Outline

1. Introduction
2. Algorithm
3. Experiments
The Discrete Fourier Transform

- Discrete Fourier transform: given \( x \in \mathbb{C}^n \), find

\[
\hat{x}_i = \sum x_j \omega_{ij}
\]

- Fundamental tool
  - Compression (audio, image, video)
  - Signal processing
  - Data analysis
  - ...

- FFT: \( O(n \log n) \) time.

**Sparse Fourier Transform**

- Often the Fourier transform is dominated by a small number of "peaks"
  - Precisely the reason to use for compression.
- If most of mass in $k$ locations, can we compute FFT faster?
Previous work

- Boolean cube: [KM92], [GL89]. What about $\mathbb{C}$?
- [Mansour-92]: $k^c \log^c n$.
- Long list of other work [GGIMS02, AGS03, Iwen10, Aka10]
- Fastest is [Gilbert-Muthukrishnan-Strauss-05]: $k \log^4 n$.
  - All have poor constants, many logs.
  - Need $n/k > 40,000$ or $\omega(\log^3 n)$ to beat FFTW.
  - Our goal: beat FFTW for smaller $n/k$ in theory and practice.
  - Result: $n/k > 2,000$ or $\omega(\log n)$ to beat FFTW.
Our result

- Simple, practical algorithm with good constants.
- Compute the $k$-sparse Fourier transform in $O(\sqrt{kn} \log^{3/2} n)$ time.
- Get $\hat{x}'$ with approximation error
  
  $$\|\hat{x}' - \hat{x}\|_\infty \leq \frac{1}{k} \|\hat{x} - \hat{x}_k\|_2$$

- If $\hat{x}$ is sparse, recover it exactly.
- Caveats:
  - Additional $\|x\|_2/n^{\Theta(1)}$ error.
  - $n$ must be a power of 2.
If $\hat{x}$ is $k$-sparse with known support $S$, find $\hat{x}_S$ exactly in $O(k \log^2 n)$ time.

In general, estimate $\hat{x}$ approximately in $\widetilde{O}(\sqrt{nk})$ time.
Intuition

$n$-dimensional DFT of first $B$ terms.

$B$-dimensional DFT of first $B$ terms.
“Hashes” into $B$ buckets in $B \log B$ time.

**Issues:**

- “Hashing” needs a random hash function
  - Access $x'_t = \omega^{-at}x_{\sigma t}$, so $\hat{x}'_t = \hat{x}_{\sigma^{-1}t+a}$ [GMS-05]
- Collisions
  - Have $B > 4k$, repeat $O(\log n)$ times and take median. [Count-Sketch, CCF02]
- Leakage
- Finding the support. [Porat-Strauss-12], talk at 9:45am.
Let $F_i = \begin{cases} 1 & i < B \\ 0 & \text{otherwise} \end{cases}$ be the “boxcar” filter. (Used in [GGIMS02,GMS05])

Observe

\[
\text{DFT}(F \cdot x, B) = \text{subsample}(\text{DFT}(F \cdot x, n), B) = \text{subsample}(\hat{F} \ast \hat{x}, B).
\]

\[
\text{DFT} \hat{F} \text{ of boxcar filter is sinc, decays as } 1/i.
\]

Need a better filter $F$!
Observe subsample $(\hat{F} \ast \hat{x}, B)$ in $O(B \log B)$ time.

Needs for $F$:
- $\text{supp}(F) \in [0, B]$
- $|\hat{F}| < \delta = 1/n^{\Theta(1)}$ except “near” 0.
- $\hat{F} \approx 1$ over $[-n/2B, n/2B]$. 

Gaussians:
- Standard deviation $\sigma = B/\sqrt{\log n}$
- DFT has $\hat{\sigma} = (n/B)\sqrt{\log n}$
- Nontrivial leakage into $O(\log n)$ buckets.
- But likely trivial contribution to correct bucket.
Observe subsample \((\hat{F} \ast \hat{x}, B)\) in \(O(B \log B)\) time.

Needs for \(F\):
- \(\text{supp}(F) \in [0, B]\)
- \(|\hat{F}| < \delta = 1/n^{\Theta(1)}\) except “near” 0.
- \(\hat{F} \approx 1\) over \([-n/2B, n/2B]\).

