In Silico Vox: Towards Speech Recognition in Silicon

Edward C. Lin, Kai Yu, Rob A. Rutenbar, Tsuhan Chen
Electrical & Computer Engineering
{eclin, kaiy, rutenbar, tsuhan}@ece.cmu.edu
Speech Recognition Today

- Quality = *OK*  Vocab = *large*

- Quality = *poor*  Vocab = *small*

- Commonality: all *software* apps
Today’s Best Software Speech Recognizers

- Best-quality recognition is computationally **hard**
  - For speaker-independent, large-vocabulary, continuous speech

- **1-10-100-1000** rule
  - For ~1X real-time recognition rate
  - For ~10% word error rate (90% accuracy)
  - Need ~100 MB memory footprint
  - Need ~100 W power
  - Need ~1000 MHz CPU

- This proves to be very **limiting** …
The Carnegie Mellon *In Silico Vox* Project

- The thesis: It’s time to liberate speech recognition from the unreasonable limitations of software

- The solution: *Speech recognition in silicon*

- Why…?
  - Tomorrow’s compelling apps need 100X – 1000X performance improvements to accomplish. (Not going to happen in software)
  - We have some successful *historical* examples of this migration
Nobody paints pixels in software anymore!

- Too limiting in max performance. Too inefficient in power.

True on the desktop (& laptop) ...and on your cellphone too

http://www.nvidia.com

http://www.mtekvision.com
Next-Gen Compelling Applications

Audio-mining
- Very fast recognizers – much faster than realtime
- App: search large media streams (DVD) quickly

Hands-free appliances
- Very portable recognizers – high quality result on << 1 watt
- App: interfaces to small devices, cellphone dictation

FIND: “Hasta la vista, baby!”

“send email to arnold – let’s do lunch…”
Our Focus: How to Get to Fast...

Audio-mining
- Very fast recognizers – much faster than realtime
- App: search large media streams (DVD) quickly

Find: “Hasta la vista, baby!”

Hands-free appliances
- Very portable recognizers – high quality result on << 1 watt
- App: interfaces to small devices, cellphone dictation

“send email to arnold – let’s do lunch...”
About This Talk

- The $2 tour: How speech recognition works
  - What happens in a recognizer

- An ASIC architecture
  - Stripping away all CPU stuff we don’t need, focus on essentials

- Results
  - ASIC version: Cycle simulator results
  - FPGA version: Live, running hardware-based recognizer
How Speech Recognition Works

Acoustic Frontend

Sampling

Feature extraction

ADC

Filter1
Filter2
Filter3
FilterN

DSP

Feature vector

x1 x2 x3 . . . xn

Feature Scoring

“ao”

HMM / Viterbi Search
Language Model Search

Acoustic units → Words → Language

Adaptation to environment/speaker

Adaptation

Acoustic

Word

Language

Æ

Æ

Æ

Grammar

Adaptation to environment/speaker

Language Adaptation

rob

ROB

BOB

…

Rob AO B

Bob B AO B

…

Rob → says

Scoring

Backend Search

robe
The frontend is all **DSP**. A discrete Fourier transform (DFT) gives us the spectra. We combine and logarithmically transform spectra in ways motivated by physiology of human ears.

Combine these with estimates of 1\textsuperscript{st} and 2\textsuperscript{nd} time derivatives.

Color is "how much energy" in transformed spectra. Green = low, red = high.

This pic is across a few sec of speech.
(2) Scoring Stage

- Each feature is a point in high-dimensional space
  - Each “atomic sound” is a region of this space
  - Score each atomic sound with Probability(sound matches feature)

Each sound approximated as a set of high-dim Gaussian densities

Note: (sounds) X (dimensions) X (Gaussians) = BIG
(3) Search: Speech Models are *Layered* Models

Language X Words X Acoustic $\rightarrow$ Layered Search

words

“acoustic units”

“sub-acoustic units”

Classical methods (HMMs, Viterbi) and idiosyncrasies

1 frame of sampled sound
English has ~50 atomic sounds (phones) but we recognize ~50x50x50 context-dependent triphones

Because “AY” sound in “five” is different than the “AY” in “nine”

“AY” in “five” != “AY” in “nine”
Similar Idea at Top: \textit{N-gram} Language Model

Suppose we have vocabulary \{W1, W2, W3, W4, \ldots\}

Lets us calculate likelihood of word \textit{W3} after \textit{W2} after \textit{W1}
Good Speech Models are BIG

- This is ~64K word “Broadcast News” task
- Unfortunately, many idiosyncratic details in how layers of model traversed
Where Does Software Spend its Time?

