Faster Algorithms for Sparse Fourier Transform

Haitham Hassanieh

Piotr Indyk Dina Katabi Eric Price

MIT

Material from:

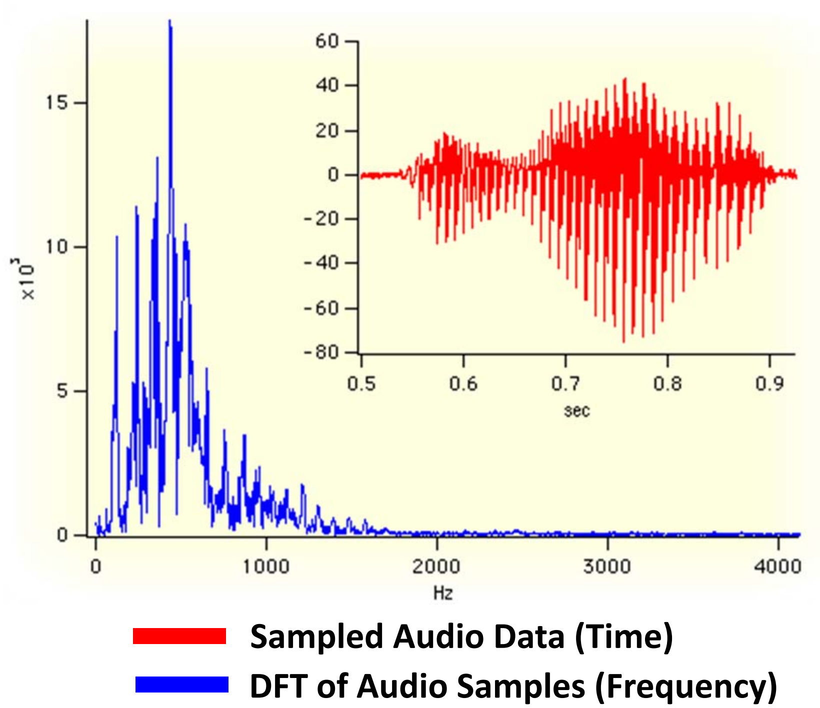
- Hassanieh, Indyk, Katabi, Price, “Simple and Practical Algorithms for Sparse Fourier Transform, SODA’12.
- Hassanieh, Indyk, Katabi, Price, “Nearly Optimal Sparse Fourier Transform”, STOC’12.

The Discrete Fourier Transform

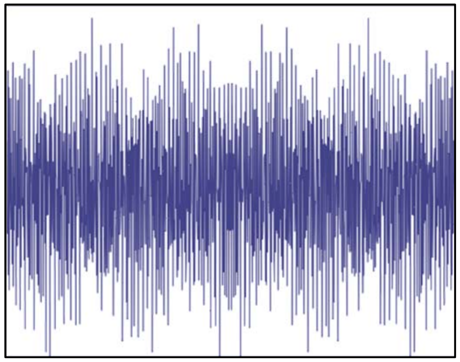
- Discrete Fourier Transform:
  - Given: a signal $\mathbf{x} \in \mathbb{C}^n$
  - Goal: compute the frequency vector $\hat{\mathbf{x}}$ such that for $f \in [1 \ldots n]$:

$$\hat{\mathbf{x}}_f = \sum \mathbf{x}_t \, e^{-\mathrm{i}\, 2\pi \, tf/n}$$

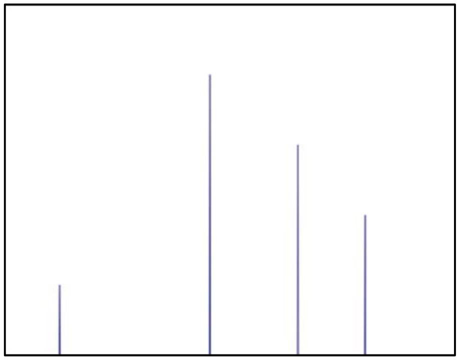
- Fundamental tool:
  - Compression (audio, image, video)
  - Signal processing
  - Data analysis
  - Wireless Communication
  - …
- FFT : $O(n \log n)$ time



