CZT implementations in C and python and python performance

BLonD Meeting

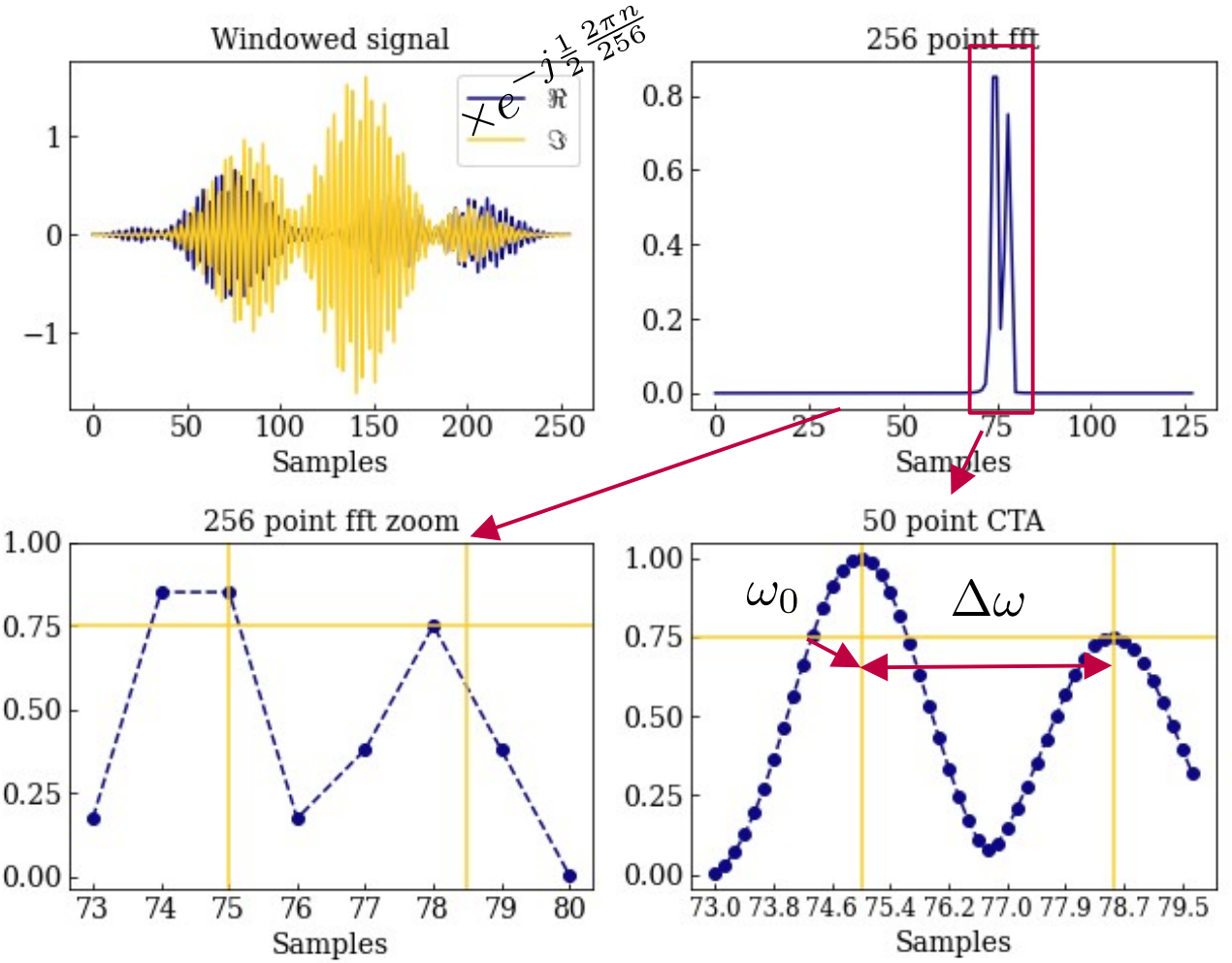
M. Vadai

Recap on CTA

- Previous presentation with math and examples (also CZT): https://indico.cern.ch/event/996624/
- CTA: chirp transform algorithm. Calculates the Fourier transform with arbitrary start and stop frequencies on the unit circle and an equal spacing between them.
- Usual calculation (1960s algorithm) ~ 2xFFT + 1xIFFT.
- Expectation on run time: computationally more efficient than the FFT when the zoom is larger than roughly 3-4 compared to the equivalently padded FFT.

