Implementing a Speech Recognition System on a GPU using CUDA

Presented by

Omid Talakoub

Astrid Yi

Outline

- Background
- Motivation
- Speech recognition algorithm
- Implementation steps
- GPU implementation strategies
- Data flow and representation
- Profiler results
- Floating point accuracy
- Future optimizations

Background

Speech recognition system:

1. Speaker-dependent or speaker-independent
2. Isolated words or continuous speech

Practical applications use isolated-word recognition systems

Motivation

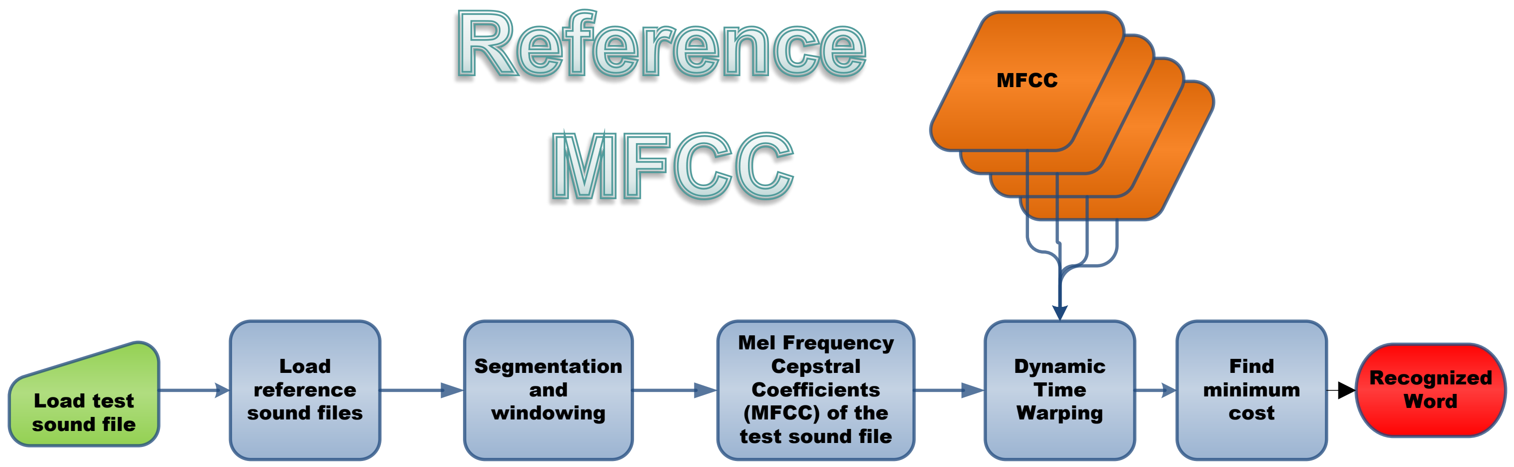
2 phases in speech recognition:

1. Memorize a set of reference templates
2. Take a test template and return the closest reference template match

Recognition accuracy improved with a larger set of reference templates

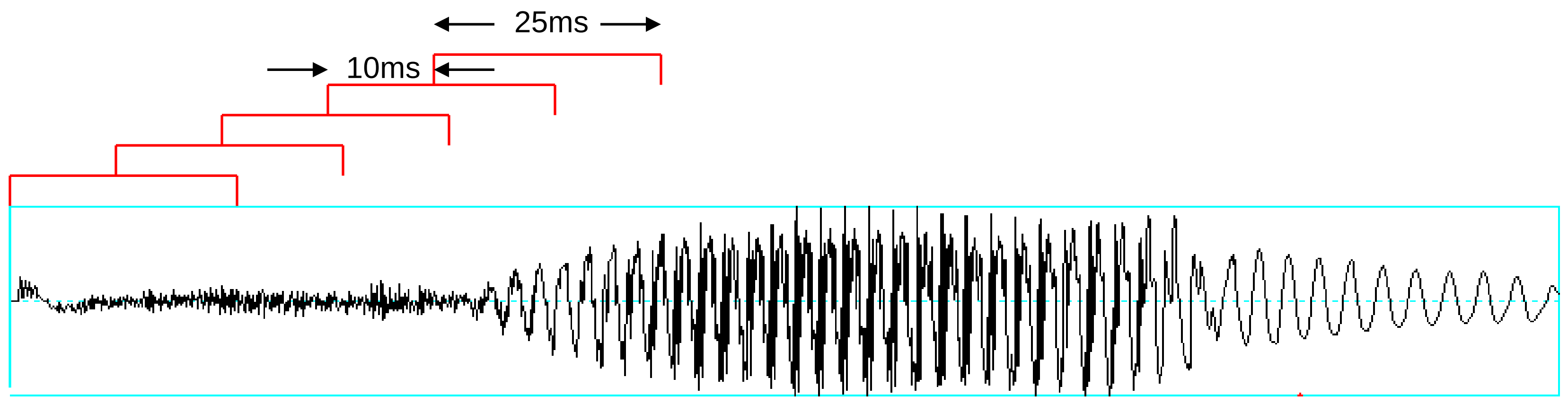
But, it takes more time to find the closest match

