AMSTERDAM
SCIENTIFIC
INSTRUMENTS

Luna

the Timepix Processing Suite

AMSTERDAM SCIENTIFIC INSTRUMENTS

Luna is a computation graph

Each step, or module, in a computation graph is a "GenericNode"

Generic nodes know how to "run" behaviours associated with them.

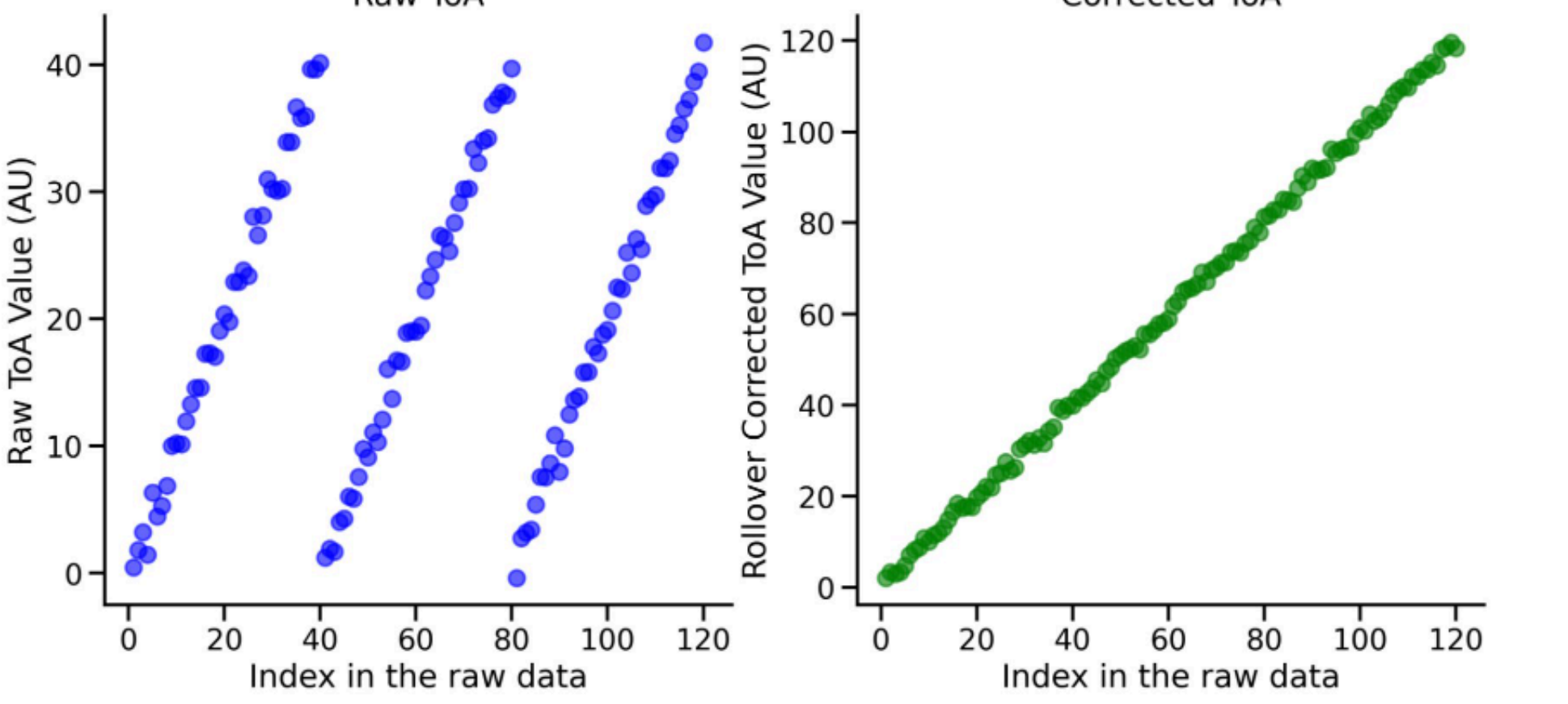
Instantiating a generic node means associating a behaviour with it.

Running a generic node means spinning up a run loop on a new thread and waiting for data to process. Nodes are connected by edges, which are "unbounded channels" from the crossbeam library (a powerful concurrency library in rust).

Once all nodes are running, data chunks are fed through channels to the next destination, eventually ending at the hdf5 sink which writes the data to file.

Unpacking

Raw binary data are 64 bit integers, unpacked using bit manipulation and parsed into rust numeric types.



Header

- Size of data to come (bytes)
- Chip ID

Data

- Counts on different clocks durations
- Counts are combined using a well defined equation based on the electronics
- Rollover correction
- Uses Chip ID to compute address

PixelHit

- X
- Y
- ToA
- ToT

TDC

- Timestamp
- Which TDC

Sorting

Pixel hits are not sorted in time (ToA) out of the box.

Problem: We are processing data in unsorted chunks.

If we were to sort this data then in their chunks then the data will sometimes be incorrectly sorted.

Raw ToA Data: 1 5 2 4 3 6 7 4 9 10 12 11 13 16 15 14

Chunks of 4: 1 5 2 4 3 6 7 4 9 10 13 11 12 16 15 14

Sorted Chunks: 1 2 4 5 3 4 6 7 9 10 11 13 12 14 15 16

How do we solve this problem?