Illustration

- 75/128 and 78.5/128 frequency sinusoids with 1 and ¾ amplitude added. Signal length: 256 samples.
- Hanning window + FFT.
- CTA calculated for the 256(+50) point signal. ~6x resolution.
- Calculating an only two point CTA would have been sufficient in this example.



Recap on CZT

- CZT: calculates a more general transform than Fourier with arbitrary start and stop frequencies. The transform operates along a spiral contour in the z plane vs the circle arc of the CTA.
- Usual calculation ~ 2xFFT + 1xIFFT.
- New since 2019: there is an efficient inverse N log N, potentially improving on the 3-4x limitation. Efficient implementations are not really available yet.

Use cases

- Astronomy: long acquired data, but only certain parts are interesting. Sometimes can be similar to our use cases.
- The original 1960s CZT was used in CCD sensors with applications in space and astronomy.
- Same thing seems to be happening now with the inverse, just for data analysis in astronomy.

Implementations in python

- A summary: https://gist.github.com/endolith/2783807
- Scipy: https://github.com/scipy/scipy/issues/4288
- The new inverse computation is implemented: https://pypi.org/project/czt/ or the same on github: https://github.com/garrettj403/CZT . Needs testing at the moment. But development is active, improved since my last presentation.
- There’s also my own implementation for the classic algorithm in examples, development code.

C / C++ implementations

- Problem of research: there are many CZT acronyms, so you have to check the code (even after filtering the obvious false positives).
- https://github.com/mickey305/CZTransform
  - Pure C. Classic algorithm.
  - Tested, compiles and works for CTA.
  - The FFT is custom, so probably was developed for a very specific purpose. The tests and source code comments are in Japanese, so I was relying on Google. Otherwise there is no other documentation.
  - Based on the calculations this seems to be CTA at first sight. This is correct technically, since CTA is a special case of CZT.
- https://github.com/thejonaslab/pychirpz
  - Python and C++. Classic algorithm.
  - Based on FFTW and numba https://numba.pydata.org/ .
  - Benchmarks for CTA vs FFT, confirming the zoom speed-up compared to FFT. As it can be shown based on the algorithm, too.
  - Tested, compiles and works for CTA and CZT, better documented.
- No new inverse CZT found in C / C++, and no time yet to write it myself.