Algorithm Overview


Reference
MFCC
MFCC
Load test sound file
Load reference sound files
Segmentation and windowing
Mel Frequency Cepstral Coefficients (MFCC) of the test sound file
Dynamic Time Warping
Find minimum cost
Recognized Word

Hamming Window

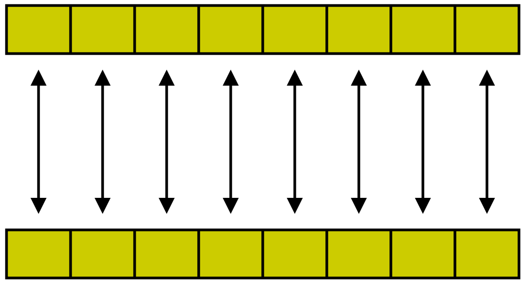
- Words are parameterised on a frame-by-frame basis
- Choose frame length, over which speech remains reasonably stationary
- Overlap frames e.g. 25ms frames, 10ms frame shift



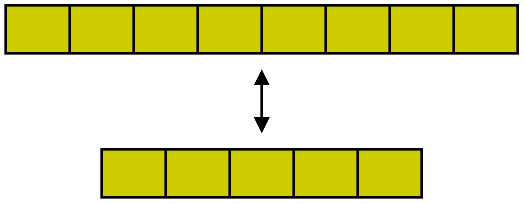
- We want to compare frames of test and reference words i.e. calculate distances between them