Gaussians:
- Standard deviation \(\sigma = B/\sqrt{\log n}\)
- DFT has \(\hat{\sigma} = (n/B)\sqrt{\log n}\)
- Nontrivial leakage into \(O(\log n)\) buckets.
- But likely trivial contribution to correct bucket.
Observe subsample($\hat{F} \ast \hat{x}, B$) in $O(B \log B)$ time.

Needs for $F$:

- $\text{supp}(F) \in [0, B \log n]$
- $|\hat{F}| < \delta = 1/n^{\Theta(1)}$ except “near” 0.
- $\hat{F} \approx 1$ over $[-n/2B, n/2B]$.

Gaussians:

- Standard deviation $\sigma = B/\sqrt{\log n} B \cdot \sqrt{\log n}$
- DFT has $\hat{\sigma} = (n/B) \sqrt{\log n} (n/B)/\sqrt{\log n}$
- Nontrivial leakage into 0 buckets.
- But likely trivial contribution to correct bucket.
Let $G$ be Gaussian with $\sigma = B \sqrt{\log n}$.

$H$ be box-car filter of length $n/B$. 
Let $G$ be Gaussian with $\sigma = B \sqrt{\log n}$

$H$ be box-car filter of length $n/B$.

Use $\hat{F} = \hat{G} \ast H$.

- $F = G \cdot \hat{H}$, so $\text{supp}(F) \subset [0, B \log n]$.
- $|\hat{F}| < 1/n^{\Theta(1)}$ outside $-n/B, n/B$.
- $|\hat{F}| = 1 \pm 1/n^{\Theta(1)}$ within $n/2B, n/B$.

Hashes correctly to one bucket, leaks to at most 1 bucket.

Replace Gaussians with “Dolph-Chebyshev window functions”: factor 2 improvement.
Algorithm to estimate $\hat{x}_S$

- For $O(\log n)$ different permutations of $\hat{x}$, compute subsample$(\hat{F} \ast \hat{x}, B)$.
- Estimate each $x_i$ as median of values it maps to.
Algorithm to estimate $\hat{x}_S$

- For $O(\log n)$ different permutations of $\hat{x}$, compute subsample($\hat{F} \ast \hat{x}, B$).
- Estimate each $x_i$ as median of values it maps to.
Algorithm to estimate $\hat{x}_S$

- For $O(\log n)$ different permutations of $\hat{x}$, compute subsample$(\hat{F} \ast \hat{x}, B)$.
- Estimate each $x_i$ as median of values it maps to.
Algorithm to estimate $\hat{x}_S$

- For $O(\log n)$ different permutations of $\hat{x}$, compute subsample($\hat{F} \ast \hat{x}, B$).
- Estimate each $x_i$ as median of values it maps to.
Algorithm in general

- For $O(\log n)$ different permutations of $\hat{x}$, compute subsample($\hat{F} \ast \hat{x}, B$).
- Estimate each $x_i$ as median of values it maps to.
Algorithm in general

- For $O(\log n)$ different permutations of $\hat{x}$, compute subsample($\hat{F} \ast \hat{x}, B$).
- Estimate each $x_i$ as median of values it maps to.
- To find $S$: choose all that map to the top $2k$ values.
Algorithm in general

- For $O(\log n)$ different permutations of $\hat{x}$, compute subsample($\hat{F} \ast \hat{x}$, $B$).
- Estimate each $x_i$ as median of values it maps to.
- To find $S$: choose all that map to the top $2k$ values.
- $nk/B$ candidates to update at each iteration: total

\[
(\frac{nk}{B} + B \log n) \log n = \sqrt{nk} \log^{3/2} n
\]

time.
Compare to FFTW, previous best sublinear algorithm (AAFFT).

Offer a heuristic that improves time to $\tilde{O}(n^{1/3}k^{2/3})$.
  ▶ Filter from [Mansour ’92].
  ▶ Can’t rerandomize, might miss elements.

Faster than FFTW for $n/k > 2,000$.

Faster than AAFFT for $n/k < 1,000,000$. 
Empirical Performance: noise

- Just like in Count-Sketch, algorithm is noise tolerant.
Conclusions

- Roughly: fastest algorithm for $n/k \in [2 \times 10^3, 10^6]$.
- Recent improvements [HIKP12b?]
  - $O(k \log n)$ for exactly sparse $\hat{x}$
  - $O(k \log \frac{n}{k} \log n)$ for approximation.
  - Beats FFTW for $n/k > 400$ (in the exact case).