Introduction

* Name and objective

Outline

* Outline main topics of discussion

Continuous Fourier Transforms

* Continuous Fourier transform
* The input x, is a continuous-time signal
* Context of embedded systems and digital signal processing
* Sampling interval, the sampling rate, and quantization
* Quantized discrete-time signal - > DFT

Discrete Fourier Transform

* Integration is limit of a Riemann Sum
* Input discrete-time signal, x, array of length N
* Input signal may be complex
* The output, y = signal in the frequency domain, array of length N
* May also be complex.
* Computation for some element of y, sum N complex terms…
* Problem at hand.

DFT Implementation

* Co-op student telecom company
* Implement generic DFT function in C/C++
* Simple and elegant with nested for loops.
* Assume complex exponential in constant time.
* Unit tests for error tolerance
* Board testing goes down south.
* Function was too slow
* Runtime is O(n2)
* Need better runtime
* Not obvious how to optimize code for speed
* Must do with complex multiplication
* Take advantage of the periodicity and symmetry of complex exponentials (twiddle factors)

Twiddle Factors

* Complex factors in DFT equation.
* Phasors on unit circle
* Correspond to N-th roots of unity
* N equally spaced points along the unit circle
* Show gif

Notation Change

* Let wdenote the .
* Superscript as twiddle index. call the twiddle index. We can think of the set of twiddle factors as a hash table of N el
* Subscript denoting set of roots of unity.
* Think of set of N unique twiddle factors as hash table

Periodicity of Twiddle Factors

* Example with 8th roots of unity
* Incrementing twiddle index by 1 goes clockwise.
* Incrementing to 8 gets us back where we started
* Further incrementing takes us in a circle, again
* Mathematically, this property is expressed as follows
* Periodic nature reduces complex multiplication

Symmetry of Twiddle Factors

* Example with 8th roots of unity.
* Offsetting index by N/2 indices results in the negation
* Assuming that N is even
* Mathematically represented as follows
* Symmetrical nature reduces complex multiplications
* Negating has no significant cycle count overhead

FFTs

* Many FFT algorithms
* Examine Cooley-Tukey’s Radix-2 Decimation in Time Fast Fourier Transform
* Radix-2: divide and conquer approach, recursively break down into two subproblems
* Achieving runtime of nlog2(n)

FFT Derivation

* Recall the mathematical equation without nested loops
* Divide and conquer approach, for two smaller summations
* Even indexed terms, starting at 0
* Odd indexed terms
* Moved the 2 from numerator to denominator
* Common factored
* N-point DFT, that is composed of two N/2-point DFTs
* Introduce Ek and Ok are results of a smaller DFT from the previous stage

Apply Periodicity

* Ek is result from a N/2 point DFT, which uses the N/2 th roots of unity
* Adding offset of N/2 to k has no affect
* Full way around circle.
* Equality in red
* Implies the following
* Same argument for Ok.

Magical Step

* We can calculate the result for one element of y.
* Another element of y, namely, at some index k + N/2
* Periodicity substitution
* Symmetry substitution, so a negation

2-point Butterfly

* Intermediate results Ek and Ok­ used to compute not one, but two elements of y
* By periodicity and symmetry
* Called the 2-point butterfly
* One complex multiply, and two complex additive operations
* That product is only computed once
* Use a local variable
* Ek and Ok­ from earlier DFT, therefore are outputs of butterflies as well.
* Recursion applied until the base case, a 2-point DFT

8-point (DIT) FFT Signal Flow Chart

* Summarized with flow chart
* Describes the computations required for each output element
* Few notes to make here, full interpretation/understanding is not required
* Number of stages
* Left to right
* Butterflies per stage

FFT Operations Summary

Conclusion

* Therefore, by taking advantage of the periodicity and symmetry of twiddle factors, we reduced the runtime of the DFT from O(n2) to O(nlogn), that’s amazing.
* Just to further emphasize the importance it all, here’s a quote from Gilbert Strang of MIT, on FFTs, calling it “the most important numerical algorithm of our lifetime”. Truly, one of the biggest disruptions in the 20th century.
* Keep in mind that when it comes to algorithm selection, runtime is not the only aspect of performance to compare. Naturally, a programmer would also want to examine space complexity, function size, and accuracy. But these are not within the scope of this TPM.
* Thank you, any questions?