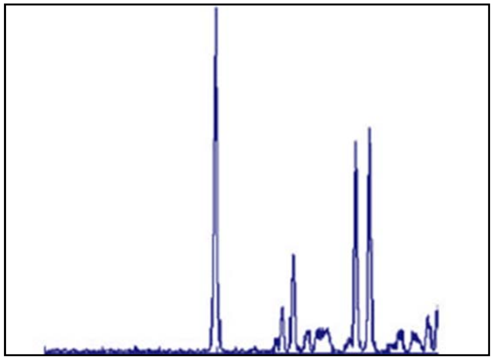
Sparse Fourier Transform



Time Domain Signal



Sparse Frequency Spectrum



Approximately Sparse Frequency Spectrum

- Often the Fourier transform is dominated by a small number of "peaks"
  - Only few of the frequency coefficients are nonzero.
  - An exactly k-sparse signal has only k nonzero frequency coefficients.
  - In practice : approximate a sparse signal using the k largest peaks.
- Problem : Can we recover the k-sparse frequency spectrum faster than FFT?

Previous Work

- Algorithms:
  - Boolean cube : [KM92], [GL89]. What about $\mathbb{C}$?
  - Complex FT: [Mansour-92, GGIMS02, AGS03, GMS05, Iwen10, Aka10]

- Best running time: [GMS05] $O(k \log^4 n)$
  - In theory : Improves over FFT for $n/k \;>>\; \log^3 n$
  - In Practice : Large constants; need $n/k \;>\; 40{,}000$ to beat FFT

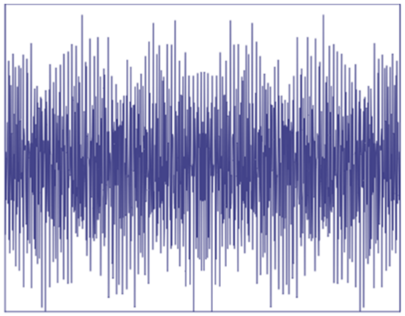
- Goal:
  - Theory: improve over FFT for **all** values of $k = o(n)$
  - Practice: faster runtime than FFT.

Our results

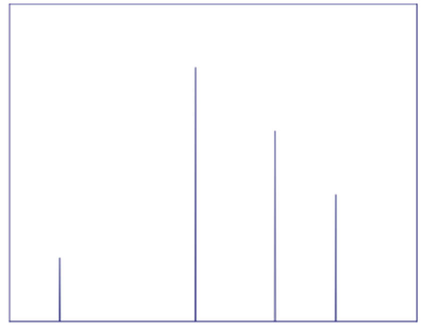
- Randomized algorithms, with constant probability of success
- Exactly k-sparse case, recover $\hat{\mathbf{x}}$: $O(k \log n)$
  - Optimal if FFT optimal
- Approximately k-sparse case, recover $\widehat{\mathbf{x}'}$:
  - Let $\mathrm{Err}_2^k(\hat{\mathbf{x}}) = \min_{k\ sparse\ \hat{\mathbf{x}}_k} \|\hat{\mathbf{x}} - \hat{\mathbf{x}}_k\|_2$
  - l_2/l_2 guarantee $\left\|\widehat{\mathbf{x}'} - \hat{\mathbf{x}}\right\|_2 \le c \times \mathrm{Err}_2^k(\hat{\mathbf{x}})$: $O(k \log(n) \log(n/k))$
    - Improves over FFT for any $k \ll n$
  - l_∞/l_2 guarantee $\left\|\widehat{\mathbf{x}'} - \hat{\mathbf{x}}\right\|_\infty \le \frac{c}{\sqrt{k}} \mathrm{Err}_2^k(\hat{\mathbf{x}})$: $O(\sqrt{nk \log n} \log n)$
    - Improves over FFT for $k \ll n/\log n$

