Physics validation of changes to `PDFastSimPAR`

And what comes next

Marc Paterno

February 26, 2024

Prologue

- I have presented performance improvement possible by replacing `fast_acosd` approximate calculation of $\cos^{-1}(x)$.
  - `std::acos(double)` (considerably slower, but closes to exact calculation)
  - `fast_acos` (what `PDFastSimPAR` uses now)
  - `hastings_acos` (44% faster than `fast_acos`, identical output)
  - `hastings_acos_4` (same speed as `hastings_acos`, better approximation)
  - `hastings_acs_5` (33% faster than `fast_acos`, much better approximation)
- This time, I will show comparisons between the output of `PDFastSimPAR` using the different algorithms.
- Goal: to decide which $\cos^{-1}(x)$ algorithm is most appropriate to use in this context.

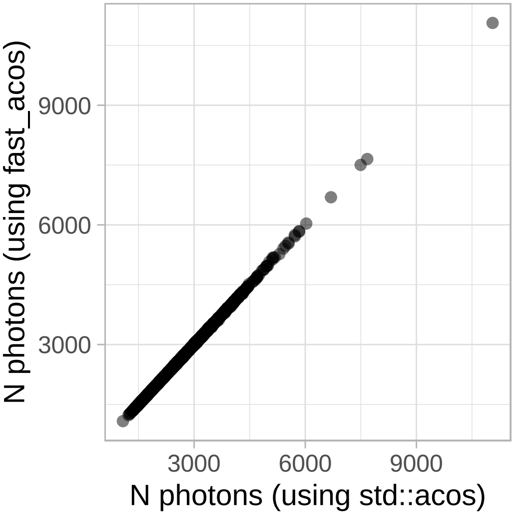
`sim::SimPhotonsLite`

- The output of `PDFastSimPAR`, in the configurations of the workflows used by DUNE, consists of `std::vector<sim::SimPhotonsLite>` (henceforth `SPL`) objects.
- Each element in the `vector` represents data for a *channel*.
- The data for each *channel* are a channel ID and a record of a time series of measurements.
- The series of measurements for a channel is recorded as an `std::map<int,int>`.
  - The first `int` (the key) represents a time, measured in *ticks*.
  - the second `int` (the value) represents a count of photons observed at that time.

Comparison of results

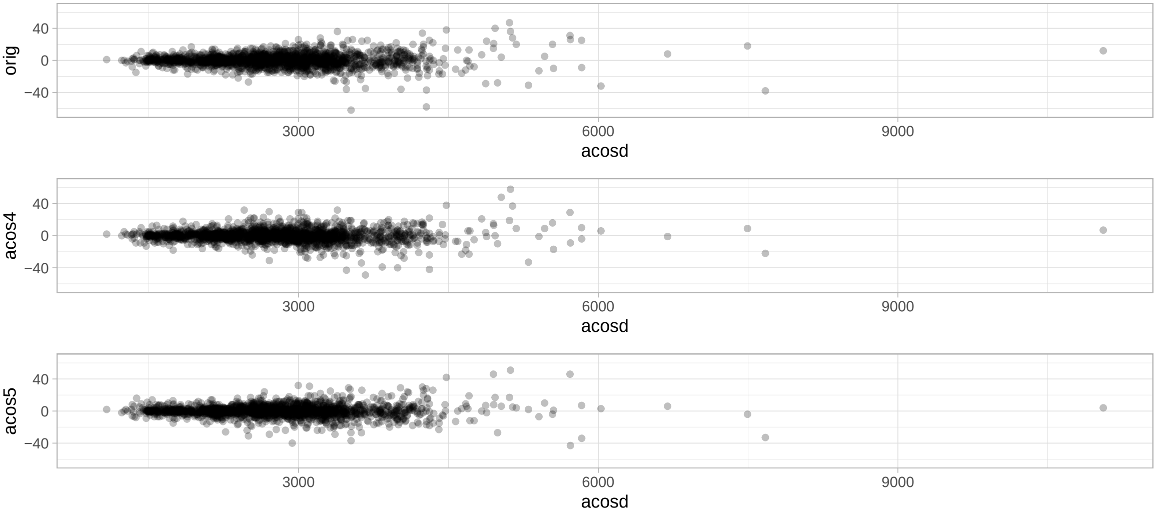
- To determine the effect of using the different algorithms for $\cos^{-1}(x)$, I compare the results from using the "exact" result of `std::acos` to the results from using the different fast approximations.
- In order *not* to confound the comparison with the effect of other complicated algorithms, I have looked at direct displays and comparisons of (distributions of) the `PDFastSimPAR` output.
- I will present several different comparisons.
- Some additional detail is available online.