Test setup

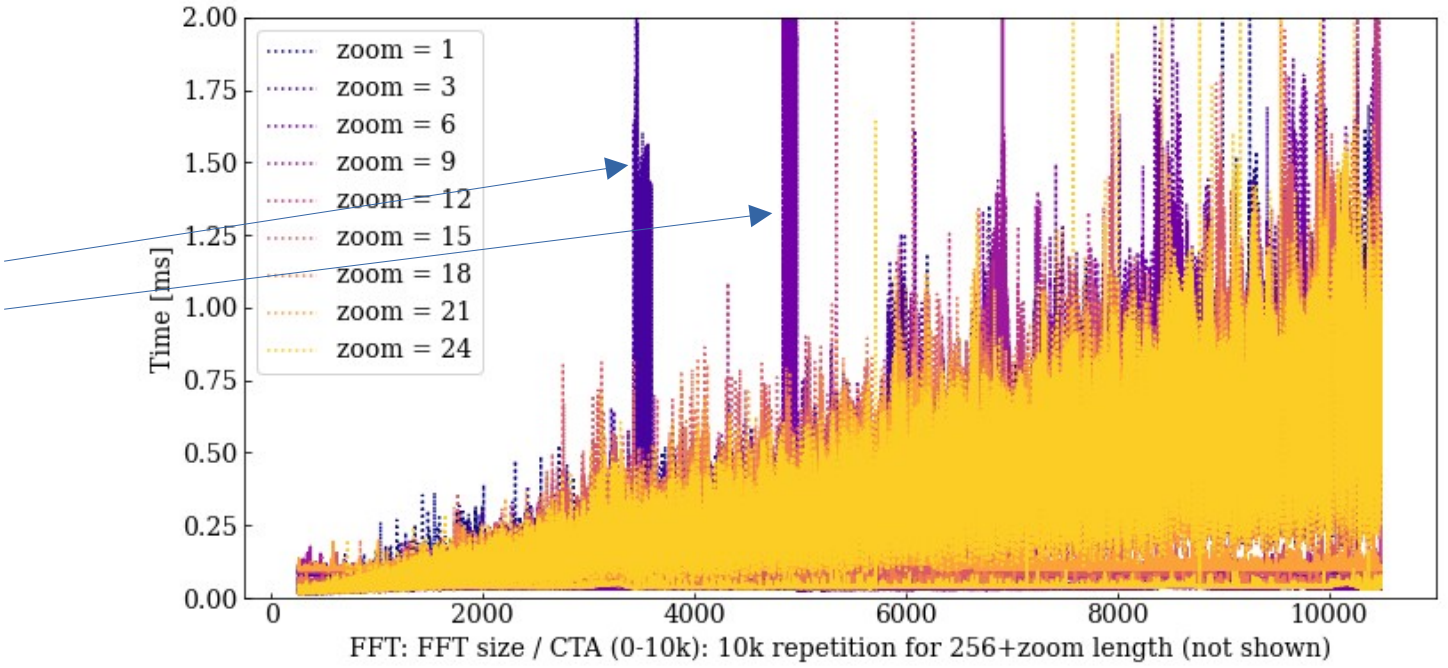
- Test aim: confirming algorithmic speed up.
- FFT vs CTA using the same numpy.fft / numpy.ifft for CTA, too, so the comparison has the same systematic errors from the software point of view. Therefore using my own CTA implementation.
- Increasing the length of the FFT, and see where CTA catches up on average.
- 9 different zoom settings for CTA for one run, two sets of settings.
  - Reproducibility on the same computer. (9 settings with CTA and FFT repetitions).
  - Decimation (in time or in frequency) FFT run time is highly sensitive to the length even for roughly the same size. (2x9 slightly different zoom settings).
- 10k point scan for FFT length corresponding to different paddings to achieve higher resolution in binning.
- Scan repeated for 2 signal sizes 256 and 2560 to check for transient effects (e.g. computer switching to performance mode) and to see the effect on reproducibility.
- perf_counter_ns() for time measurements, single thread performance for algorithm comparison.

Measurements

- Measurements:
  - How well does the algorithm behave?
    - Checking systematic deviation due to different partitioning of the DFTs because of the different FFT lengths.
  - How well does the computer behave?
    - Checking reproducibility for both FFT and CZT for the same partitioning.
  - How is the measured algorithm performance?
    - Comparing equivalent FFT and CZTs for a given zoom.

