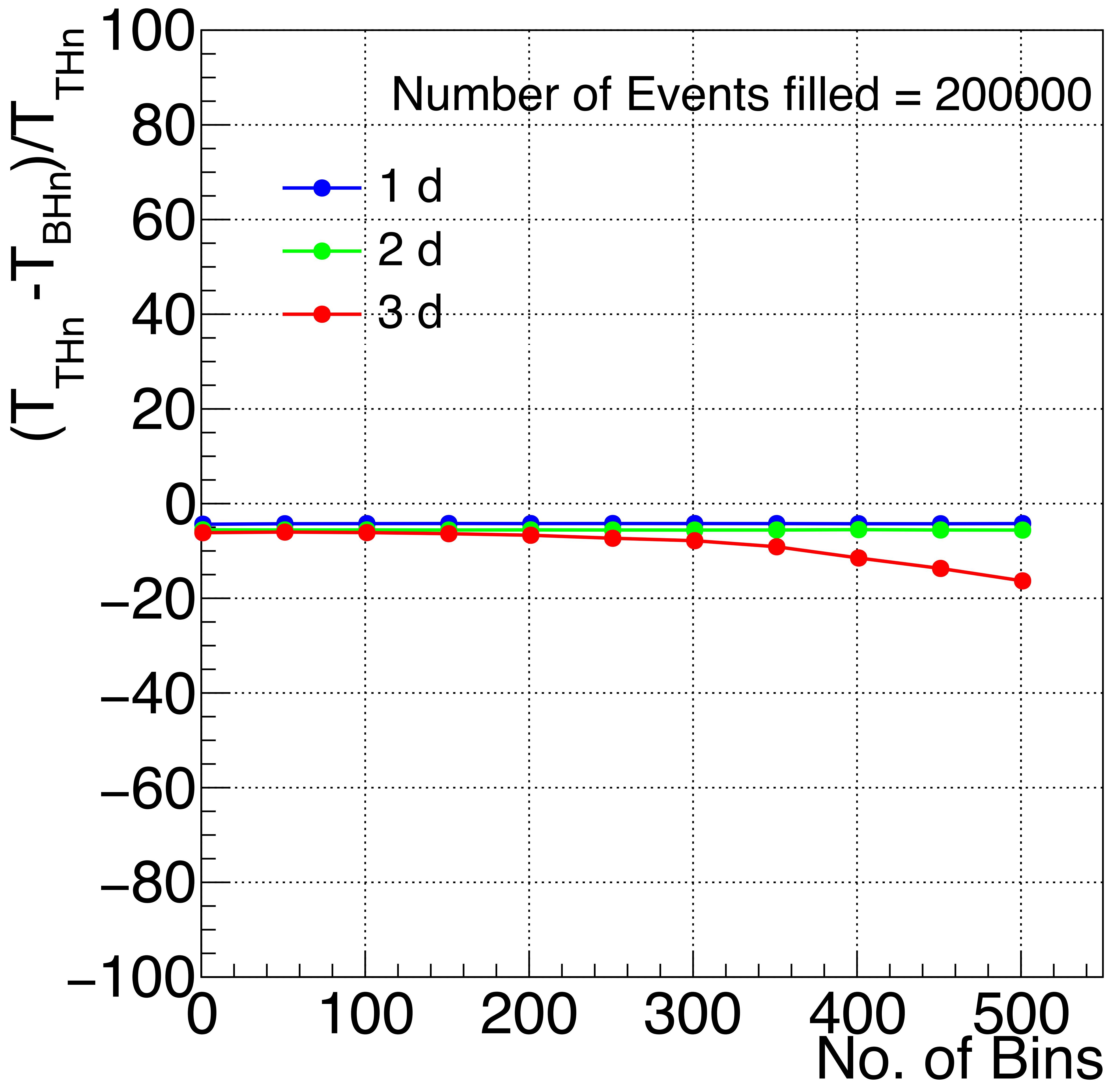


```
auto start = high_resolution_clock::now();
for (int j = 0; j < nEvents; j++)
{
  double xval = abs(rangen.Gaus(MomMean, MomSpread));
  nHisto(xval);
}
auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start);
```

- **No bias:** in measurement of filling method
- **Minimum bias:** due to random number generator
  - sufficiently large number of events in **Normal distribution** in [0-10]

BHn vs THn: #bins vs time

- Total Events filled = **200000** in each dimension
- axis range- 0 -10
- filled with random number generator [0,10]



BHn vs THnsparseD: Time vs #of dim.

- Check run-time with :

A

```
Create BHn object;
start1 clock( );
loop(events){
    BHn_histo->Fill(x,y… ) //call operator::accepts n args
}
stop1 clock( );
THnsparse has different way of filling values

Create THnSparse object;
start2 clock( );
loop(events){
   loop(dim){
   fill and array[dim] = values
   }
THnSparse->Fill( array[d] ) accepts n dimensional object
}
stop2 clock( );
```

- Check run-time with :

