Faster GPS via the Sparse Fourier Transform

Haitham Hassanieh
Fadel Adib
Dina Katabi
Piotr Indyk
GPS Is Widely Used

Faster GPS benefits many applications
How Do We Improve GPS?

Need to Improve GPS Synchronization
GPS Synchronization

Synchronization is locking onto a satellite’s signal
  • Consumes 30%-75% of GPS receiver’s power
    [ORG447X datasheet, Venus 6 datasheet]

GPS signals are very weak, less than -20dB SNR

100s of millions of multiplications
  [Team, Kaplan]
Goal

Faster Synchronization Algorithm
Reduce number of operations

Reduction in power consumption and delay
Rest of this Talk

- GPS Primer
- Our GPS Synchronization Algorithm
- Empirical Results
How Does GPS Work?

Compute the distance to the GPS satellites
How Does GPS Work?

Compute the distance to the GPS satellites

\[ \text{distance} = \text{propagation delay} \times \text{speed of light} \]
How to Compute the Propagation Delay?

Satellite Transmits CDMA code
How to Compute the Propagation Delay?

Code arrives shifted by propagation delay
How to Compute the Propagation Delay?

Receiver knows the code and when the satellite starts transmitting
How to Compute the Propagation Delay?

Correlation
How to Compute the Propagation Delay?

Correlation delay
How to Compute the Propagation Delay?

Correlation delay
How to Compute the Propagation Delay?

Correlation

Spike determines the delay
GPS Synchronization is a convolution with CDMA code

Convolutions in Time ⟷ Multiplication in Frequency

\[ O(n^2) \quad \text{and} \quad O(n \log n) \]

\( n \): Number of samples in the code
GPS Synchronization is a convolution with CDMA code

Convolution in Time \[ O(n^2) \]

Multiplication in Frequency \[ O(n \log n) \]

State of the art GPS synchronization algorithm: \[ O(n \log n) \]
Rest of this Talk

- GPS Primer
- Our GPS Synchronization Algorithm
- Empirical Results
QuickSync

• Fastest GPS synchronization algorithm to date

• Analytical complexity:
  – $O(n\sqrt{\log n})$ for any SNR
  – $O(n)$ for moderately low SNR

• Empirical Results:
  – Evaluated on real GPS signals
  – Improves performance by 2.2x
How can we make GPS synchronization faster than FFT-Based synchronization?
FFT-Based GPS Synchronization

Received Signal → FFT → Signal in Freq. → FFT of Code → IFFT → Output
FFT-Based GPS Synchronization

Received Signal → FFT → Signal in Freq. → FFT of Code → IFFT → Output

FFT Stage
IFFT Stage
FFT-Based GPS Synchronization

Received Signal → FFT → Signal in Freq. → FFT of Code → IFFT → Output

FFT Stage

IFFT Stage

Each stage takes $O(n \log n)$

→ need to reduce complexity of both stages
FFT-Based GPS Synchronization

Received Signal → FFT → Signal in Freq. → FFT of Code → IFFT → Output

FFT Stage

IFFT Stage
FFT-Based GPS Synchronization

Received Signal → FFT → Signal in Freq. → FFT of Code → IFFT → Sparse

FFT Stage → IFFT Stage → Output

correct shift
FFT-Based GPS Synchronization

Received Signal → FFT → Signal in Freq. → IFFT → Output

FFT Stage

IFFT Stage

Sparse IFFT
QuickSync
A Sparse IFFT algorithm customized for GPS

• Exactly One Spike ➔ Simpler algorithm

• Extends to the FFT-stage which is different (will discuss later)
QuickSync’s Sparse IFFT

1- Bucketize
Divide output into a few buckets

2- Estimate
Estimate the largest coefficient in the largest bucket

Original Output

value of bucket = \( \Sigma \) samples
QuickSync’s Sparse IFFT

1- Bucketize
Divide output into a few buckets

2- Estimate
Estimate the largest coefficient in the largest bucket

So how can we bucketize and estimate efficiently?
How to Bucketize Efficiently?

input samples

IFFT

output samples
How to Bucketize Efficiently?

input samples

IFFT

output samples

Subsamples

IFFT

Buckets
How to Bucketize Efficiently?

Efficient since: small IFFT of size equal to the number buckets
How to Estimate Efficiently?

• Keep largest bucket; ignore all the rest

• Out of the samples in the large bucket, which one is the spike?
How to Estimate Efficiently?