Calculating Distances

- Easy:
  - Sum differences between corresponding frames

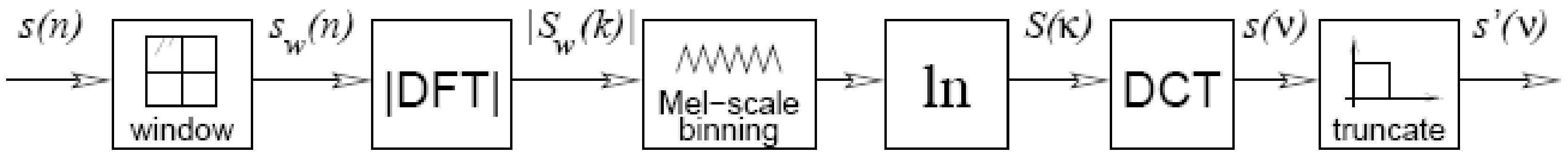


- Problem:
  - Number of frames won’t always correspond

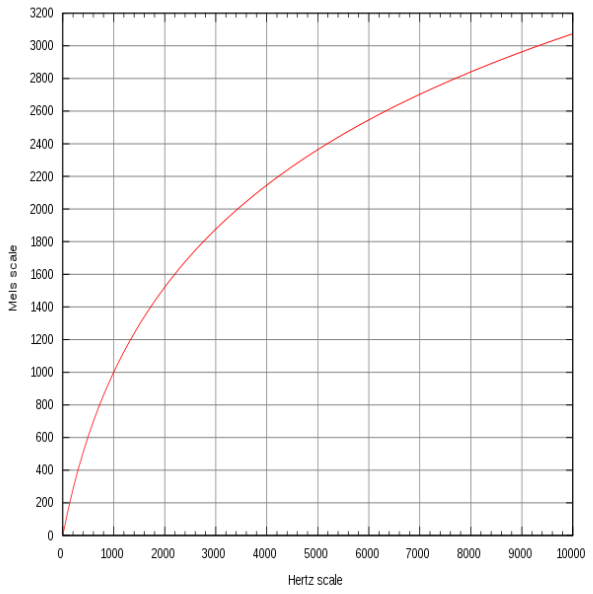


Feature Extraction

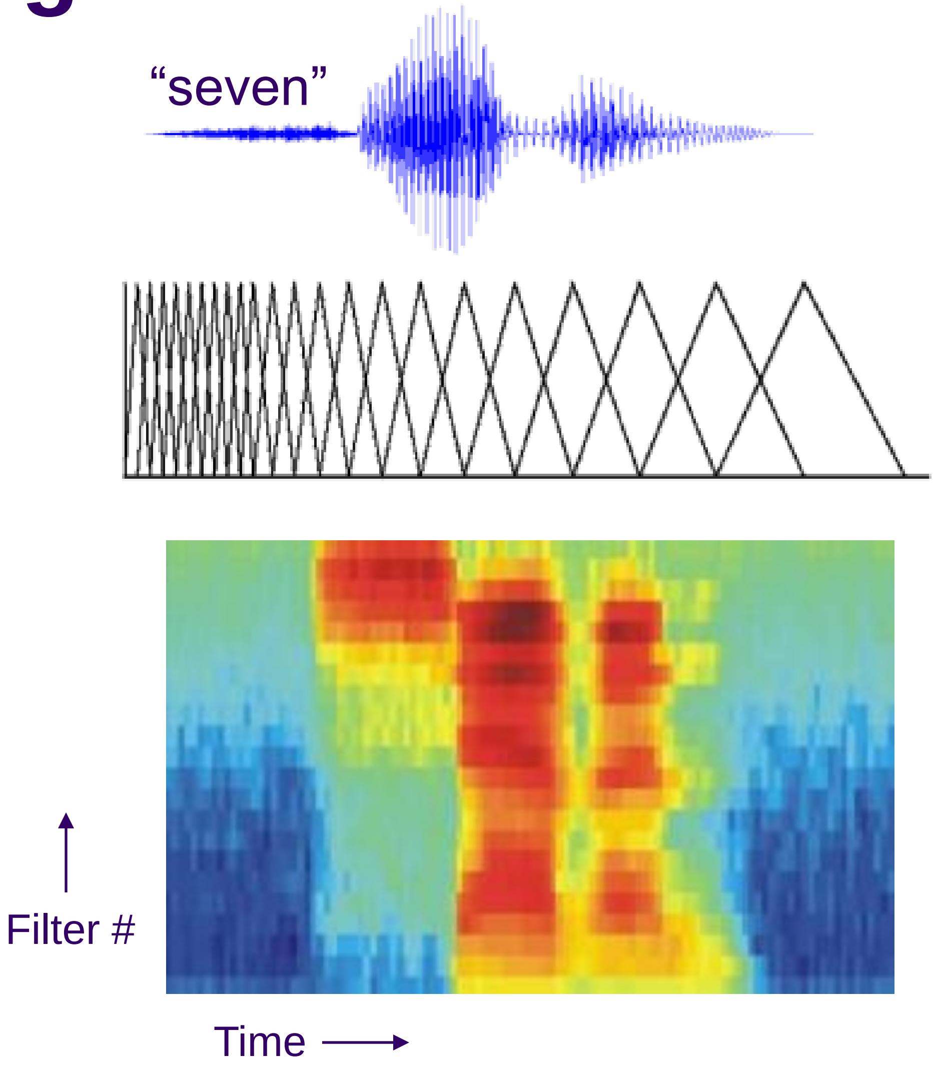
Calculating Mel-frequency cepstral coefficients (MFCCs):



MFCCs are coefficients of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear *mel scale* of frequency.