Sparse FFT - Algorithm

Intuition



Time Domain Signal

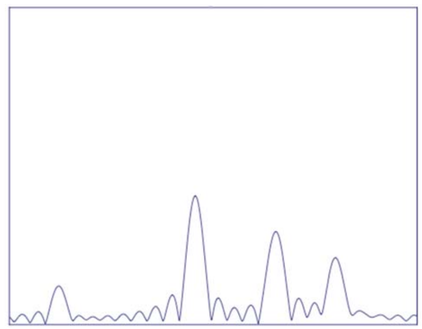


Frequency Domain

n-point DFT : $n \log(n)$

$\mathbf{x} \rightarrow \hat{\mathbf{x}}$

Cut off Time signal

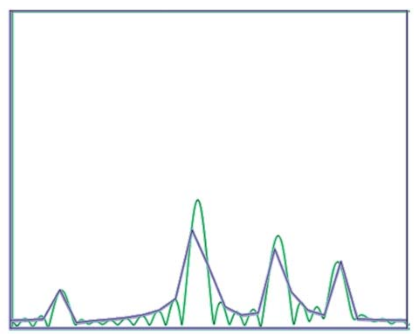


Frequency Domain

n-point DFT of first B terms : $n \log(n)$

$\mathbf{x} \times$ **Boxcar** $\rightarrow$ $\hat{\mathbf{x}} *$ **sinc**

First B samples



Frequency Domain

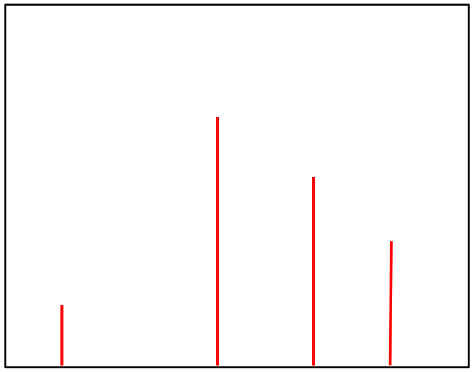
B-point DFT of first B terms: $B \log(B)$