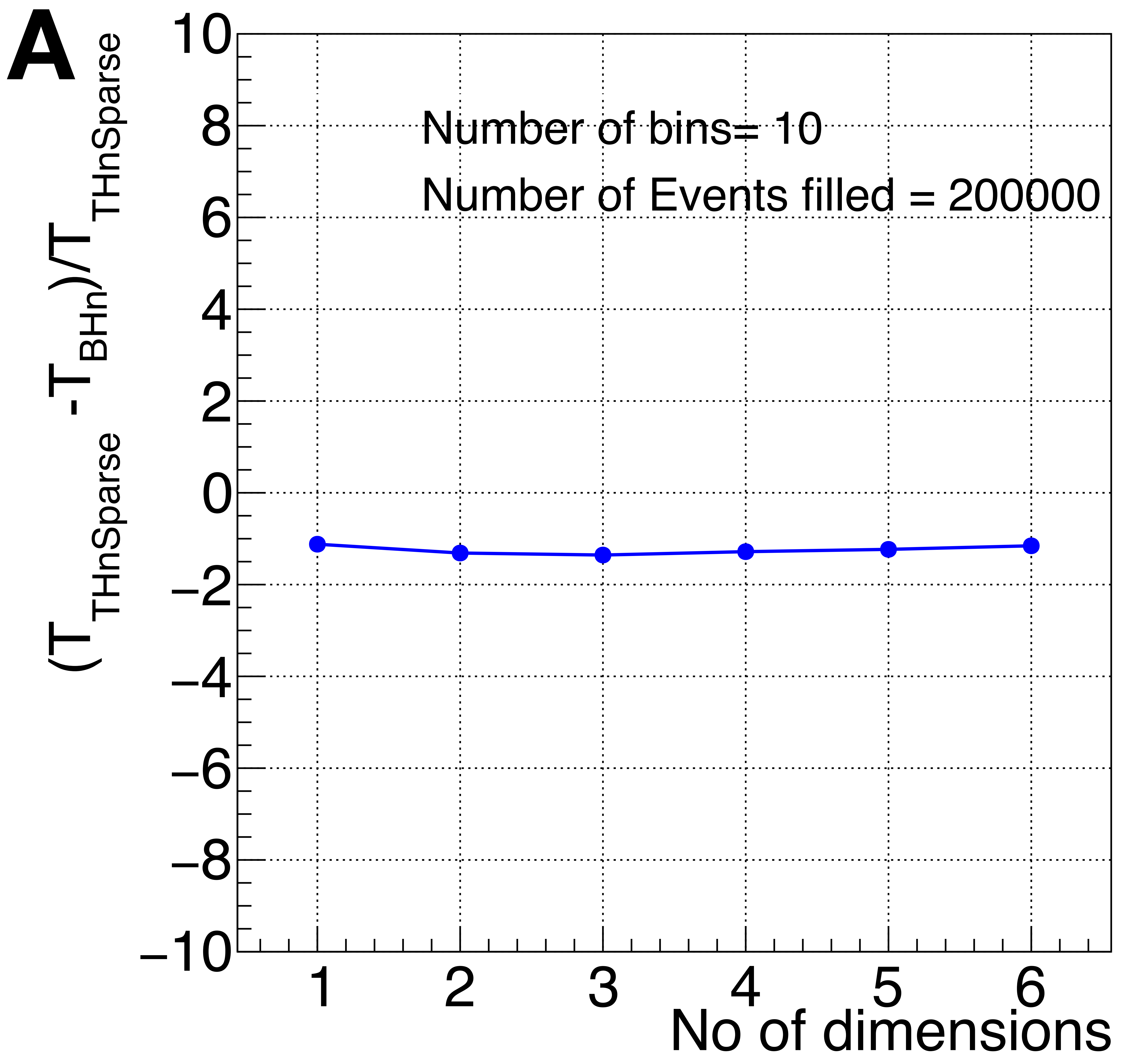
B

```
Create BHn object;
create vector of vectors;
start1 clock( );
loop(dim){
  loop(events){
                vector(vector)
  }
}
BHn_histo->Fill(vector(vector)) //call operator::chunk wise fill
stop1 clock( );
THnsparse has different way of filling values

Create THnSparse object;
start2 clock( );
loop(events){
  loop(dim){
  fill and array[dim]
  }
THnSparse->Fill( array[d] ) accepts n dimensional object
}
stop2 clock( );
```

BHn vs THnsparseD: Time vs #of dim.

- Check run-time with :
  - $THn_{SparseD}$ = time in filling THnSparse filling 200000 events in 10 bins in 0-10
  - THn_{BHn} = time in filling BHn filling 200000 events in 10 bins in 0 -10

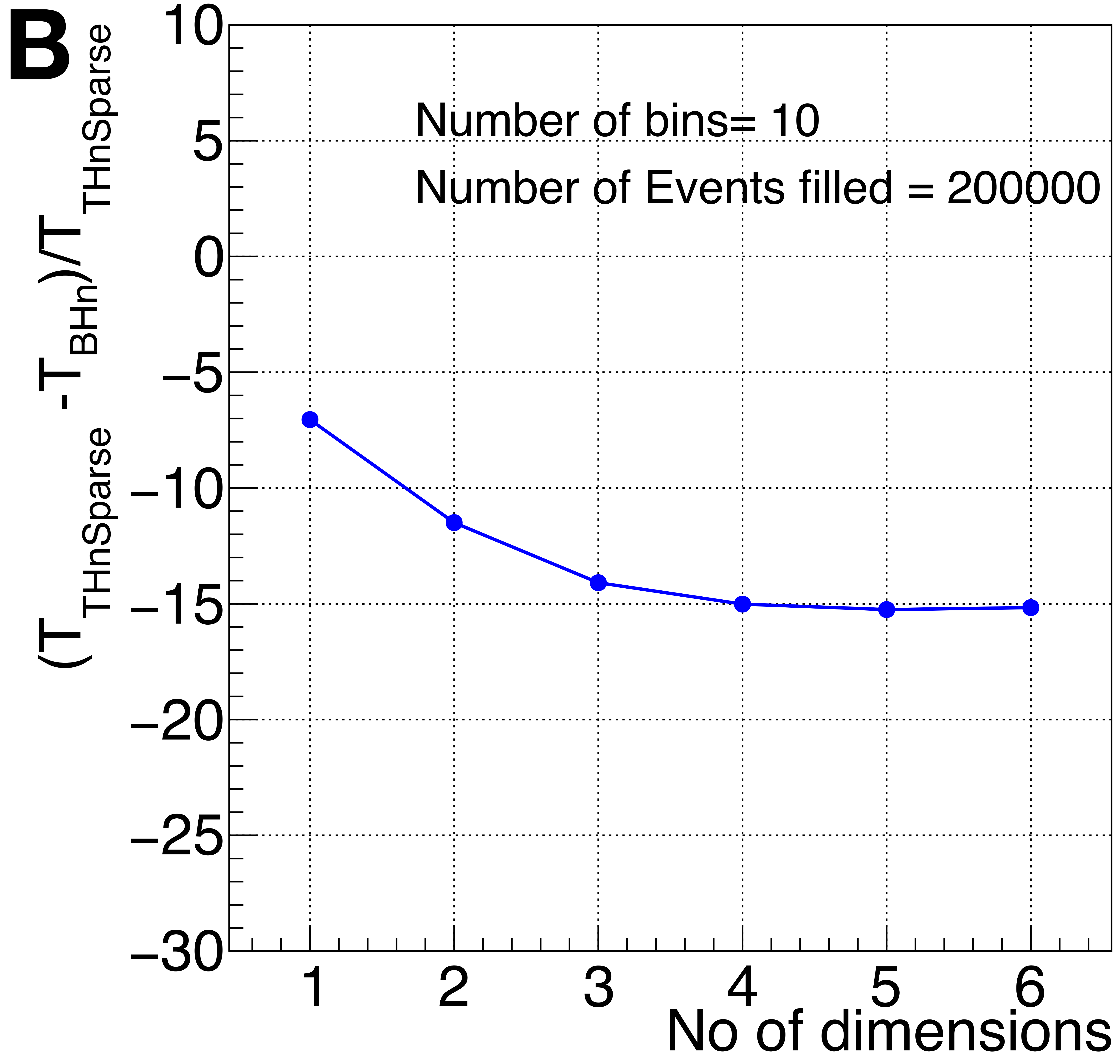
Seems BHn class is slower than THnSparseD



