IN2060 - Obligatorisk innlevering 4, rapport 26/10/2020
Jørgen Lunder, William Resvoll Skaug

Oppgaveteksten nevner fire eksempler på metoder som vil effektivisere algoritmen. Dette er ved hjelp av Stride-parameter, bruk av cache, rekursjon, eller threads. Disse punktene vil på sitt vis gjøre prosessene raskere ved å minimere tiden de bruker eller ved threads, hvor man kan kjøre flere prosesser parallelt for å bli ferdig før.

Vi har implementert en stride-parameter,som er en ekstra parameter i det rekursive funksjonskallet. For hvert nye kall dobles Stride-parameteret, og in-pekeren i det andre fft-kallet vil legges til Stride-verdien.

Hvordan skal dette implementeres (forklar kort uten kode)
Vi fulgte et eksempel fra Cooley-Tukey FTT-algoritmen sin wiki, og slavisk implementerte Stride-parameteret inn i den oppgitte kildekoden. Her fjernet vi også oppdelingen til oddetall og partall, da vi fant ut at det ikke var nødvendig. Et nytt parameter er lagt til i de rekursive funksjonskallet, men det blir satt av seg selv når den blir kalt første gangen, så trengs ikke å endres i “hovedprogram”.
https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm

Hvordan svarer den faktiske kjøretiden til forventningene på forhånd?
Resultatet ved stride ble kjøretiden ca 20-30% raskere en den opprinnelige koden. Vi trodde ikke den skulle gjøre programmet så mye raskere. fordi den må fortsatt inn i minne og hente verdiene, men den slipper å lagre den på nye plasser hver gang.

Konklusjonen er at å utføre en vanlig DFT-algoritme kan være O(N^2) i tidskompleksitet. FFT kan redusere tidskompleksiteten ned til O(N*log(N)). Dette stemmer ganske godt overens med vår besparelse ved hjelp av Stride-parameteret. Graf over tiden ligger under.

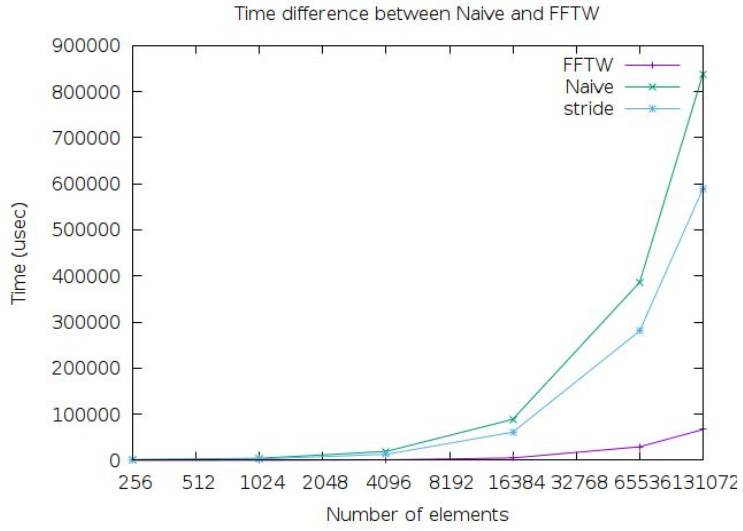
Teoretisk, andre metoden Threads
Ideen om å fordele prosesser mellom flere tasks er at dersom du har fire threads kan den totale tiden gå opp mot fire ganger så fort. Dette fungerer ikke helt i praksis, da det krever en del ekstra tid for å utføre det tekniske for å dele opp prosessene til trådene.

Et annet problem med threads vi fant i boken(kap. 7.7.7) er at det er ikke alltid alle prosessene kan løses parallelt, og dermed må vente på hverandre. Threads kan også bruke mye tid i minne pga begrenset kapasitetet i cache/ ram mengde.

Vi har ikke implementert multithreading, men vet at andre grupper har gjort dette. Dette ga ikke en stor økning i hastighet.

Vårt program og tidsbesparelser:

Tiden regneoperasjonene tok med naiv sammenlignet med bruk av Stride-parameter.



Vår programkode:

```
#include "fft.h"
#include <math.h>
#include <stdlib.h>

void fft_compute(const complex* in, complex* out, const int n) {
	fft(in, out, n, 1);
}

void fft(const complex* in, complex* out, const int n, const int s){
	if(n == 1) {
	out[0] = in[0];
	}
	else {
		const int half = n / 2;
		fft(in, out, half, 2*s);
		fft(in+s, out+half, half, 2*s);

		for(int i = 0; i < half; ++i) {
				const complex e = out[i];
				const complex o = out[i + half];
				const complex w = cexp(0 - (2. * M_PI * i) / n * I);
				out[i]	= e + w * o;
				out[i + half] = e - w * o;
		}
	}
}
```