Correlation in number of photons in each channel in each event

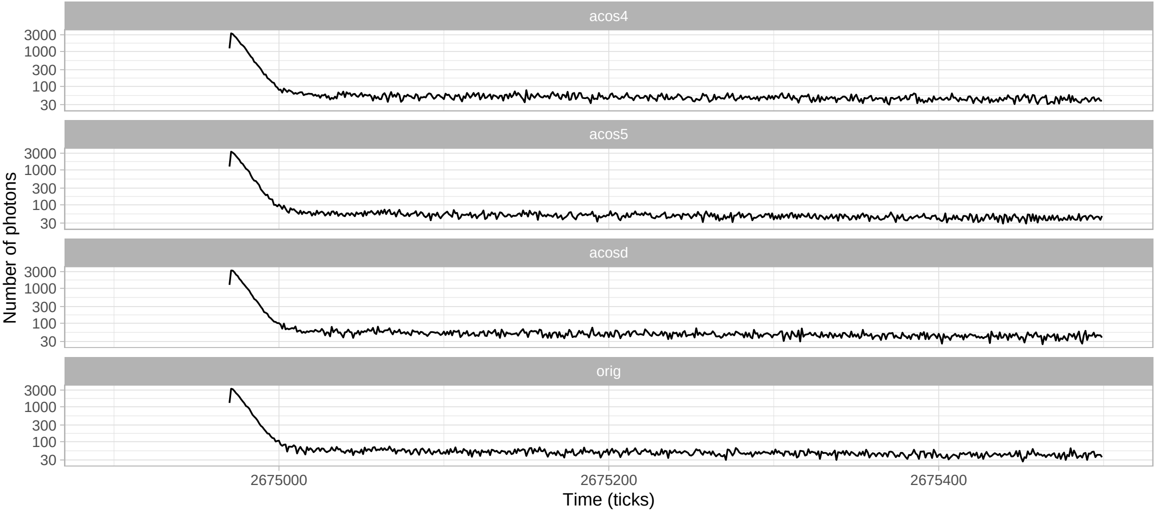


- Using the other algorithms, the correlation is *identical* (`hastings_acos`) or very slightly different (`hastings_acos_4`, `hastings_acos_5`).
- This is a very coarse comparison — the ancillary document has some more detail.
- The distribution of photon counts per measurement varies little by changing which $\cos^{-1}(x)$ algorithm is used.

Deviation from exact correlation, vs photon count using `std::acosd`



Detailed look at the busiest signal (event 6, channel 108, 600 tick span)



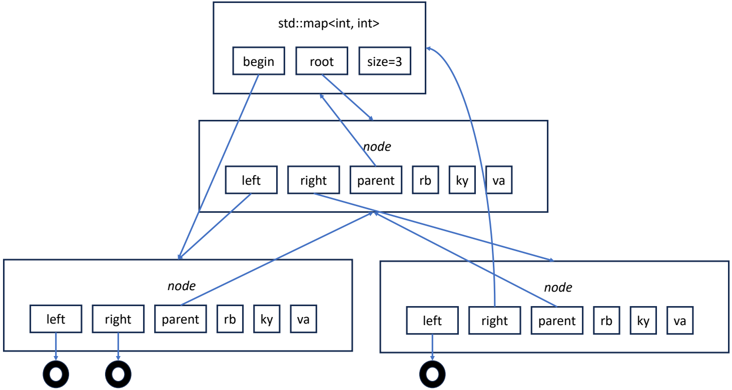
How to decide which is best?

- Which of the trigonometic algorithms used makes very little difference to the output of `PDFastSimPAR`.
- Since the original (and least accurate) approximation has been adequate, it does not seem that a more accurate approximation is required.
- Both `hastings_acos` and `hastings_acos_4` have identical speed, and are about 44% faster than the current algorithm.
- It seems to me that one of these two would be the best choice.

Next step

- My next step in optimization is the parallelization of the processing.
- The first part of this is creating a data structure that is efficient for parallel processing.
- The current data structure is far from being efficient.
- The main issue is terrible *locality of reference*, leading to terrible cache usage.
- The secondary issue is the amount of wasted memory.

What does a `std::map` look like in memory?



rb = *red/black* (char); ky = key (int); va = value (int)

- For SPL, the size of each node is probably 36 bytes (including padding); of this 8 bytes are the data ky and va (more than 75% of the space is overhead).
- The nodes are distributed all around memory.
- The typical `map` (channel) has about 3000 such nodes.
- Each `vector<SPL>` requires about 500,000 allocations and deallocations.

What would be a more efficient data structure?

- This depends upon the access pattern(s) of code using the data.
- *If* the common pattern is iteration through the channel, then replacing `std::map<int,int>` with `std::vector<std::pair<int,int>>` would be more efficient.
- Possibly still better is two parallel vectors:

```
std::vector<int> tick;
std::vector<int> n_photons;
```

- I propose to survey the code consuming `SPLs` to determine which access patterns are observed.
- It will be necessary to measure the results of any changes, to see whether relevant performance improvement is observed.
- *If* such a change is worthwhile, deployment will require dealing properly with schema evolution, to retain usefulness of existing data files.

Thanks for your attention

Questions?

Extras: Number of measurements per channel (original algorithm)