BHn vs THnsparseD: Time vs #of dim.

- Check run-time with :
  - THn$_{\text{SparseD}}$ = time in filling THnSparse filling 200000 events in 10 bins in 0-10
  - THn$_{\text{BHn}}$ = time in filling BHn filling 200000 events in 10 bins in 0 -10

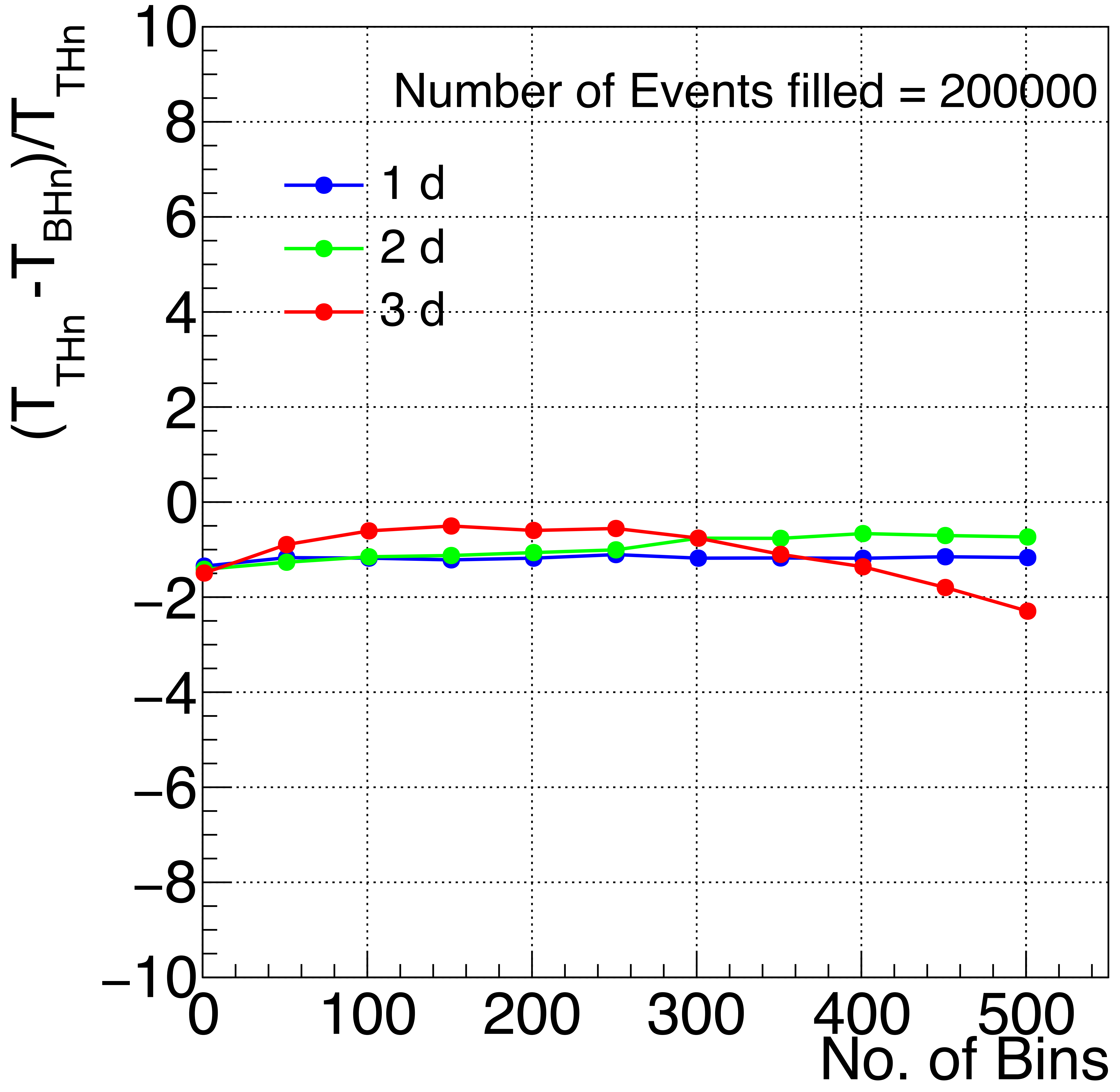
Seems BHn class is slower than THnSparseD