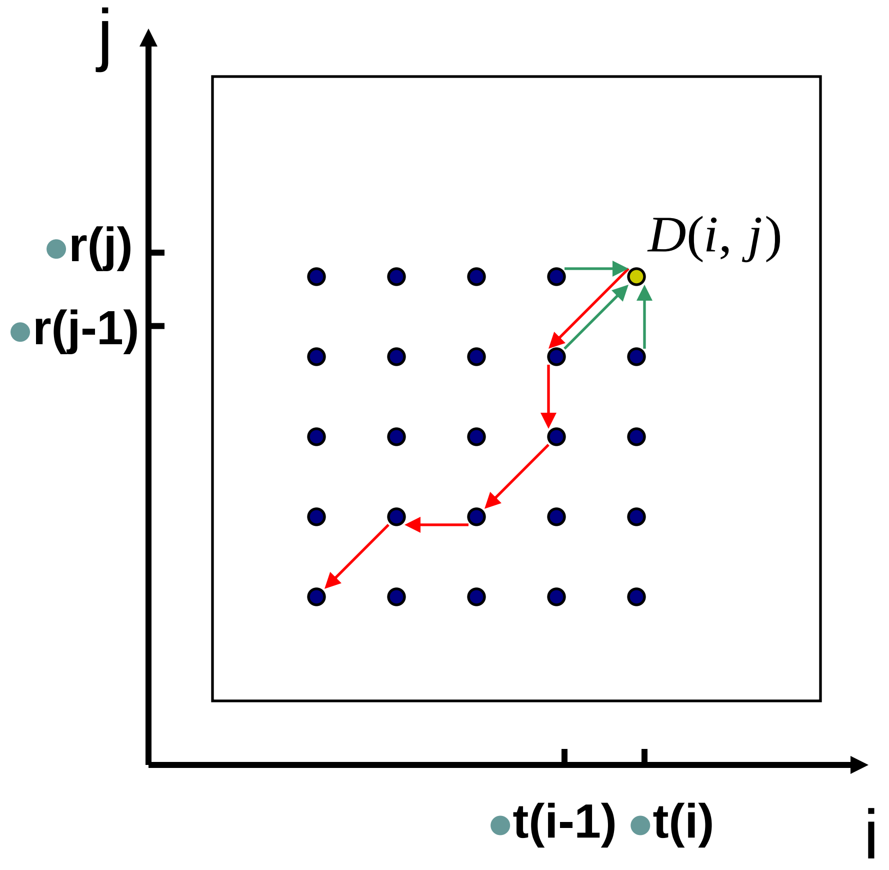


MFCC algorithm



$$\begin{tikzcd}
x(t) \arrow[d, "\text{Fourier}"] \\
\text{Mel-scaled filter bank} \arrow[d] \\
\text{Log energy} \arrow[d, "\text{DCT}"] \\
\text{Cepstral domain}
\end{tikzcd}$$

Dynamic Time Warping



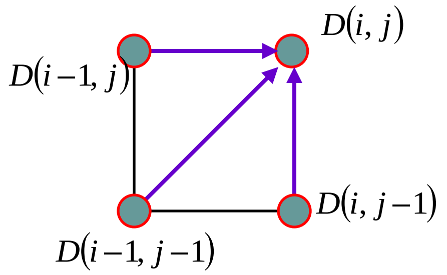
- t: input MFCC matrix
- (Each row is a frame's feature.)
- r: reference MFCC matrix
- Local paths: 0-45-90 degrees

- DTW recurrence:

$$D(i,j) = \left\| t(i), r(j) \right\| + \min \begin{Bmatrix} D(i, j-1) \\ D(i-1, j-1) \\ D(i-1, j) \end{Bmatrix}$$

Local Path Constraints

0-45-90 local paths

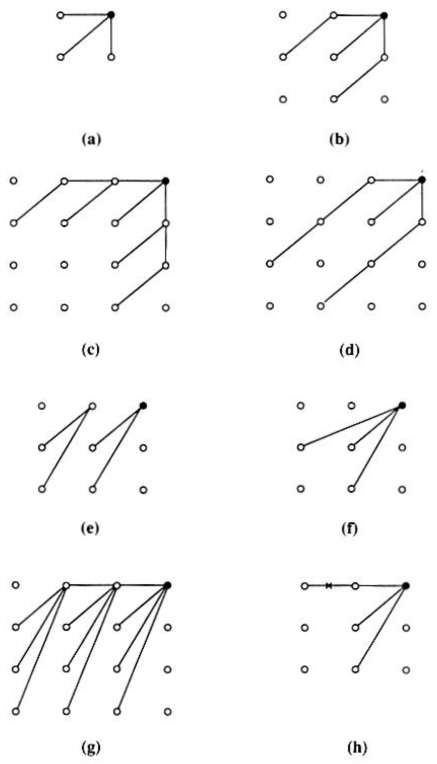


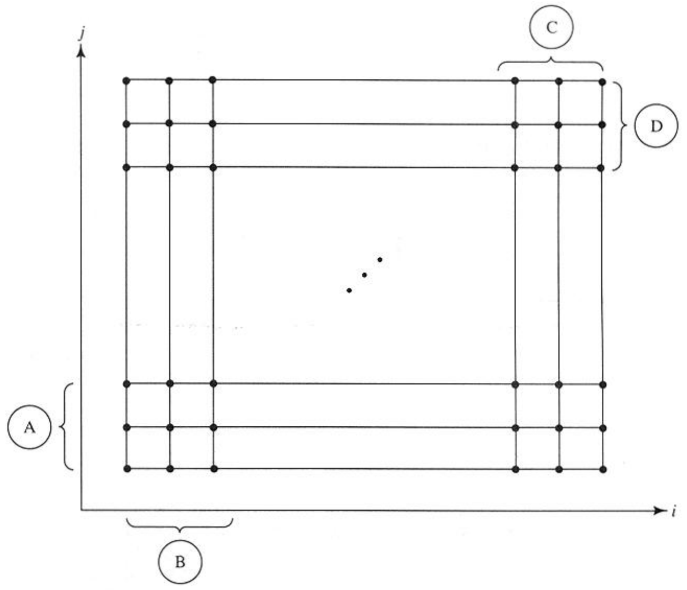
$$D(i, j) = \|t(i) - r(j)\| + \min \begin{Bmatrix} D(i, j-1) \\ D(i-1, j-1) \\ D(i-1, j) \end{Bmatrix}$$