Alias $(\mathbf{x} \times$ **Boxcar**$)$

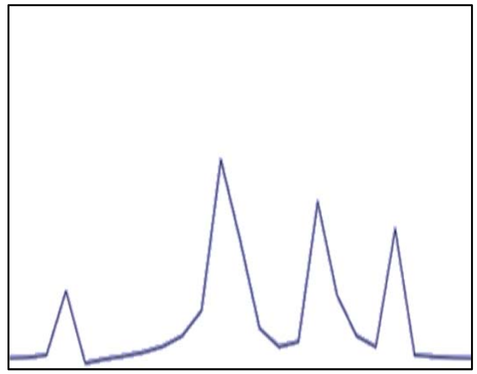
$\downarrow$

Subsample $(\hat{\mathbf{x}} *$ **sinc**$)$

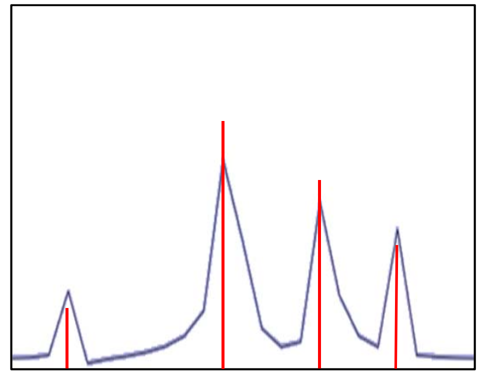
Framework



n-point DFT
of all n samples



B-point DFT
of first B sample

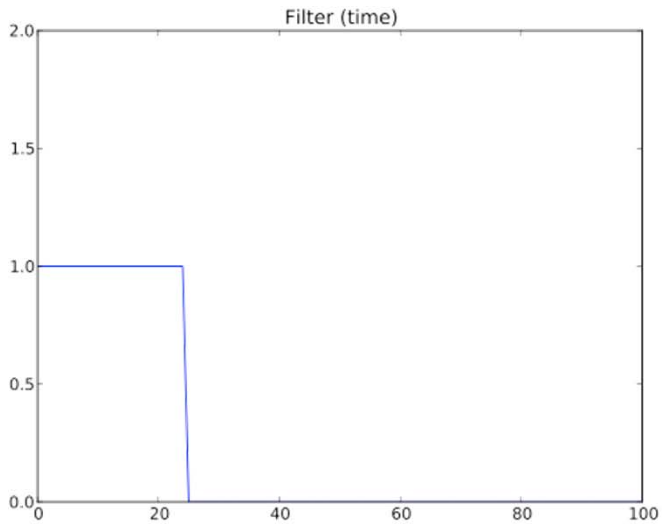


n frequencies hash
into B buckets

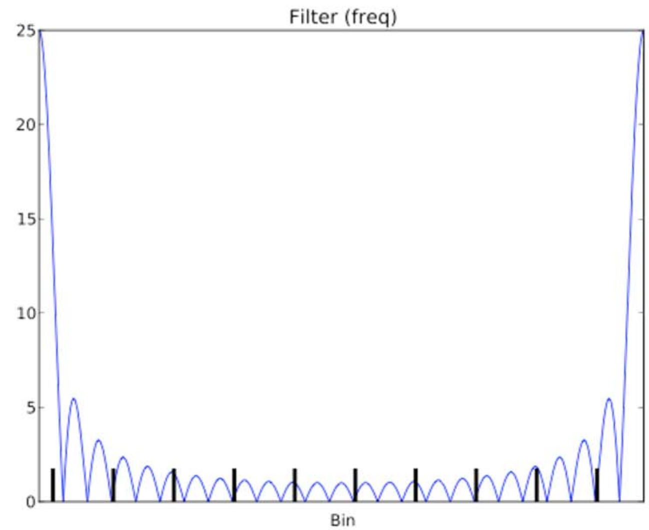
- "Hashes" the n Fourier coefficients into B buckets in $O(B \log B)$ time

- Issues
  - Leakage : **Subsample** $(\hat{\mathbf{x}} * \textbf{Filter})$
  - Given these B buckets, how can we estimate the locations and values the k large frequencies?