BHn vs THnsparseD: #bins vs time

- Total Events filled = **200000** in each dimension **A**

Increasing bins seems even worse for 3D

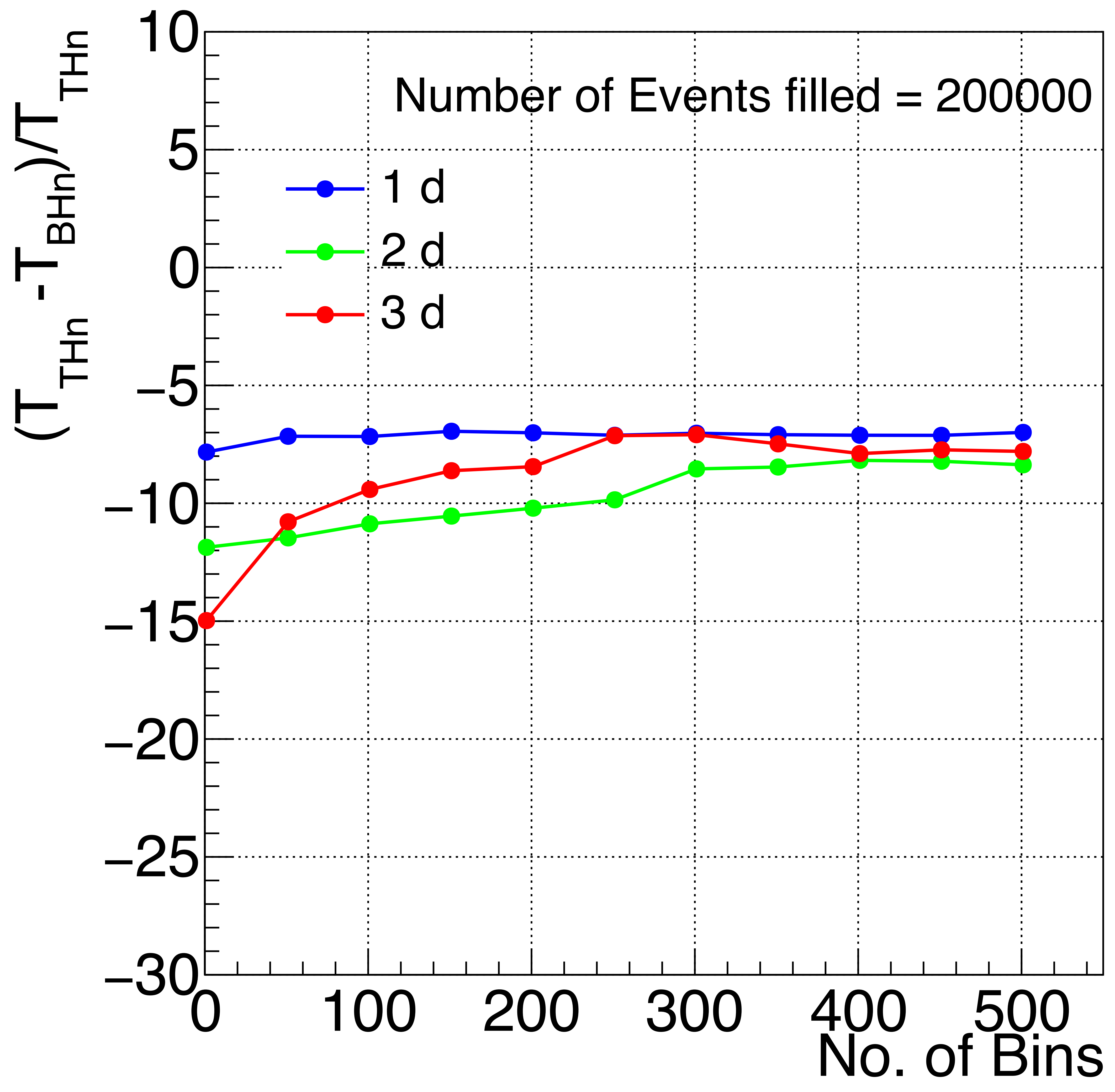


BHn vs THnsparseD: #bins vs time

- Total Events filled = **200000** in each dimension

BH3 is worse for low number of bins

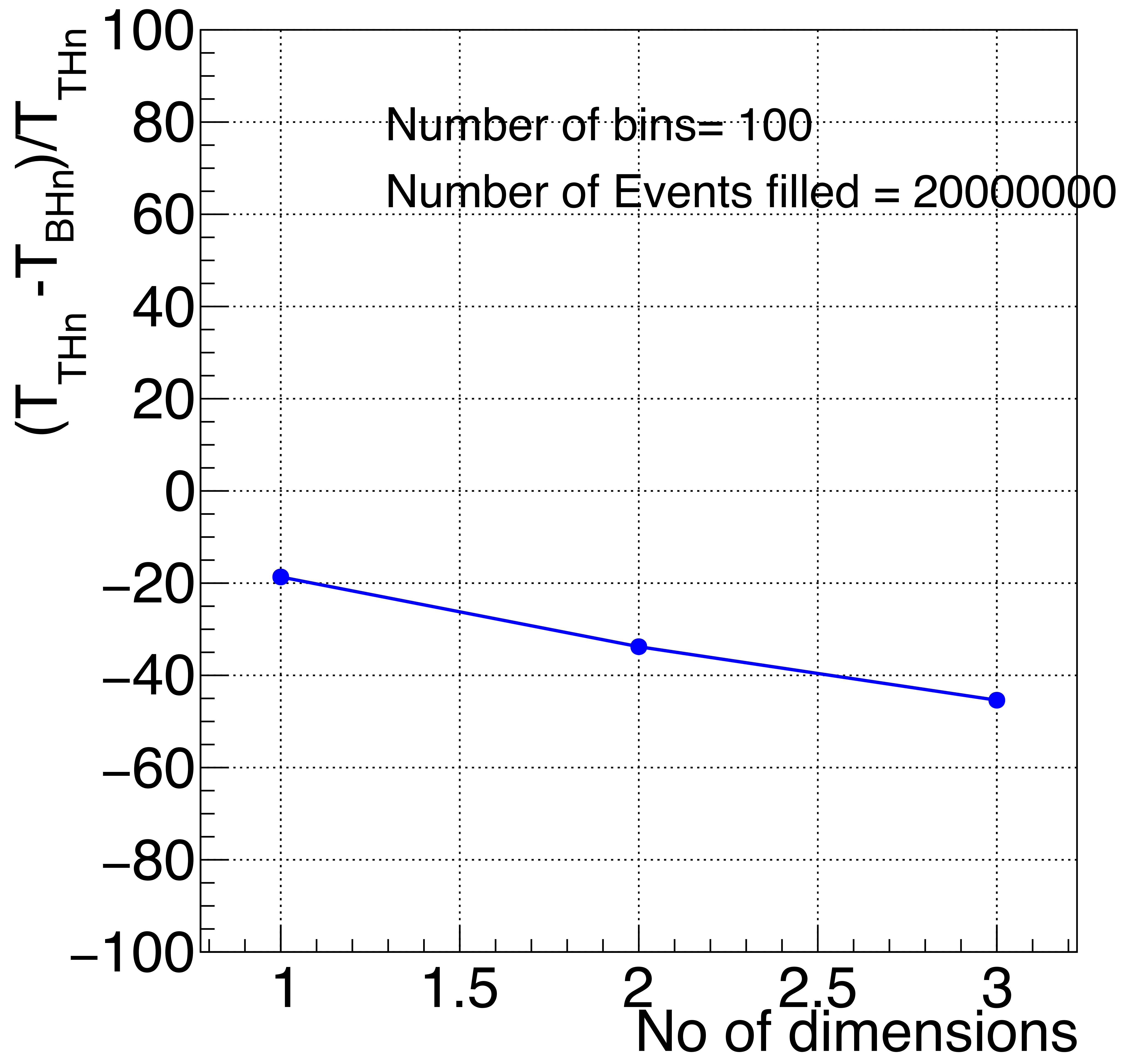