- CPU time for CMU Sphinx 3.0
  - Prior studies targeted less capable versions (v1, v2)
  - SimpleScalar & Intel Vtune
  - 64K-word “Broadcast News” benchmark

- So: It’s all **backend**

<table>
<thead>
<tr>
<th>Software Activity</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling</td>
<td>27%</td>
</tr>
<tr>
<td>Feature extraction</td>
<td>33%</td>
</tr>
<tr>
<td>Feature vector</td>
<td>19%</td>
</tr>
<tr>
<td>Feature scoring</td>
<td>15%</td>
</tr>
<tr>
<td>Acoustic units</td>
<td>6%</td>
</tr>
<tr>
<td>Word</td>
<td>19%</td>
</tr>
<tr>
<td>Language</td>
<td>6%</td>
</tr>
<tr>
<td>HMM / Viterbi Search</td>
<td>19%</td>
</tr>
<tr>
<td>Language Model Search</td>
<td>15%</td>
</tr>
</tbody>
</table>

~0% of time!

CPU time for CMU Sphinx 3.0

Prior studies targeted less capable versions (v1, v2)
SimpleScalar & Intel Vtune
64K-word “Broadcast News” benchmark

So: It’s all **backend**
Memory Usage? SPHINX 3.0 vs Spec CPU2000

- **Cache sizes**
  - L1: 64 KB, direct mapped
  - DL1: 64 KB, direct mapped
  - UL2: 512 KB, 4-way set assoc

- **So…**
  - **Terrible locality** (no surprise, graph search + huge datasets)
  - **Load dominated** (no surprise, reads a lot, computes a little)
  - Not an insignificant footprint

### Memory Footprint
- SPHINX 3.0
- Gcc: 64 MB
- Gzip: 24 MB
- Equake: 186 MB
- Total: 42 MB

### Cache Miss Rates
- DL1: 0.04
- L2: 0.48

### Instruction Mixes
- Loads: SPHINX 3.0: 0.27, Gcc: 0.25, Gzip: 0.2, Equake: 0.27
- Stores: SPHINX 3.0: 0.05, Gcc: 0.15, Gzip: 0.09, Equake: 0.08
- Branch’s: SPHINX 3.0: 0.14, Gcc: 0.2, Gzip: 0.17, Equake: 0.12

### Branch Misprediction Rates
- SPHINX 3.0: 0.025, Gcc: 0.07, Gzip: 0.08, Equake: 0.02

### Cycles
- SPHINX 3.0: 53 T
- Gcc: 55B
- Gzip: 15 B
- Equake: 23 B

### IPC
- SPHINX 3.0: 0.69
- Gcc: 0.29
- Gzip: 1.05
- Equake: 0.7

© Rob A. Rutenbar 2006
A Silicon Architecture: Big Picture

- **Acoustic Frontend**
  - SRAM (constants)
  - KB

- **Gaussian Scoring**
  - SRAM (communic)
  - KB

- **Backend Search**
  - MBs SRAM (active recog)
  - 10-100MB DRAM (models & active recog)

### Computations (Ops)
- Low

### SRAM (size)
- Small

### DRAM (size)
- Medium/Large
- Large

### DRAM (bandwidth)
- High
- High
Essential Implementation Ideas

- **Custom precision, everywhere**
  - Every bit counts, no extras, no floating point – all fixed point

- **(Almost) no caching**
  - Like graphics chips: fetch from SDRAM, do careful data placement
  - (Little bit of caching for bandwidth filtering on big language models)

- **Aggressive pipelining**
  - If we can possibly overlap computations – we try to do so

- **Algorithm transformation**
  - Some software computations are just bad news for hardware
  - Substitute some “deep computation” with hardware-friendly versions
Example: Aggressive Pipelining

Pipelined Get-HMM/Viterbi and Transition stages

<table>
<thead>
<tr>
<th>Fetch Word</th>
<th>Fetch HMM/Viterbi</th>
<th>Transition/Prune/Writeback</th>
<th>Fetch HMM/Viterbi</th>
<th>Transition/Prune/Writeback</th>
<th>Language Model</th>
</tr>
</thead>
</table>

Pipelined Get-Word and Get-HMM stages

<table>
<thead>
<tr>
<th>Fetch Word</th>
<th>Fetch HMM/Viterbi</th>
<th>Transition/Prune/Writeback</th>
<th>Fetch HMM/Viterbi</th>
<th>Transition/Prune/Writeback</th>
<th>Language Model</th>
</tr>
</thead>
</table>