Other Variants

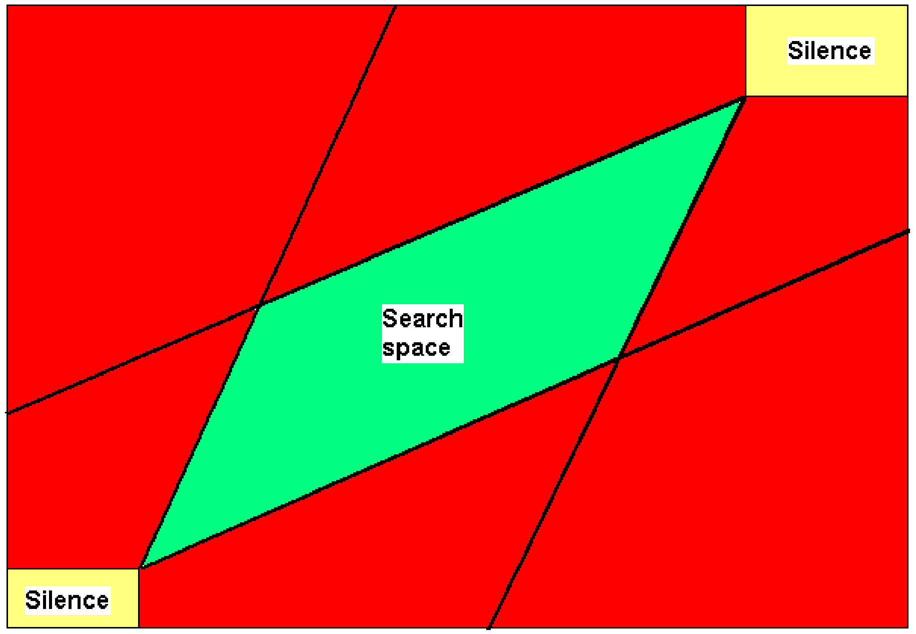
Local constraints

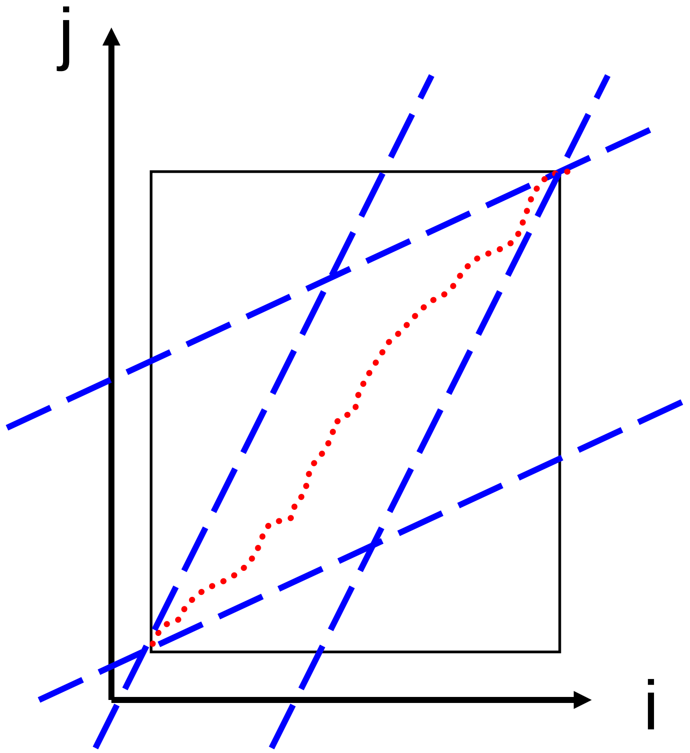
Start/ending area



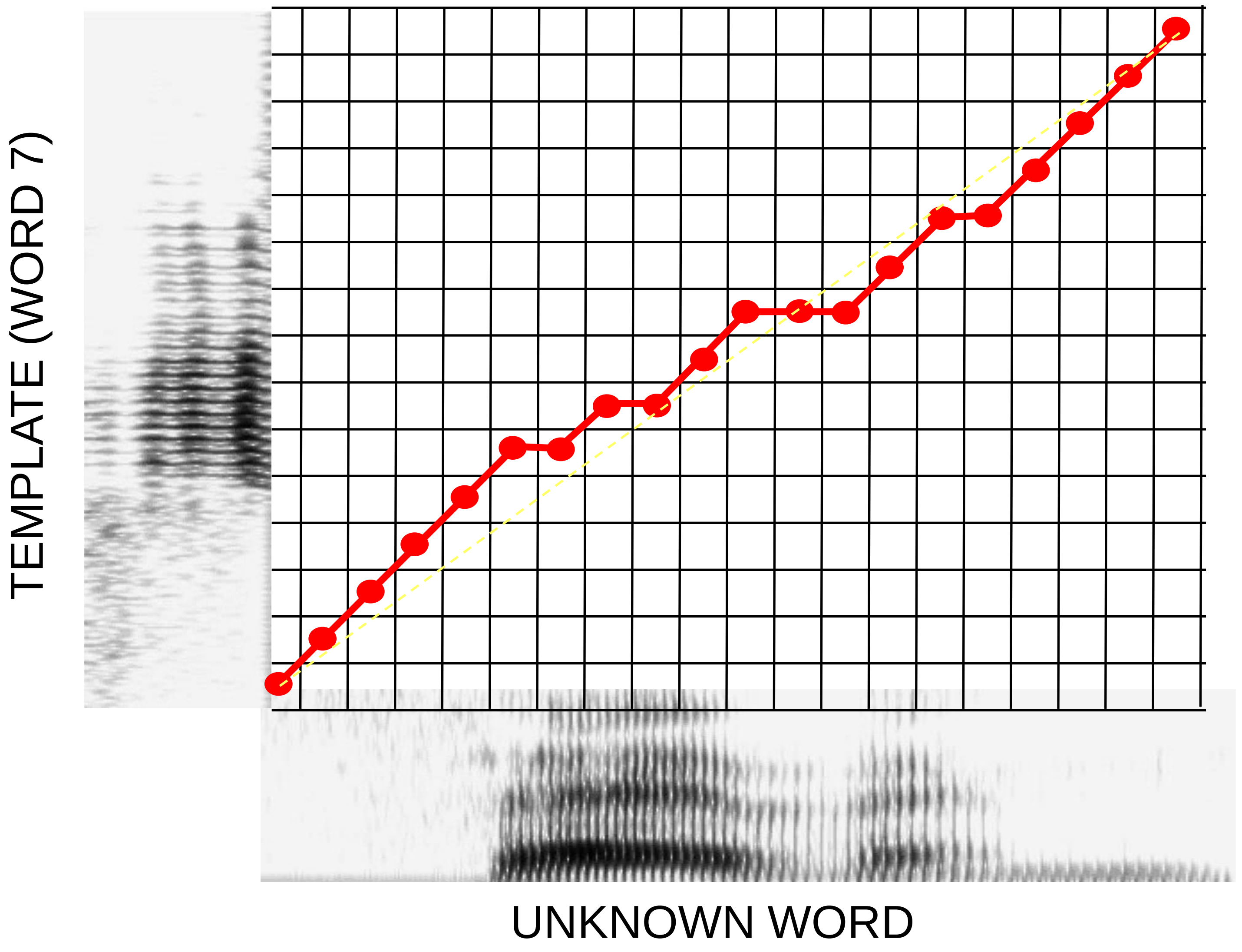


DTW Paths of “Match Corners”