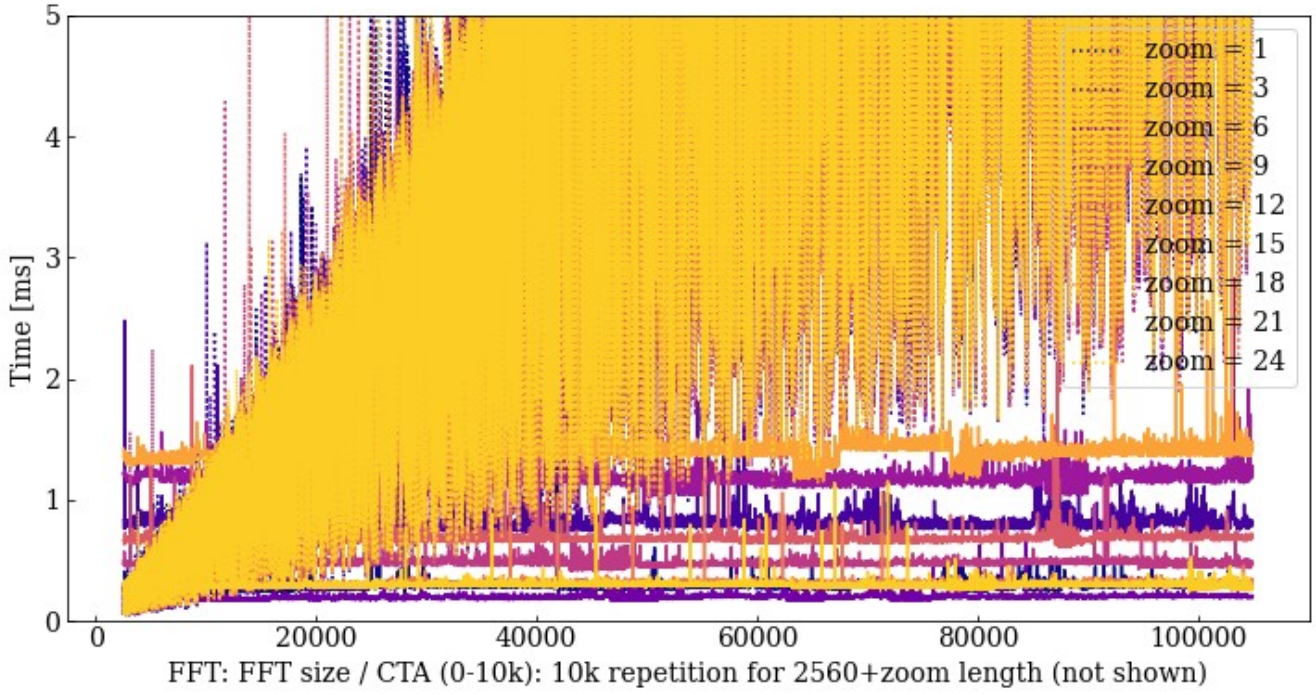
Results, raw timings

- Line: CTA
- Dotted: FFT.
- L = 256
- There are temporary slowdowns when system adapts to load. Draws some power from the network, spins up fan, etc. Systematics.
- As expected from a decimation algorithm, FFT size is critical in both FFT and CTA as seen from FFT spread here (CTA later).