Filter: Sinc



F = Boxcar

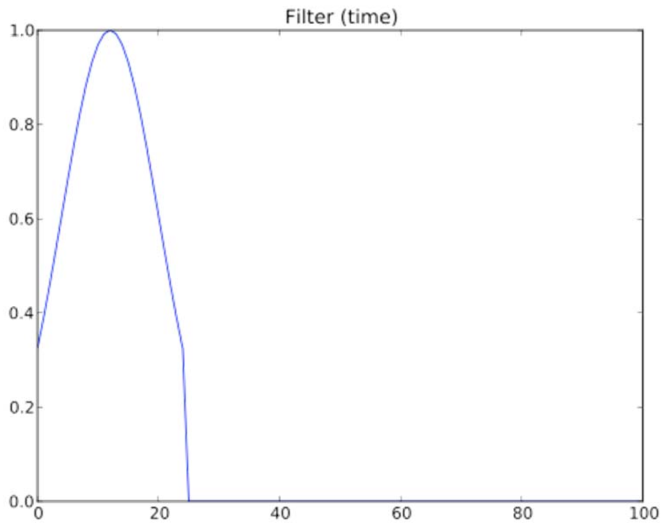


$\hat{F}$ = Sinc

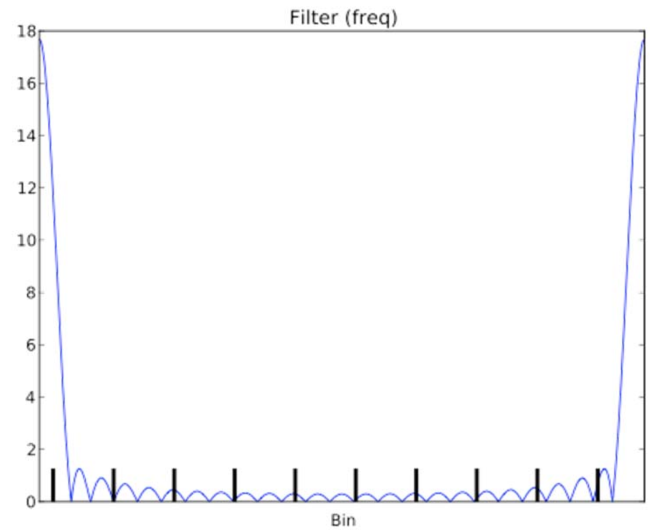
– Polynomial decay

– Leaking many buckets

Filter: Gaussian



F = Gaussian

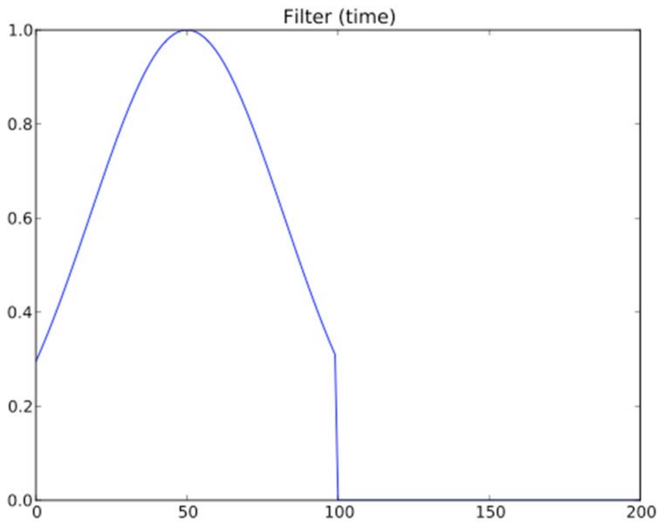


$\hat{F}$ = Gaussian

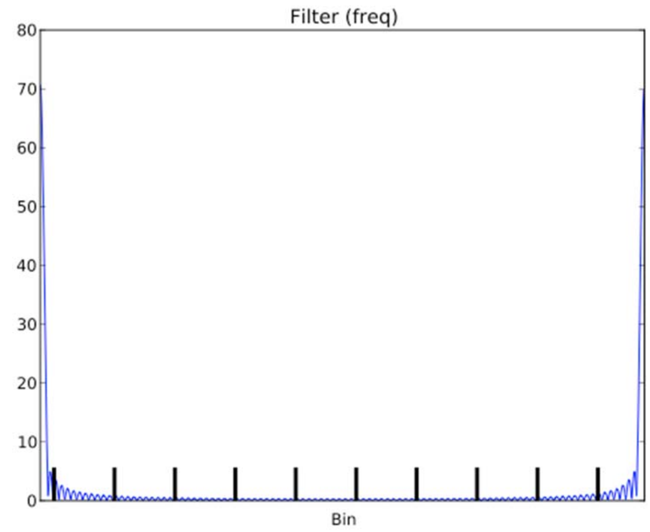
– Exponential decay

– Leaking to $\sqrt{\log n}$ buckets

Filters: Wider Gaussian



F = Wider Gaussian



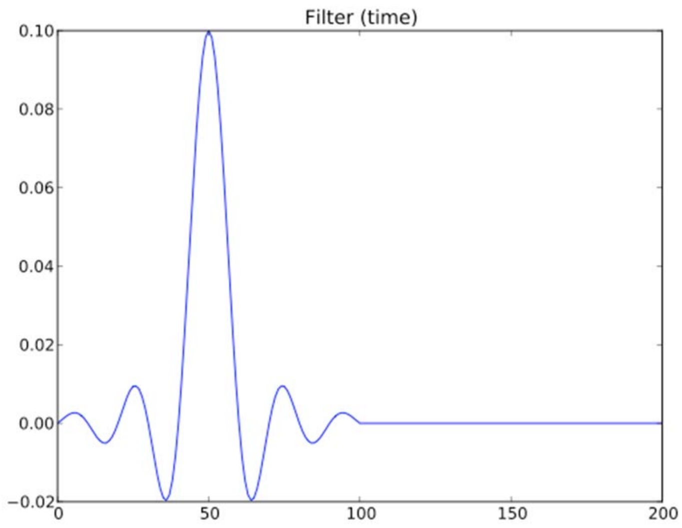
$\hat{F}$ = Narrow Gaussian

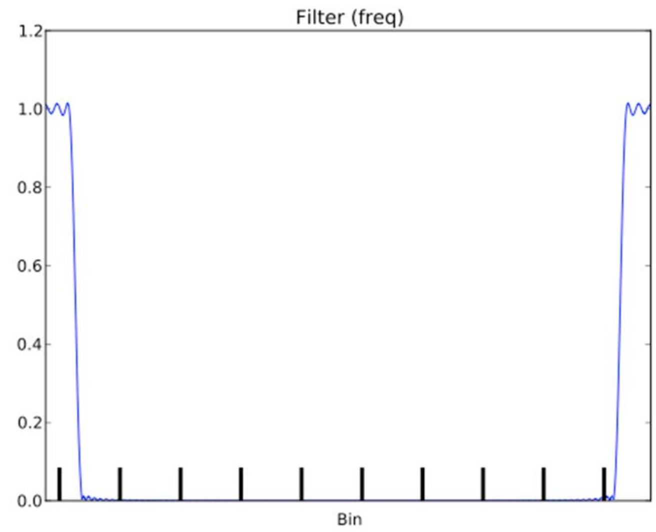
– Exponential decay

– Leaking to <1 buckets

– But trivial contribution to the correct bucket

Filters: Sinc $\times$ Gaussian





$F =$ Sinc × Gaussian

$\hat{F} =$ Boxcar $*$ Gaussian

- Boxcar size n/B: n/B frequencies hash into each bucket
- Still exponential decay
- Leaking to at most 1 bucket
- Sufficient contribution to the correct bucket $[-n/2B, n/2B]$
- Replace Gaussians with “Dolph-Chebyshev window functions”

Finding the support

- $\hat{\mathbf{y}}$ = B-point DFT $(\mathbf{x} \times F)$ = Subsample $\left(\hat{\mathbf{x}} * \widehat{F}\right)$

- Assume no collisions:
  - At most one large frequency hashes into each bucket.
  - Large frequency f_1 hashes to bucket b_1 :

$$\hat{\mathbf{y}}_{b_1} = \hat{\mathbf{x}}_{f_1} \times \widehat{F}_{\Delta} + \text{~~leakage~~}$$

  - Recall: DFT($\mathbf{x}^{\tau}$) $=$ $\hat{\mathbf{x}} \times e^{-\mathrm{i}\,2\pi\,\tau f/n}$