B



BHn vs TH1D,TH2D and TH3D

- Total Events filled = **20,000,000** in each dimension
- # of bins in range: **100** in {0, 10}

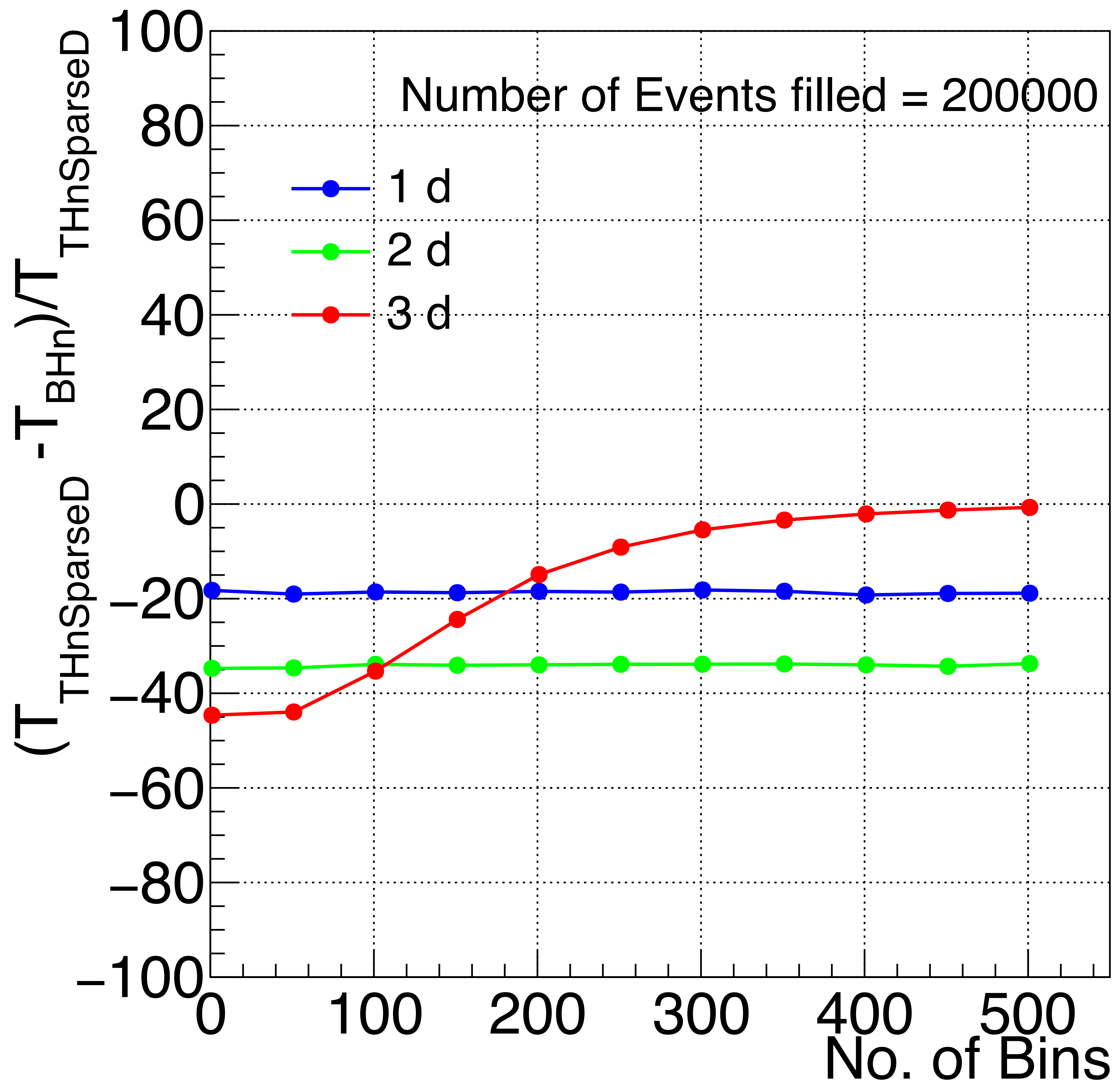
THn wins here as well



BHn vs THn: #bins vs time

- Total Events filled = **200000** in each dimension
- axis range- 0 -10
- filled with random number generator [0,10]