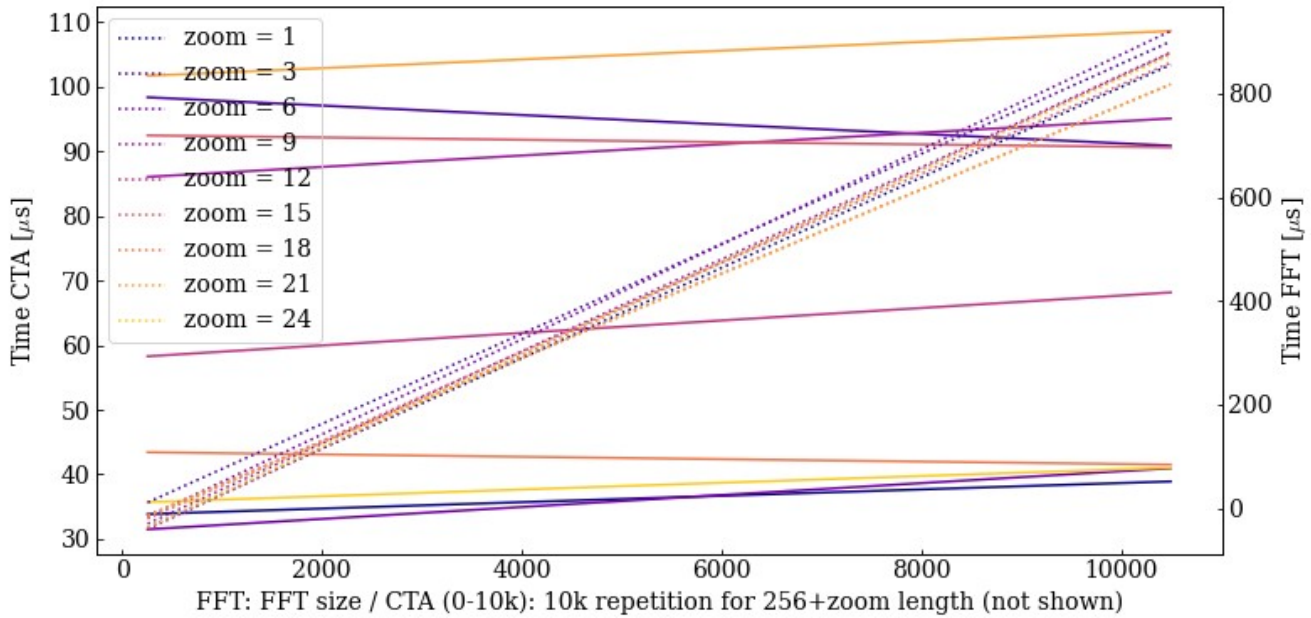
Results, raw timings

- Line: CTA
- Dotted: FFT.
- L = 2560
- As expected from a decimation algorithm, FFT size is critical in both FFT and CTA as seen from FFT spread here.
- Here the raw spread of the CTA (straight lines) can also be seen.