  - $\hat{\mathbf{y}}^{\tau}$ = B-point DFT $(\mathbf{x}^{\tau} \times F)$:

$$\hat{\mathbf{y}}^{\tau}{}_{b_1} = \hat{\mathbf{x}}_{f_1} \times e^{-\mathrm{i}\,2\pi\,\tau f_1/n} \times \widehat{F}_{\Delta} + \text{~~leakage~~}$$

Finding the support

- $\hat{\mathbf{y}}$ = B-point DFT $(\mathbf{x} \times F)$ = Subsample $(\hat{\mathbf{x}} * \hat{F})$

- Assume no collisions:
  - At most one large frequency hashes into each bucket.
  - Large frequency f_1 hashes to bucket b_1 :

$$\hat{\mathbf{y}}_{b_1} = \hat{\mathbf{x}}_{f_1} \times \hat{F}_{\Delta}$$

$$\hat{\mathbf{y}}^{\mathbf{1}}{}_{b_1} = \hat{\mathbf{x}}_{f_1} \times e^{-\mathrm{i}\,2\pi\,1 f_1/n} \times \hat{F}_{\Delta}$$

$$f_1 = -\frac{n}{2\pi}\angle\left(\frac{\hat{\mathbf{y}}_{b_1}}{\hat{\mathbf{y}}^{\mathbf{1}}{}_{b_1}}\right) \bmod n \qquad \hat{\mathbf{x}}_{f_1} = \frac{\hat{\mathbf{y}}_{b_1}}{\hat{F}_{\Delta}}$$

  - Find all frequencies in $2B\log(B)$

Random Hashing

- Some Large frequencies collide:
  - Subtract and recurs
  - Small number of collisions → converges in few iterations

- Every iteration needs new random hashing:
  - Permute time domain signal → permute frequency domain

  - σ is invertible mod n :

    $\mathbf{x'}_t = \mathbf{x}_{\sigma t} \times e^{-\mathrm{i}\,2\pi\, t\beta/n}$ $\hat{\mathbf{x}}'_f = \hat{\mathbf{x}}_{\sigma^{-1} f + \beta}$