Sorting and Caching

After unpacking data are sorted into a "TimeBinCache" before further processing

(Floor divide by chunk_size=4)

Raw ToA Data: [1, 5, 2, 4, 3, 6, 7, 4, 9, 10, 12, 11, 13, 16, 15, 14]

Compute Bin ID [0, 1, 0, 1, 0, 1, 1, 1, 2, 2, 3, 2, 3, 4, 3, 3]

Move data into cache using ID [1, 2, 3, 5, 4, 6, 7, 4, 9, 10 ,11 , 12, 13, 15, 14, 16]

Now we can sort accurately... [1, 2, 3, 4, 4, 5 6, 7, 9, 10 ,11 , 12, 13, 14, 15, 16]

Sorting and Caching

Using different parameters: --time-bin-interval = 4
--num-time-bins = 3

Cache has a fixed size of 3.

Raw ToA Data:

[1, 5, 2, 4, 3, 6, 7, 4, 9, 10, 12, 11, 13, 16, 15, 14]

Compute Bin ID

[0, 1, 0, 1, 0, 1, 1, 1, 2, 2, 3, 2, 3, 4, 3, 3]

(Floor divide by chunk_size=4)

Insert data

[1, 2, 3, 5, 4, 6, 7, 4, 9, 10 ,11]

The moment 12 is entered, green is evicted!

Evict green, continue inserting

Cache is full

[5, 4, 6, 7, 4, 9, 10 ,11, 12, 13, 15, 14] [1, 2, 3]

The moment 16 is entered, orange is evicted!

Evict orange, continue inserting

Green is sent to the sorting node.

[9, 10 ,11 , 12, 13, 15, 14, 16] [5, 4, 6, 7, 4]

Orange is sent to the sorting node.

No more data, flush.

[16] [12, 13, 15, 14] [9, 10 ,11]

Cache is flushed, earliest first

AMSTERDAM SCIENTIFIC INSTRUMENTS

Clustering

Chunks are processed in the same order that they are evicted from the cache.

Chunks are clustered individually and the boundaries are checked for split clusters and merged if needed.

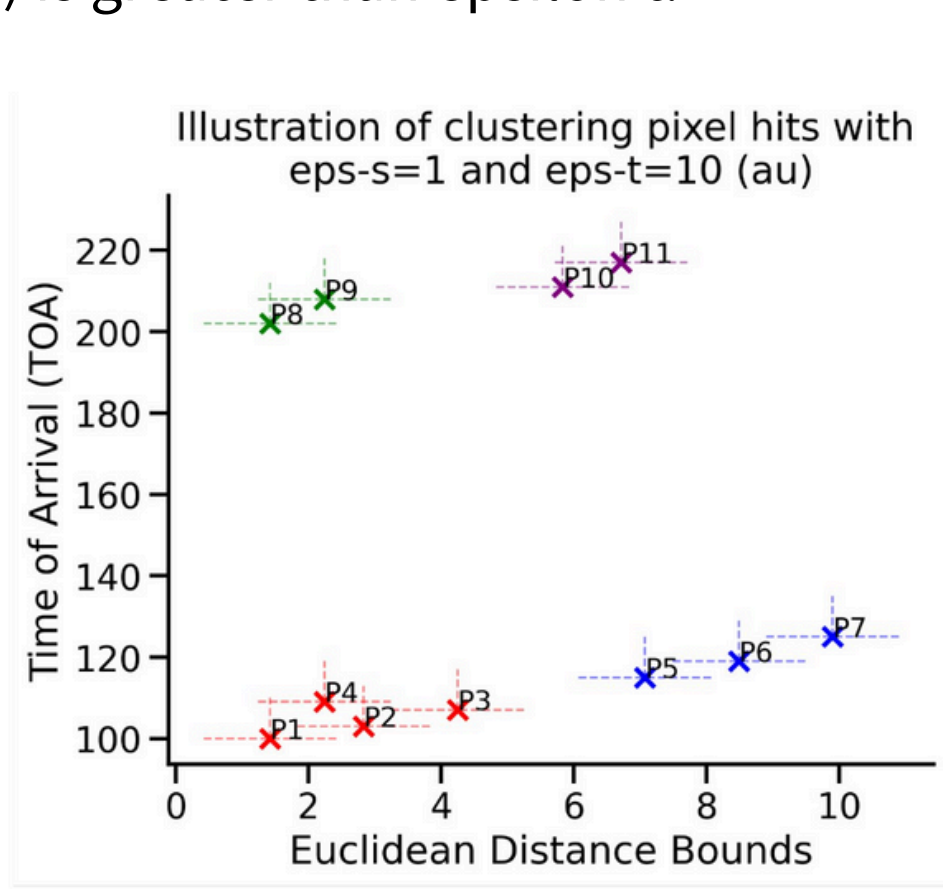
The algorithm we use is a simple quadratic algorithm with some optimizations. Its slow.

We iterate over every pixel in a chunk and try to find clusters starting from that pixel.

Compare each pixel with every other pixel and calculate

- time difference <= epsilon t
- Euclidean difference <= epsilon s
- If both are true, add it to the current cluster
- If a pixel is marked as already processed, it is skipped.

Optimization: To reduce the number of comparisons we close a cluster after difference in time (toa) is greater than epsilon t.



AMSTERDAM
SCIENTIFIC
INSTRUMENTS

For more information, contact our sales team
hello@amscins.com