ECE 421
Introduction to Signal Processing

Dror Baron
Assistant Professor
Dept. of Electrical and Computer Engr.
North Carolina State University, NC, USA
Fast Fourier Transform

[Reading material: Chapter 8]
Motivation

- We’ve discussed that convolution is very useful (filtering)

- Recall multipath example
  - 100M samples/sec
  - 1000 taps
  - 100B multiplications/sec
  - 100B additions/sec

- Filtering can be performed as $y = \text{IDFT}(\text{DFT}(x) \cdot \text{DFT}(h))$
  - DFT($h$) can be pre-computed (once)
  - Computing IDFT & DFT analogous (minus signs in exponents)

- How fast can we compute DFT?
Naïve approach

- Recall $X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N}$

- Matlab:
  
  ```matlab
  N=2000; % signal length
  x=randn(N,1); % random input
  xf=zeros(N,1); % initialize DFT coeffs
  tic % start clock
  for k=0:N-1 % loop over k
    xf(k+1)=exp(-j*2*pi*k*(0:N-1)/N)*x; % computes everything one line
  end;
  toc
  
  ```

- Complicated line: inner product between row $\exp(...)$ column $x$
Why is it slow?

- Main line runs N times
  - N complex multiplications
  - N-1 complex additions
  - N complex exponentials
  - More real-valued arithmetic (set up exponents)

- Quadratic computation
What’s wrong with quadratic runtime?

Let’s compare $N^2$ (DFT runtime) with $N \times \log_2(N)$ for FFT

<table>
<thead>
<tr>
<th>$N$</th>
<th>$N^2$</th>
<th>$N \log_2(N)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^3$</td>
<td>$10^6$ (feasible 1960s)</td>
<td>$\sim 10^4$</td>
</tr>
<tr>
<td>$10^6$</td>
<td>$10^{12}$ (feasible 2000s)</td>
<td>$\sim 2 \times 10^7$</td>
</tr>
<tr>
<td>$10^9$</td>
<td>$10^{18}$ infeasible</td>
<td>$\sim 3 \times 10^{10}$</td>
</tr>
</tbody>
</table>

Reminder about computer speeds:

– 1960s: dozens to thousands operations/second
– 1980s: millions
– 2000s: billions
– Modern general purpose graphics processing units: 100x
Divide and Conquer
How can we improve DFT speed?

- **Divide** into blocks
- Compute DFT of each block (**conquer**)
- Merge - “book keeping” to patch together DFT’s of blocks
  - Merging step needs to be fast (linear in N)

- **Trick:** use divide & conquer hierarchically within blocks

```
DFT(N)  
DFT(N/2) DFT(N/2)  
DFT(N/4) DFT(N/4) DFT(N/4) DFT(N/4)
```
Example: Sorting

- Let’s see another (simpler) $N \times \log_2(N)$ algorithm

- Want to *sort* set of numbers, $x = \{1, 9, -2, 3, 6, -1, 7, 4\}$
  - Step1: partition into $x_1 = \{1, 9, -2, 3\}$ and $x_2 = \{6, -1, 7, 4\}$
  - Step2: sort each component
    - $\text{Sort}(x_1) = \{-2, 1, 3, 9\}$
    - $\text{Sort}(x_2) = \{-1, 4, 6, 7\}$
  - Merging step:
    - $-2 < -1 \rightarrow \text{Sort}(x) \leftarrow \{-2\}$, $\text{Sort}(x_1) \leftarrow \{1, 3, 9\}$
    - $-1 < 1 \rightarrow \text{Sort}(x) \leftarrow \{-2, -1\}$, $\text{Sort}(x_2) \leftarrow \{4, 6, 7\}$
    - $1 < 4 \rightarrow \text{Sort}(x) \leftarrow \{-2, -1, 1\}$, $\text{Sort}(x_1) \leftarrow \{3, 9\}$
    - ...

- Can see that merging requires runtime linear in $N$
- “Mergesort” requires $N \times \log_2(N)$ runtime
How fast is divide & conquer?

- Denote runtime of size-n problem by $t(n)$
- Suppose merge($N$) takes $C_1 N$ time

- **Result:** $t(N) \leq C_2 N \cdot \log_2(N)$
- **Proof:** (induction on $N$)
  - Basis case: take $N=2$ or $4$...
  - Inductive step: we assume $t(N) \leq C_2 N \cdot \log_2(N)$
  - Want to show $t(2N) \leq C_2 2N \cdot \log_2(2N)$
    - $t(2N) \leq 2t(N) + C_1 N \leq 2C_2 N \log_2(N) + C_1 N = C_2 N[2 \log_2(N) + \frac{C_1}{C_2}]$
  - Suppose $C_2 \geq C_1$ (else increase $C_2$)
    - $t(2N) \leq C_2 N[2 \log_2(N) + 1] < C_2 N[2 \log_2(N) + 2]$
    - $= 2C_2 N[\log_2(N) + 1] = 2C_2 N \log_2(2N)$
DFT Using Divide and Conquer
How to apply divide & conquer

- Partition $N$ into $L \times M$
- Compute DFT for $L$ blocks, each of length $M$
  - Can compute hierarchically by partitioning $M$ further
- Merge $L$ blocks

- Runtime will be proportional to $N \cdot \log_2(N)$

- If $N$ doesn’t factor nicely into prime numbers, can zero-pad

- Example Matlab implementations on course webpage
How to “merge” DFT’s in linear time?

- Let’s relate length-\( N \) DFT to \( L \) length-\( M \) DFTs (linear time)
- Take \( N = L \cdot M \), \( n = l + mL \), \( k = Mp + q \)

\[
X(p, q) = X(k = Mp + q) = \sum_{n=0}^{N-1} x(n) w_N^{kn} = \\
\sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(n = l + mL) w_N^{(Mp+q)(l+mL)} = \\
\sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l, m) w_N^{(Mp+q)(l+mL)}
\]

- Let’s rewrite \( w \) term:

\[
w_N^{(Mp+q)(l+mL)} = w_N^{Mpl+ql+MpmL+qml} = \\
w_N^{Mpl} w_N^{ql} w_N^{MpmL} w_N^{qml} = \\
w_L^{pl} w_N^{ql} w_M^{qm} = w_L^M w_N^L = w_M^L = 1
\]
How to “merge” DFT’s continued

- We now have:

\[ X(p, q) = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l, m) w_N^{(Mp+q)(l+mL)} \]

\[ = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l, m) w_L^{pl} w_N^{ql} w_M^{qm} \]

\[ = \sum_{l=0}^{L-1} w_L^{pl} \{ w_N^{ql} \sum_{m=0}^{M-1} x(l, m) w_M^{qm} \} \]

\[ = \sum_{l=0}^{L-1} w_L^{pl} \{ w_N^{ql} DFT_M(x(l,\cdot)) \} \]

- Conclusion:

1) Compute L DFT’s, each of M points
2) Modulate by \( w_N^{ql} \) (this is new – required for merge)
3) Compute M DFT’s, each of L points (made fast w/small L)