  - Permutation : $f = \sigma^{-1} f + \beta \mod n$

Algorithm

- Iteration i :
  - $B_i \propto k / 2^{i-1}$
  - Permute spectrum : $\mathbf{x'}_{\boldsymbol{t}} = \mathbf{x}_{\sigma \boldsymbol{t}} \times e^{-\mathrm{i}\, 2\pi\, t\beta/n}$
  - $\hat{\mathbf{y}} = B_i$ -point DFT $(\mathbf{x}' \times F)$ = Subsample $\left(\hat{\mathbf{x}}' * \hat{F}\right)$
  - Recover locations and values of large frequencies

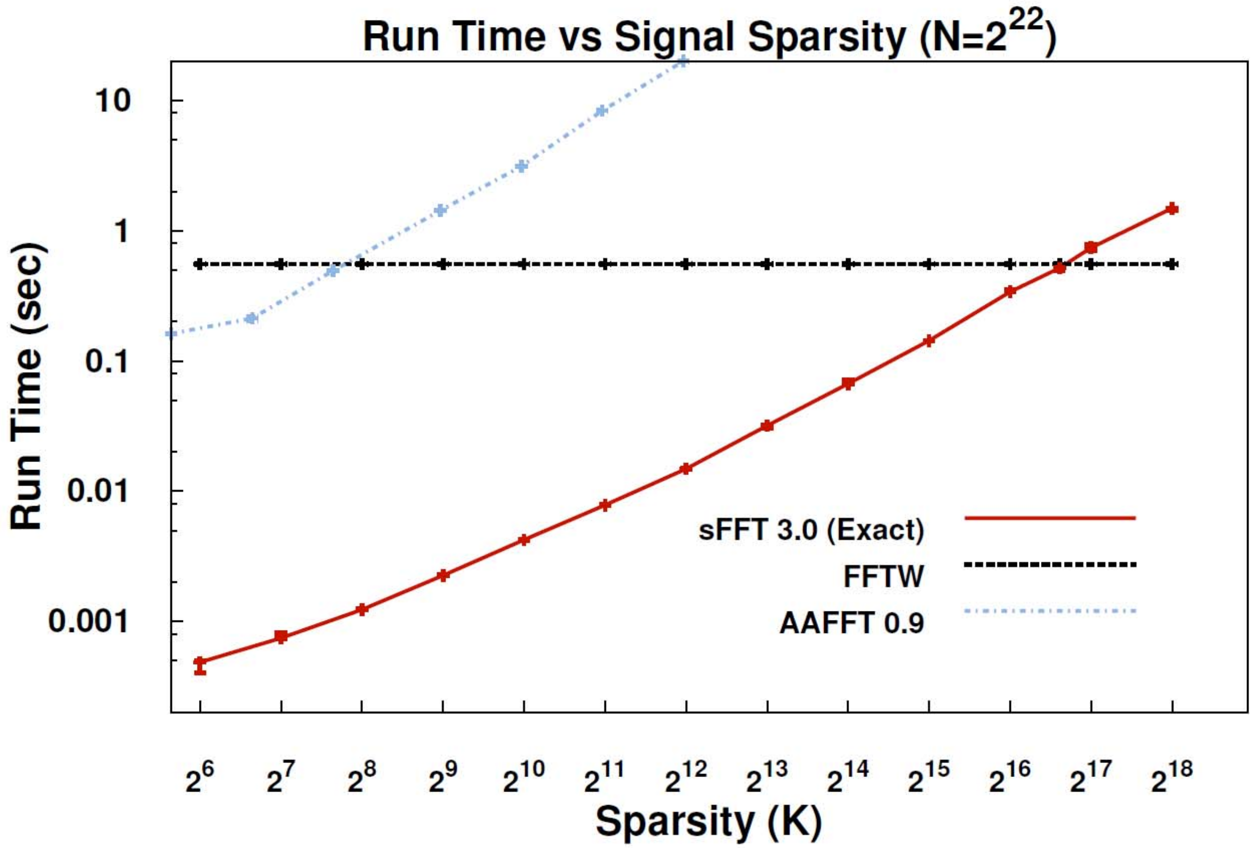
- Each iteration takes O($B_i \log B_i$) time.
- $O(\log k)$ iterations
- Total time : $O(k \log n)$

Experiments
(exactly k-sparse algorithm)

