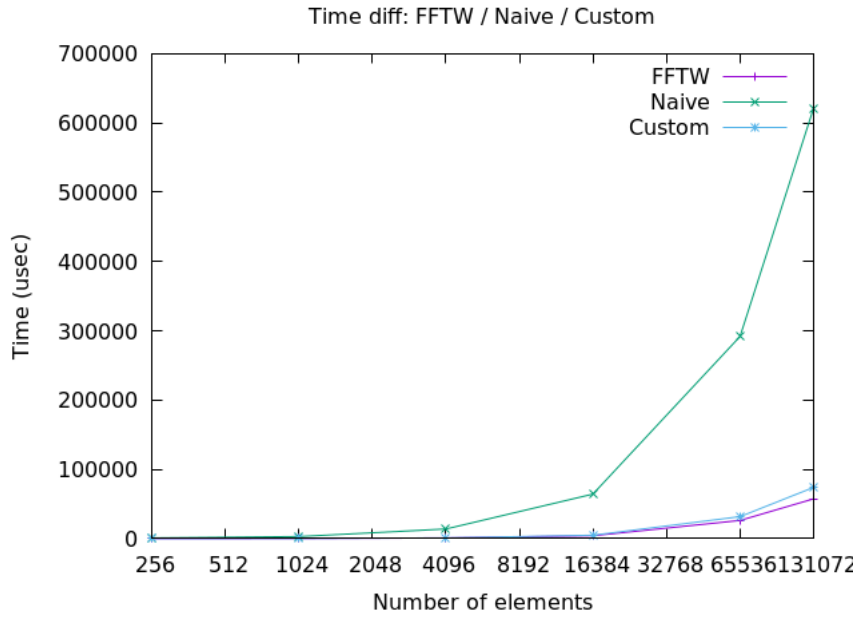
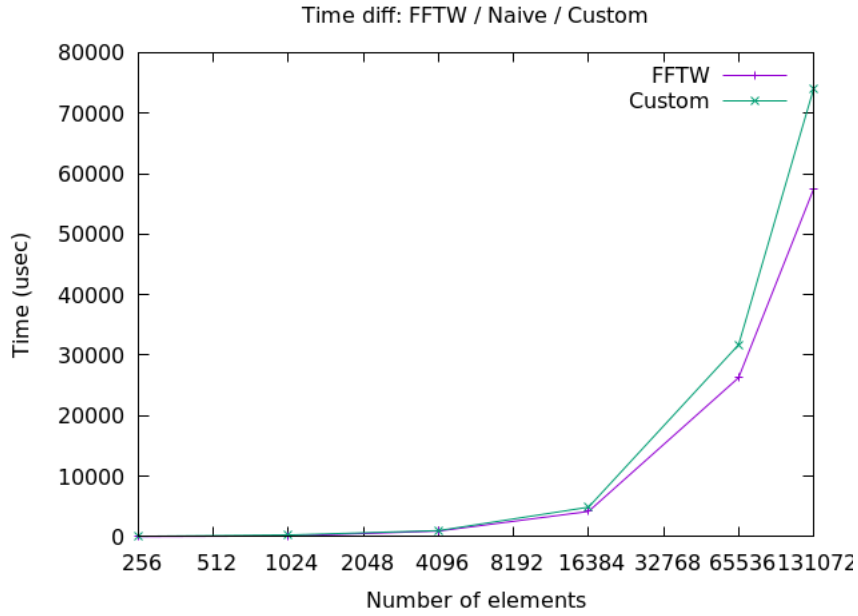
Optimalisering av FFT

Første jeg så på var algoritmen brukt i den naive implemtasjonen. Den naive implemtasjonen har implementert Cooley-Tukey algoritmen ved dybde-først rekursjon. Jeg viste at hvis det var mulig å omskrive algoritmen om til bredde-først iterasjon ville dette ha flere fordeler. Ved å løse problemet iterativt blir plass kompleksiteten $O(1)$, siden algoritmen blir in-place. Videre slipper man mulig overhead/branching som regel medfølger med rekursjon. Heldigvis er det et helt kapittel kun om effektive fft algoritmer i boka "Introduction to Algorithms", som inkluderer pseoudokode for en iterativ versjon av Cooley-Tukey.

Etter å ha implementert den iterative versjonen var forbedringen klare. Den naive implementasjonens kjøretid vokser dramatisk for de større inputene, mens den iterative versjonen har mye mer stabil vekst. En annen forbedring jeg gjorde var å prekomputere alle twiddle faktorene for hver iterasjon. Dette ga ca 5% forbedring på kjøretiden. Den siste forbedringen jeg gjorde var å endre kompelleringsflaggene til å utnytte mest mulig optimalisering. Dette gjorde overaskende mye forskjell, kjøretiden halverte seg nesten. Jeg prøvde også å parallisere algoritmen, men fant ingen måte å både ha algoritmen parallel og iterativ.




Time diff: FFTW / Naive / Custom
Time (usec)
80000
70000
60000
50000
40000
30000
20000
10000
0
256
512
1024
2048
4096
8192
16384
32768
65536
131072
Number of elements
FFTW
Custom