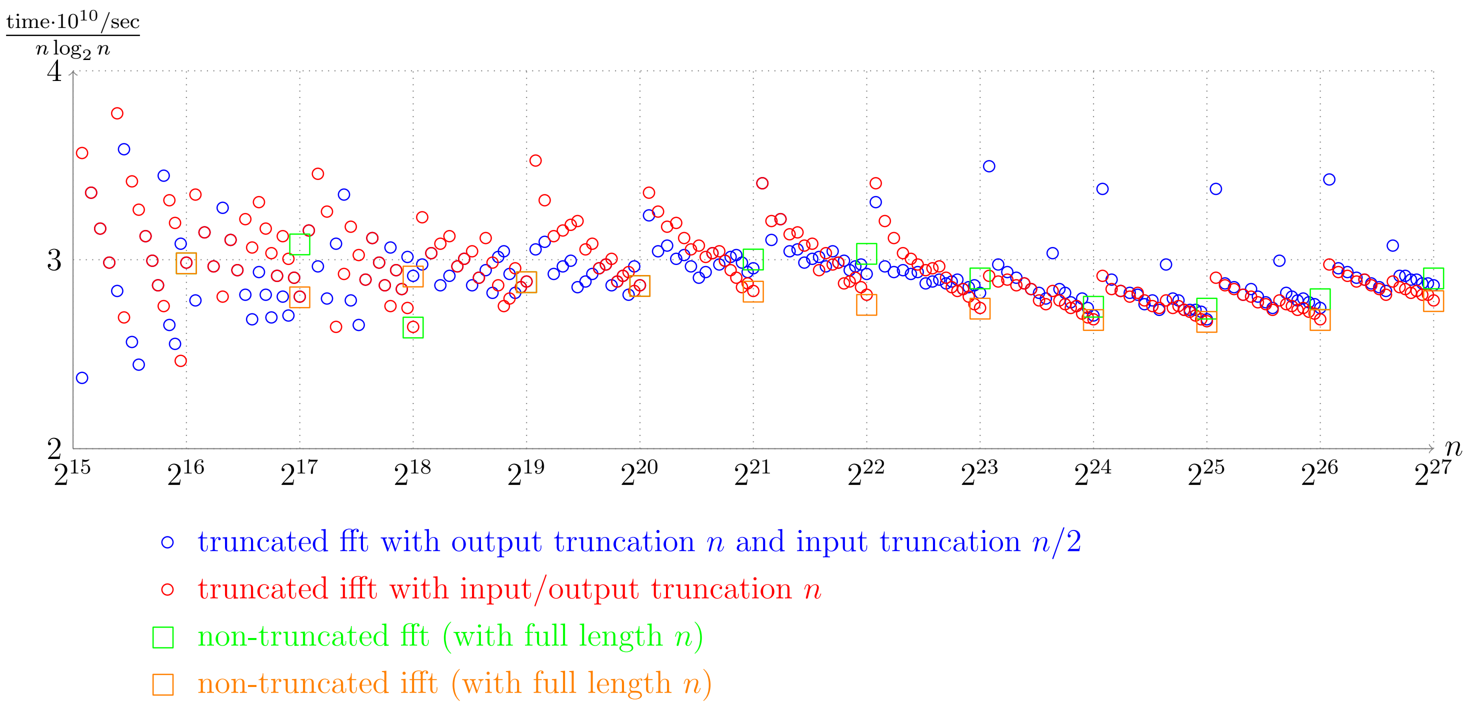
1. FFT/IFFT

2. TRUNCATION

Since zero-padding the input and output data to the next power of two obviously leads to performance jumps at powers of two, a *truncated fft* is necessary, which assumes certain portions of the input and output are zero. If the convolution length is denoted by n we expect

$$\text{runtime} \sim n \log n,$$

so the more constant the ratio is, the better the truncation is working. Below is a plot of such a ratio when the input is further truncated to length $n/2$. which corresponds to assuming the top half of the n inputs are zero. As expected, the truncated fft/ifft performs worst right after a power of two, but slightly unexpected is how bad the fft is there after $n > 2^{23}$ and how well the ifft is there. There are performance bumps at $3 \cdot 2^n$, and, rather surprisingly, the non-truncated ifft is eventuallybeating the non-truncated fft by about 7%. This is surprising for the non-truncated version because they are performing the exact same calculations, just in a different order.



Same picture for {i}fft_mfa_truncate_sqrt2 with 2^{16} bit coefficients to support the final long convolution length of 2^{18}.