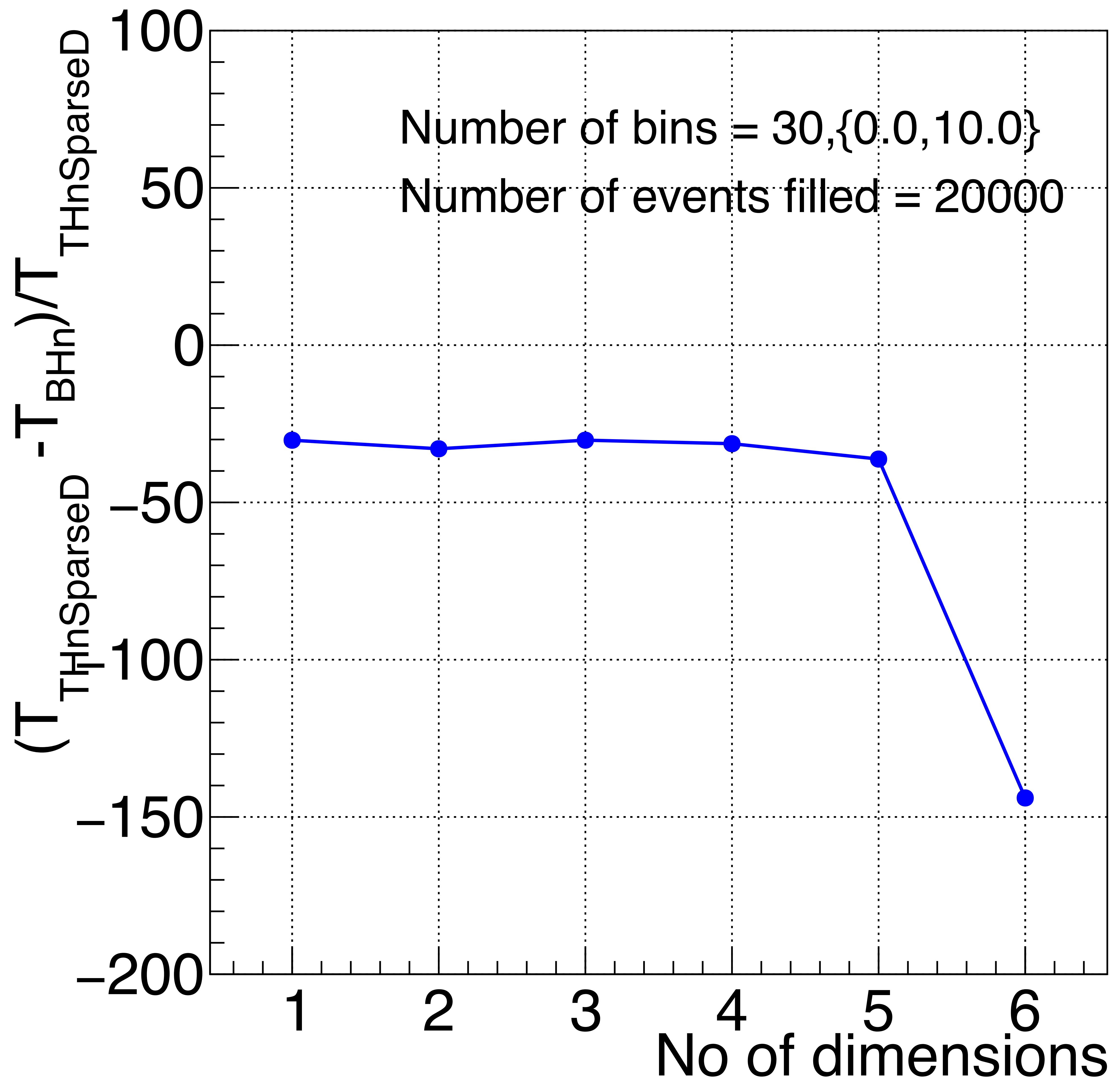
BH3 takes equivalent time wrt TH3D for large # of bins



BHn vs THnsparseD: Time vs #of dim.

- Check run-time with :
  - $THn_{SparseD}$ = time in filling THnSparse filling 20000 events in 30 bins in 0-10
  - THn_{BHn} = time in filling BHn filling 20000 events in 30 bins in 0 -10