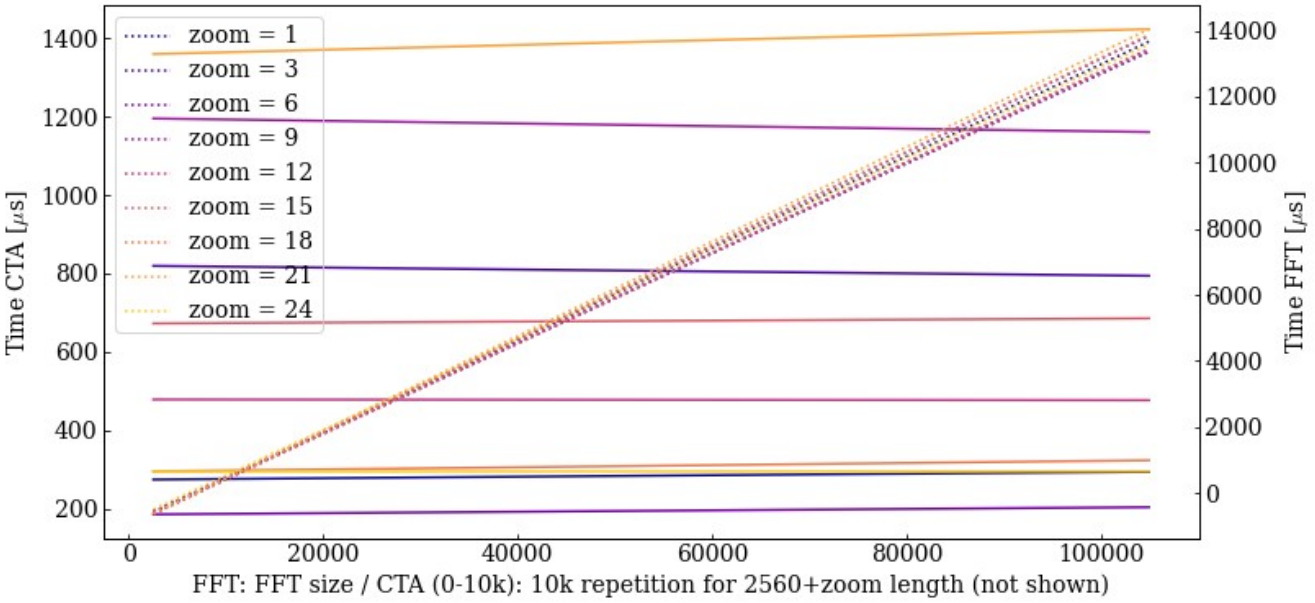
Gradient of linear fits

- Reproducible computer behaviour on average.
- Line: CTA.
- Dotted: FFT.
- L = 256.
- About 13% spread in FFT gradient.
- About the same deviation for the CTA, which should be ideally constant.
- Especially visible in CTA: the different zooms can take a very different time to run (10k repetition of the same calculation). This is because of the behaviour of the underlying FFTs.
- If a better partitioning exists, a slightly longer FFT will run *much* faster – here up to 5x.



Gradient of linear fits

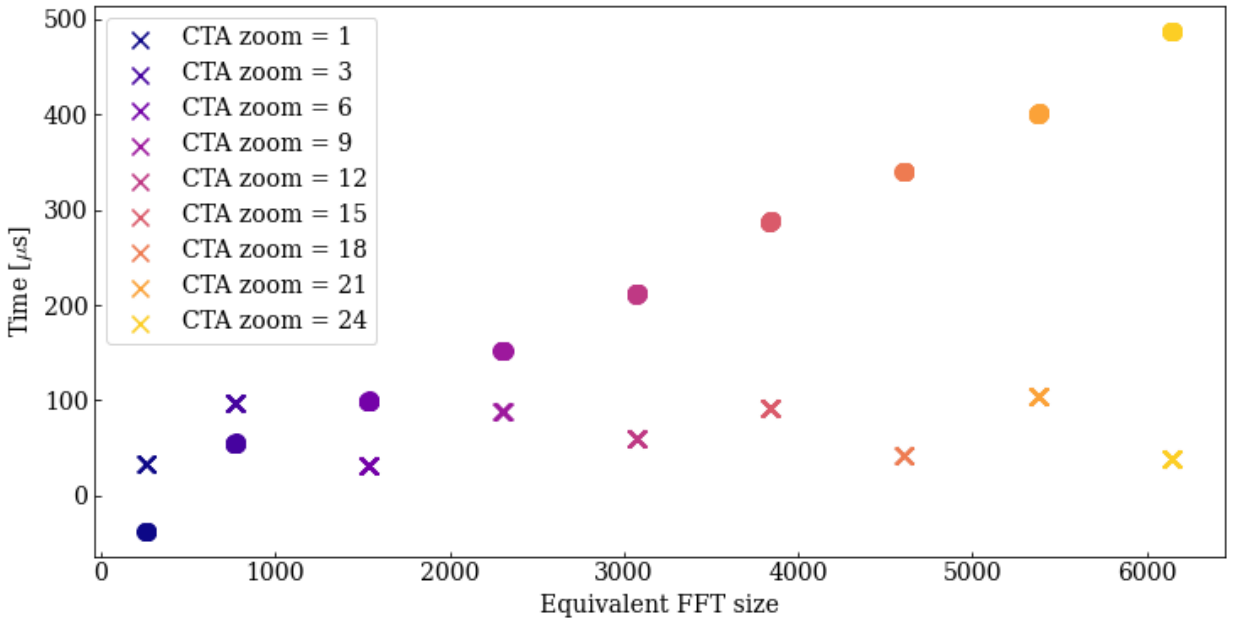
- Line: CTA.
- Dotted: FFT.
- L = 2560.
- Spreads tighten with larger FFT and CTA sizes as size and transient effect become less relevant.