- Keep largest bucket; ignore all the rest
- Out of the samples in the large bucket, which one is the spike?

The spike is the sample that has the maximum correlation.
How to Estimate Efficiently?

- Keep largest bucket; ignore all the rest
- Out of the samples in the large bucket, which one is the spike?

The spike is the sample that has the maximum correlation

Efficient since: compute correlation only for few samples in the largest bucket
QuickSync’s Sparse IFFT

- $n$ is number of samples
- $k$ samples per bucket $\rightarrow n/k$ buckets

Bucketization: $n/k \log(n/k)$

Estimation: $k \times n$

$$k = \sqrt{\log n} \quad \Rightarrow \quad O(n\sqrt{\log n})$$
QuickSync Synchronization

Input → FFT → Signal in Freq. → FFT of Code → IFFT → Output

FFT Stage

Sparse IFFT
QuickSync Synchronization

Input → FFT → Signal in Freq. → FFT of Code → IFFT → Output

FFT Stage

Output is not sparse
Cannot Use Sparse FFT
QuickSync Synchronization

Input $\rightarrow$ FFT $\rightarrow$ Signal in Freq. $\rightarrow$ FFT of Code $\rightarrow$ IFFT $\rightarrow$ Output

FFT Stage

Sparse IFFT

Input to next stage
QuickSync Synchronization

Input → FFT → Signal in Freq. → FFT of Code → IFFT → Subsampled FFT → Sparse IFFT → Output

- Subsampled FFT: Need only few samples of FFT output
- Sparse IFFT: IFFT samples its input
QuickSync Synchronization

Input $\rightarrow$ FFT $\rightarrow$ Signal in Freq. $\rightarrow$ FFT of Code $\rightarrow$ IFFT $\rightarrow$ Output

Subsampled FFT  Sparse IFFT

FFT and IFFT are dual of each other
QuickSync Synchronization

Input → FFT → Signal in Freq. → FFT of Code → IFFT → Output

- Subsampled FFT
- Sparse IFFT

Bucketization → FFT → Subsampling → IFFT → Bucketization
QuickSync Synchronization

Input → FFT → Signal in Freq. → FFT of Code → IFFT → Output

Subsampled FFT: $O(n\sqrt{\log n})$

Sparse IFFT: $O(n\sqrt{\log n})$
QuickSync Synchronization

Input → Subsampled FFT → Sparse IFFT → Correct delay

Subsampled FFT of Code
Theorem: (informally restated)
For any SNR QuickSync achieves the same accuracy as FFT-Based synchronization and has a complexity of $O(n\sqrt{\log n})$ where $n$ is the number of samples in the code.

For moderately low SNR (i.e. noise is bounded by $O(n/\log^2 n)$), QuickSync has $O(n)$ complexity.
Rest of this Talk

- GPS Primer
- Our GPS Synchronization Algorithm
- Empirical Results
• Traces are collected both US and Europe
• Different locations: urban – suburban
• Different weather conditions: cloudy – clear
Compared Schemes

• QuickSync Synchronization
• FFT-Based Synchronization
Metrics

- Hardware implementations

\[
\text{Multiplication Gain} = \frac{\text{Multiplications of baseline}}{\text{Multiplications of QuickSync}}
\]

- Software implementations

\[
\text{FLOPS Gain} = \frac{\text{FLOPS of baseline}}{\text{FLOPS of QuickSync}}
\]
Multiplication Gain

Graph showing CDF vs. GAIN with US Traces. Key points:
- 1.3x at a certain CDF value
- 2.1x at a higher CDF value
- 3x at the highest CDF value.
QuickSync provides an average gain of 2.1x
QuickSync provides an average gain of 2.2 ×
Does the Gain Depend on the GPS SNR?

QuickSync improves over FFT-Based for the whole range of GPS SNRs
Related Work

• Past work on GPS [NC91, SA08, RZL11]
  – QuickSync presents the fastest algorithm to date

• Sparse FFT Algorithms [GMS05, HKIP12a, HKIP12b]
  – QuickSync’s bucketization leverages duality
    → reduces the complexity of both stages in GPS
Conclusion

• Fastest GPS synchronization algorithm
  – $O(n\sqrt{\log n})$ for any SNR
  – $O(n)$ for moderately low SNR

• Empirical results show an average 2x gain

• QuickSync applies to general synchronization tasks beyond GPS