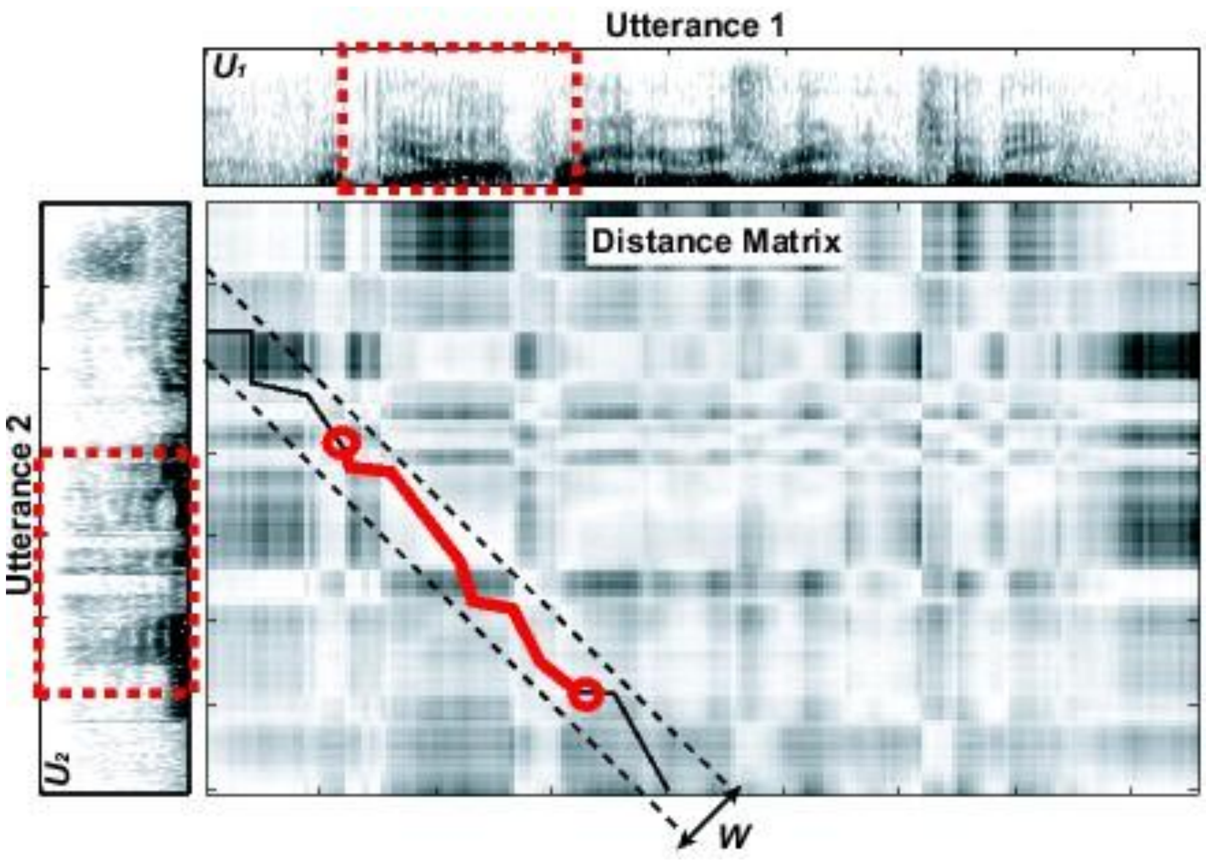




Dynamic Time Warping
The Brute Force of the Engineering Approach



Another Example of DTW Minimum Path



Reference Implementation

- Reference code implemented on CPU
- Provided a base on which to compare results from GPU
- Based on original MATLAB code
- Used as a base to compare performance increase by shifting work to the GPU
- Single threaded with no SIMD type execution used

GPU Implementation Strategies

Two main approaches to implementation

1. Do more work in the same amount of time
2. To the same amount of work in less time

How to choose an approach in general

1. Amdahl’s law states that performance is dictated by how much can be parallelized