Equivalent FFT size for a given CTA zoom

CTA: x

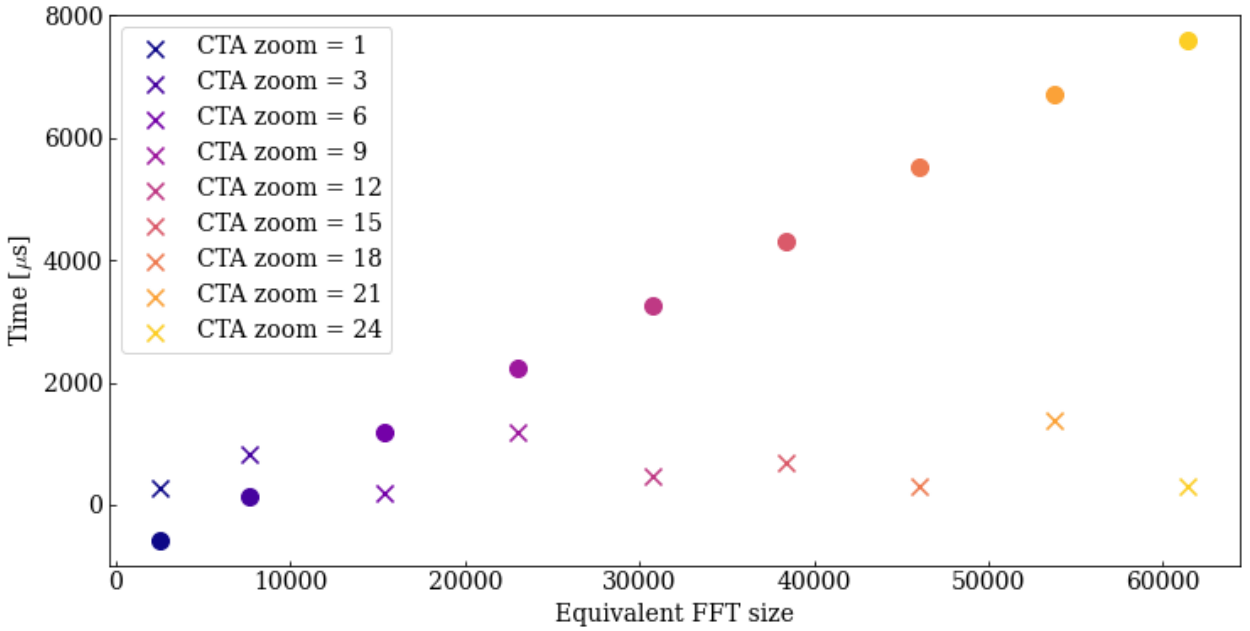
- FFT: o
- L = 256
- As expected from the algorithm, the larger the zoom, the faster the CTA is compared to an equivalent FFT.
- The larger spread in CTA is due to systematics, much larger than reproducibility.
- The reason is the FFT sizes. The base signal length is 256, and CTA does not need a lot more points for a much larger zoom, where some FFT sizes are much faster than others. But the trend is representative of bigger sizes, too.
- Hence a zoom 24 can be almost as fast as a zoom 6...



Equivalent FFT size for a given CTA zoom

CTA: x

- FFT: o
- L = 2560
- As expected from the algorithm, the larger the zoom, the faster the CTA is compared to an equivalent FFT.
- Even better seen that the larger spread in CTA is due to systematics, because the algorithm scales exceptionally well with zoom, therefore the effects of the underlying implementation are seen.



Conclusions

- Existing Python and C / C++ implementations of the CZT were surveyed, there are not many.
- The new inverse is only implemented in Python and it is still at the early stages of development, but it is in active development.
- A test to compare the performance of the forward CZT with the FFT in numpy was carried out.
- On average, the algorithms perform as expected from theory.
- The running time – as expected – is highly sensitive (up to 5x) to the precise length of the FFT for a given approximate length.

Next steps

- For a fair benchmark a proper ICZT with C performance is to be implemented.
  - Idea: use numba for all calculations.
  - With that ICZT can be implemented fast and the rest of the calculations will use the same optimisation, too.
  - Multi thread capability is promised out of the box, not tested.
- Then compare performance in BLonD.