Pipelined non-LanguageModel and LanguageModel stages

<table>
<thead>
<tr>
<th>Fetch Word</th>
<th>Fetch HMM/Viterbi</th>
<th>Transition/Prune/Writeback</th>
<th>Language Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch Word</td>
<td>Fetch HMM/Viterbi</td>
<td>Transition/Prune/Writeback</td>
<td>Fetch HMM/Viterbi</td>
</tr>
</tbody>
</table>
Example: Algorithmic Changes

- **Acoustic-level pruning threshold**
  - **Software**: Use best score of *current* frame (after Viterbi on Active HMMs)
  - **Silicon**: Use best score of *previous* frame (nixes big temporal bottleneck)

- **Tradeoffs**
  - Less memory bandwidth, can pipeline, little pessimistic on scores

```
Sphinx 3.0

Start Frame

Initialize Frame

Done

Fetch Active Word

Fetch Active HMM

Viterbi

Writeback HMM

Done all Active HMM

Fetch Active HMM

Viterbi

Writeback HMM

Done all Active HMM

Language Model

Transition/Prune/Writeback

Silicon

Start Frame

Initialize Frame

Done

Fetch Active Word

Fetch Active HMM

Viterbi

Writeback HMM

Done all Active HMM

Language Model

Transition/Prune/Writeback
```
Predicted Performance: C++ Cycle Simulator

- **Benchmark:** 5K-word “Wall Street Journal” task
- **Results:**
  - No accuracy loss; not quite 2X @ 125MHz ASIC clock
  - Backend search needs: ~1.5MB SRAM, ~30MB DRAM

<table>
<thead>
<tr>
<th>Recognizer Engine</th>
<th>Word Error Rate (%)</th>
<th>Clock (GHz)</th>
<th>Speedup Over Real Time (bigger is better)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software: Sphinx 3.3 (fast decoder)</td>
<td>7.32%</td>
<td>1 GHz</td>
<td>0.74X</td>
</tr>
<tr>
<td>Software: Sphinx 4 (single CPU)</td>
<td>6.97%</td>
<td>1 GHz</td>
<td>0.82X</td>
</tr>
<tr>
<td>Software: Sphinx 4 (dual CPU)</td>
<td>6.97%</td>
<td>1 GHz</td>
<td>1.05X</td>
</tr>
<tr>
<td>Software: Sphinx 3.0 (single CPU)</td>
<td>6.707%</td>
<td>2.8 GHz</td>
<td>0.59X</td>
</tr>
<tr>
<td><strong>Hardware: Our Proposed Recognizer</strong></td>
<td><strong>6.725%</strong></td>
<td><strong>0.125 GHz</strong></td>
<td><strong>1.67X</strong></td>
</tr>
</tbody>
</table>
Aside: Bit-Level Verification Hurts (A Lot)

- We have newfound sympathy for others doing silicon designs that handle large media streams
  - Generating these sort of tradeoff curves: CPU days $\rightarrow$ weeks

![Graph showing speedup vs software time](image-url)
In any “system design” research, you reach a point where you just want to see it work – *for real*

- **Goal:** *Full recognizer 1 FPGA + 1 DRAM*

- A benchmark that fits on chip
  - 1000-word “Resource Mgt” task
  - Slightly simplified: no tri-grams
  - Slower: not real time, ~2.3X slower
  - Resource limited: slices, mem bandwidth
Aside: as far as we know, this is the *most complex* recognizer architecture ever fully mapped into a hardware-only form.
Summary

- **Software is too constraining for speech recognition**
  - Evolution of graphics chips suggests alternative: **Do it in silicon**
  - Compelling performance and power reasons for silicon speech recog

- **Several “in silico vox” architectures in design**
  - ASIC version: ~1.6X realtime for 5K-word task; 10X version in progress
  - FPGA version: tiny design successfully running 1000-word benchmark

- **Directions**
  - Exploit Berkeley BEE2 emulation engine: ~25X more FPGA resources
  - Detailed architecture/performance/power tradeoffs for mobile apps
Acknowledgements

- **Work supported by**
  - National Science Foundation
  - Semiconductor Research Corporation
  - MARCO/DARPA Center for Circuit & System Solutions (C2S2)

- **We are grateful for the advice and speech recognition expertise shared with us by**
  - Richard M. Stern, CMU
  - Arthur Chan, CMU