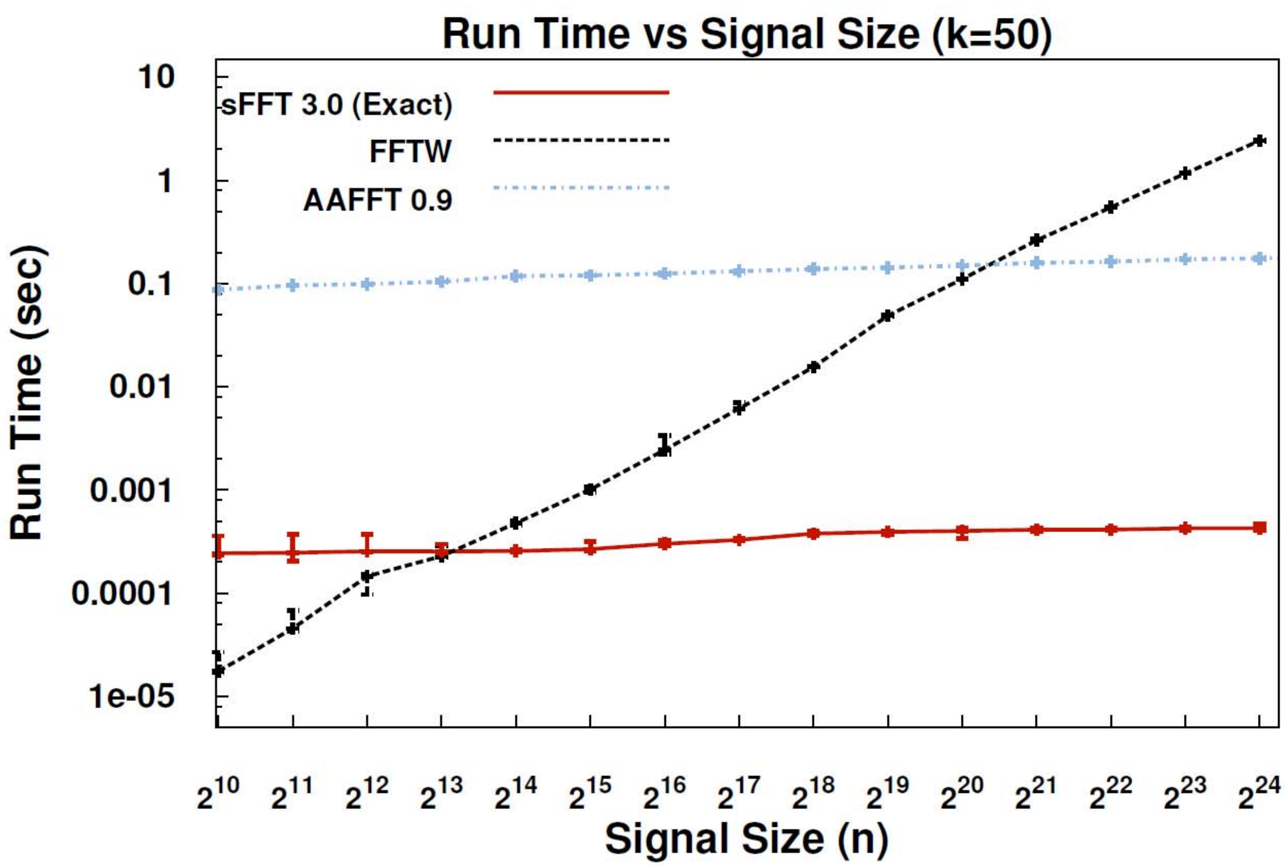
Setup

- Similar to earlier work:
  - Random 0-1 k-sparse vectors $\hat{\mathbf{x}}$
  - Fix n, vary k
  - Fix k, vary n

Experiments



Experiments, ctd



Conclusions

- Sparse FFT with running times :
  - $O(k \log n)$ for exactly sparse case
  - $O(k \log(n) \log(n/k))$ for approximately sparse case
  - Improves over FFT for $k << n$

- Significant improvement in practice

- $k \log n$ time for **approximately** sparse signals?
  - Not clear: $k \log(n/k)$ samples needed, extra $\log n$ for FT