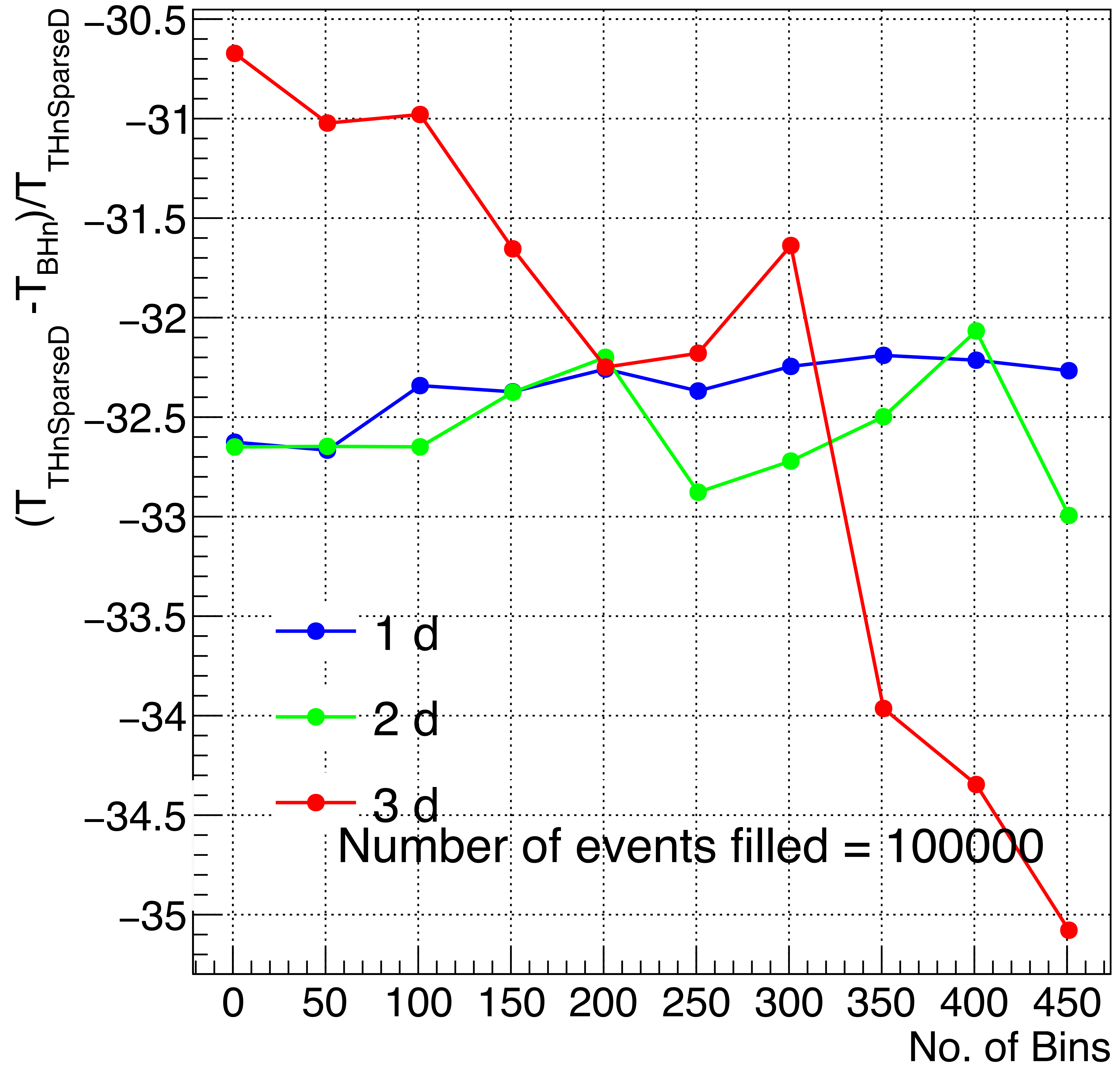
Seems BHn class is slower than THnSparseD