2. If algorithm is serial, do more work simultaneously instead of faster

Data Flow and Representation

- Matrices stored in row major format with a pitch equal to width
- After initial data load, all operations on data take place on device and transform the data in place or to another location
- Transformation implemented in the form of CUDA kernels
- Kernels can be invoked to process all the data at a specific transformation stage

Profiler Results

CPU implementation was used as a base

Profiled the generation of the MFCCs which is the computationally expensive portion

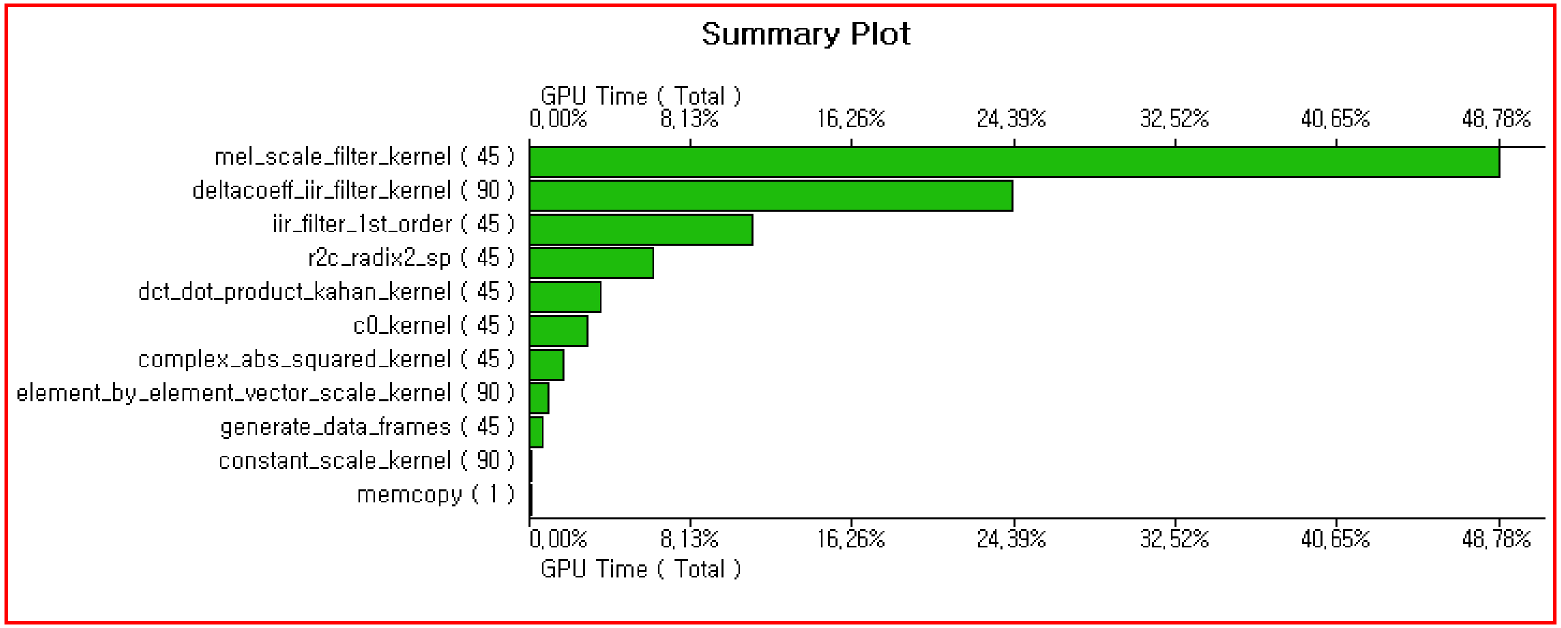
Averaging the results over 5 runs of 45 MFCC calls yields the following results:

1. GPU Time: 1.4080285 seconds
2. CPU Time: 8.6354016 seconds
3. Speedup: 5.8

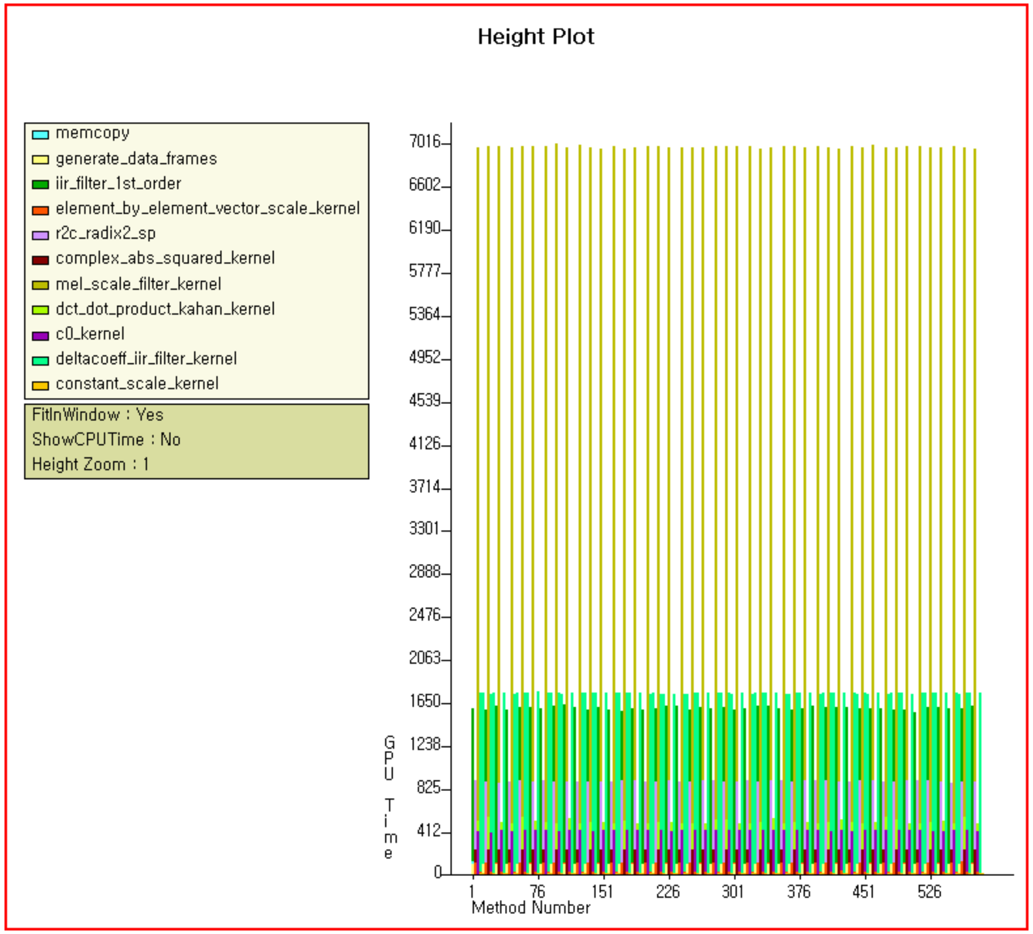
Kernel GPU Utilization

Half of GPU runtime is limited to a single kernel

Further optimizations should concentrate on first three listed kernels



Kernel Runtimes



Floating Point Accuracy

Floating point accuracy was an issue during verification

Two main problem areas:

1. FFT
2. DCT

Problems from the FFT could not be fixed due to the existing implementation in CUFFT

Problems in the DCT occurred due to cancellation

Floating Point Accuracy con’t

- Determined that differences between implementations did not affect results
- To continue comparing against reference results, CPU data was loaded for the GPU during verification

Future Optimizations

- Use the already optimized CUBLAS library for some operations
- Requires reordering of data to use column major ordering
- IIR filters used have coefficients such that their invocations can be better parallelized
- Allocate memory for matrices based using pitches to align data, ie. cudaMalloc2D