BHn vs THnsparseD: #bins vs time

- Total Events filled = **100000** in each dimension

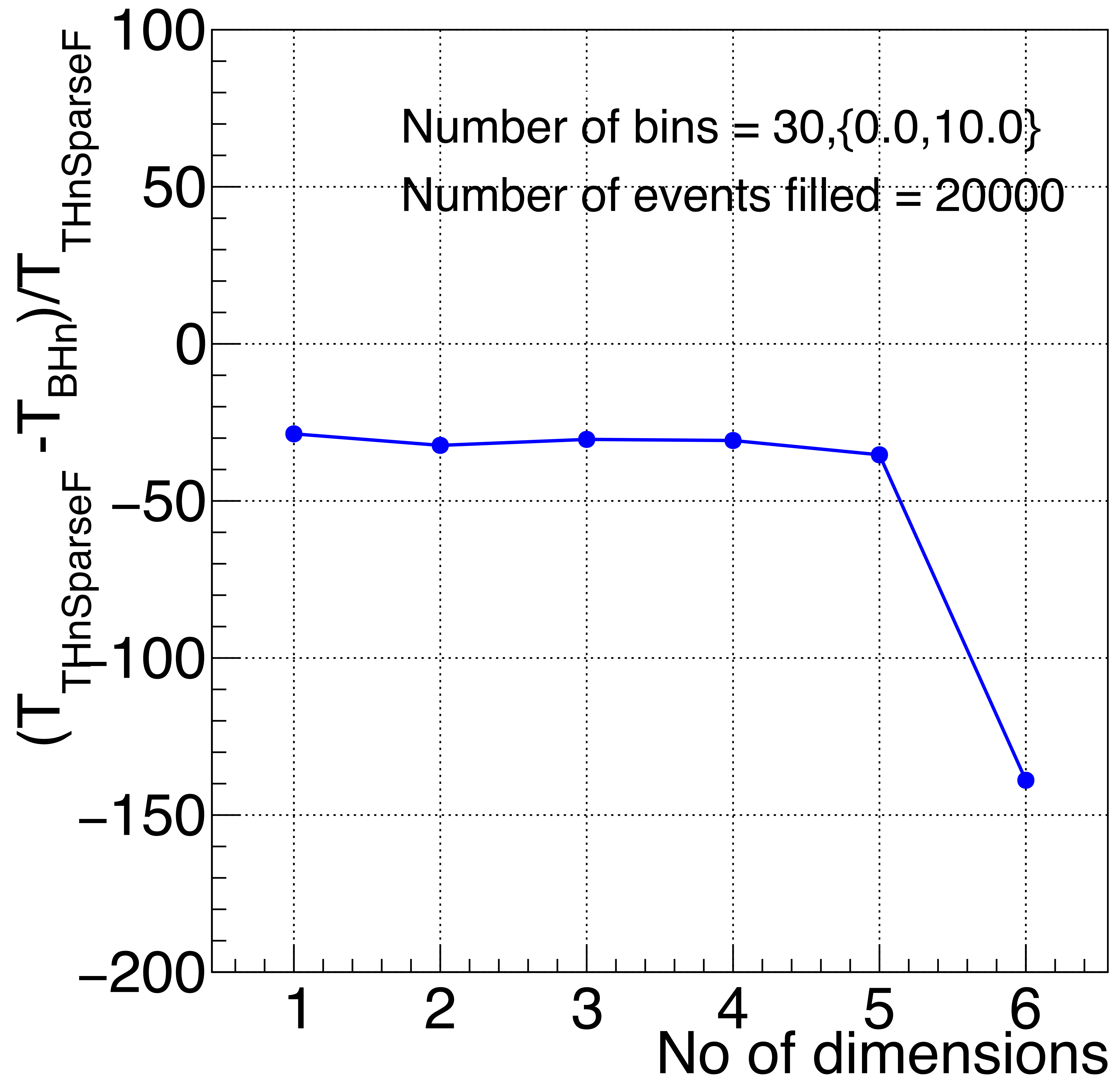
Increasing bins seems even worse for 3D



BHn vs THnsparseF: Time vs #of dim.

- Check run-time with :
  - $THn_{SparseD}$ = time in filling THnSparse filling 20000 events in 30 bins in 0-10
  - THn_{BHn} = time in filling BHn filling 20000 events in 30 bins in 0 -10

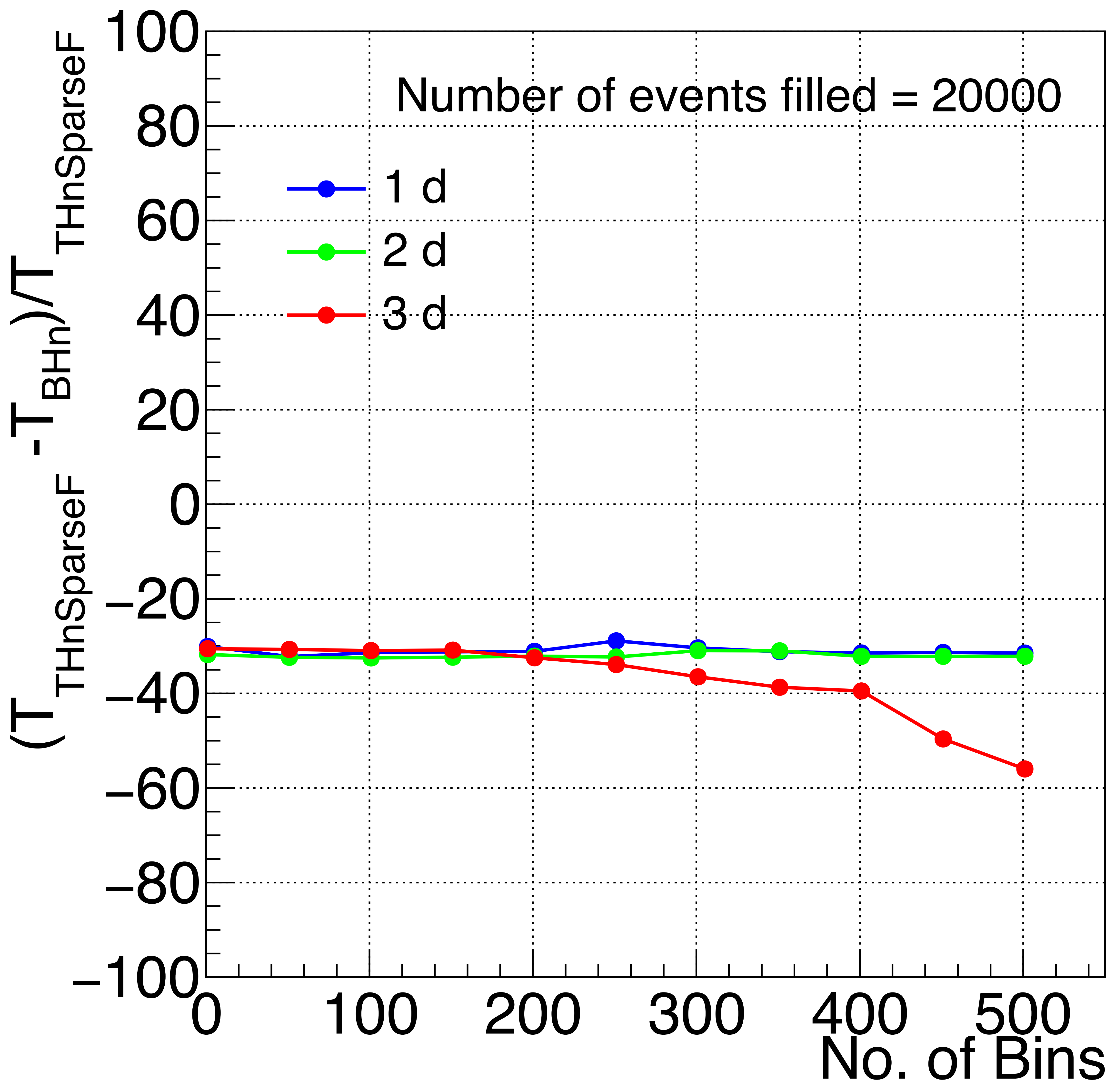
BHn class is slower than THnSparseD for float as well



BHn vs THnsparseF: #bins vs time

- Total Events filled = **20000** in each dimension

Increasing bins seems even worse for 3D


Number of events filled = 20000
1 d
2 d
3 d
$(T_{THnSparseF} - T_{BHn})/T_{THnSparseF}$
No. of Bins