Towards Speech Recognition in Silicon: The Carnegie Mellon In Silico Vox Project

Rob A. Rutenbar
Professor, Electrical & Computer Engineering
rutenbar@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*
Aside: About the Name “In Silico Vox”

- **In Vivo**
  - Latin: an experiment done in a living organism

- **In Vitro**
  - Latin: an experiment done in an artificial lab environment

- **In Silico**
  - (Not real Latin): an experiment done via computation only

- **Vox**
  - Latin: voice, or word

About This Talk

- **Some philosophy**
  - Why silicon? Why now? Why us (CMU)?

- **A quick tour: How speech recognition works**
  - What happens in a recognizer

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

- **Results**
  - ASIC version: Simulation results
  - FPGA version: Live, running hardware-based recognizer
About This Talk

- **Some philosophy**
  - Why silicon? Why now? Why us (CMU)?

- **A quick tour: How speech recognition works**
  - What happens in a recognizer

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

- **Results**
  - ASIC version: Simulation results
  - FPGA version: Live, running hardware-based recognizer

Why Silicon? Why Now?

**Why? Two reasons:**

- **History**
  - We have some successful historical examples of this migration

- **Performance**
  - Tomorrow’s compelling apps need 100X – 1000X more performance
  - (Not going to happen in software)
History: Graphics Engines

- 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

Performance: Next-Gen Compelling Applications

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...”
Silicon Solution: Speed and Power Wins

- A famous graph from Prof. Bob Brodersen of Berkeley
  - Study looked at 20 designs published at ISSCC, from 1997-2002
  - In slightly older technologies, relative to today: 180nm – 250nm
  - Dedicated designs up to 10,000X better energy efficiency (MOPS/mW)

![Graph showing energy efficiency comparison between different processor types.](image)

Recent Example: Parallel Radio Baseband DSP

- 90nm CMOS: adaptive DSP for multipath MIMO channel
  - Power efficiency = 2.1GOPS/mW
  - Area efficiency = 20GOPS/mm²

![Diagram showing parallel radio baseband DSP architecture.](image)

(Source: Prof. Dejan Markovitz, UCLA)
Why Us...?

- 1 site (Carnegie Mellon), 3 sets of world-class experts
  - Impossible to do projects like this without cross-area linkages

- Computer Science
  - SPHINX Speech recognition group

- Electrical & Computer Engineering
  - Silicon system implementation group

- Electrical & Computer Engineering
  - Media / DSP group

Us: the CMU In Silico Vox Team

From left: Kai Yu, Rob Rutenbar, Edward Lin, Richard Stern, Tsuhan Chen, Patrick Bourke (not shown: Jeff Johnston)
About This Talk

- Some philosophy
  - Why silicon? Why now? Why us (CMU)?

- A quick tour: How speech recognition works
  - What happens in a recognizer

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

- Results
  - ASIC version: Simulation results
  - FPGA version: Live, running hardware-based recognizer

How Speech Recognition Works

Acoustic Frontend  Scoring  Backend Search

Adaptation to environment/speaker  Feature extraction  Feature Scoring  HMM / Viterbi Search

Sampling  Word  Language Model Search

Acoustic units  Words  Language

Rob
(1) Acoustic Frontend

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 1st and 2nd 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
  - But 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 → Layered Search

**YES**

/Y/ [EH/][/S/]

words

“acoustic units”

**NO**

/[I]/ [IOW/]

“sub-acoustic units”

---

Context Matters: At Bottom -- *Triphones*

- English has ~50 atomic sounds (*phones*) but we recognize ~50x50x50 context-dependent *triphones*
  - Because ‘I’ sound in “five” is different than the ‘I’ in “nine”

Five

\[ F(\cdot, I) \_{\text{cross-word}} \]

\[ F(\cdot, V) \_{\text{word-internal}} \]

\[ V(I, \cdot) \_{\text{cross-word}} \]

Nine

\[ N(\cdot, I) \_{\text{cross-word}} \]

\[ N(I, N) \_{\text{word-internal}} \]

\[ N(I, \cdot) \_{\text{cross-word}} \]

“I” in “five” ≠ “I” in “nine”
Similar for Other Languages, like Japanese

- ...but different basic building blocks (different phones)

Source: Microsoft Speech API 5.3 Japanese Phonemes

Example of Different Phone Building Blocks

- Let’s use my name as an example

- English: /r/ /OO/ /l/ /n/ /b/ /ar/

- Japanese: /ル/ /ーラ/ /テラ/ /ン/ /バ/ /ーラ/

- Aside:
  - Japanese has some reputation as being an “easier” language for automatic recognition
  - Mapping from basic sounds (mora) to words is simpler than English
Suppose we have vocabulary 
\{ W1, W2, W3, W4, \ldots \}

Let's us calculate likelihood of word \textit{W3} after \textit{W2} after \textit{W1}

- This is \sim 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)
  - Tools: 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**

<table>
<thead>
<tr>
<th></th>
<th>SPHINX 3.0</th>
<th>Gcc</th>
<th>Gzip</th>
<th>Equake</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>53 B</td>
<td>55 B</td>
<td>15 B</td>
<td>23 B</td>
</tr>
<tr>
<td>IPC</td>
<td>0.69</td>
<td>0.29</td>
<td>1.05</td>
<td>0.7</td>
</tr>
<tr>
<td>Instruction Mixes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loads</td>
<td>0.27</td>
<td>0.25</td>
<td>0.2</td>
<td>0.27</td>
</tr>
<tr>
<td>Stores</td>
<td>0.05</td>
<td>0.15</td>
<td>0.09</td>
<td>0.08</td>
</tr>
<tr>
<td>Branch's</td>
<td>0.14</td>
<td>0.2</td>
<td>0.17</td>
<td>0.12</td>
</tr>
<tr>
<td>Branch Misprediction Rates</td>
<td>0.025</td>
<td>0.07</td>
<td>0.08</td>
<td>0.02</td>
</tr>
<tr>
<td>Cache Miss Rates</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DL1</td>
<td>0.04</td>
<td>0.02</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>L2</td>
<td>0.48</td>
<td>0.06</td>
<td>0.03</td>
<td>0.30</td>
</tr>
<tr>
<td>Memory Footprint</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64 MB</td>
<td>24 MB</td>
<td>186 MB</td>
<td>42 MB</td>
<td></td>
</tr>
</tbody>
</table>
About This Talk

- **Some philosophy**
  - Why silicon? Why now? Why us (CMU)?

- **A quick tour: How speech recognition works**
  - What happens in a recognizer

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

- **Results**
  - ASIC version: Simulation results
  - FPGA version: Live, running hardware-based recognizer

This Talk: How to Get to Fast...

**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…”
Speech: Complex Task to do in Silicon

- ADC
- Feature extraction
- Feature Scoring
- Adaptation
- Sampling

Acoustic units → Words → Language

HMM / Viterbi Search
Language Model Search

Frontend
Ops: Low
SRAM (constants)

Scoring
Ops: High
SRAM (communic)

Backend
Ops: Medium
MBs SRAM

A Silicon Architecture: Breakdowns

- Acoustic Frontend
  - SRAM (constants)
- Gaussian Scoring
  - SRAM (communic)
- Backend Search
  - MBs SRAM (active recog)

Computations (Ops) | Low | High | Medium
SRAM (size)         | Small | Small | Large
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**

**Pipelined Get-Word and Get-HMM stages**

**Pipelined non-LanguageModel and LanguageModel stages**
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

---

About This Talk

- **Some philosophy**
  - Why silicon? Why now? Why us (CMU)?

- **A quick tour**: How speech recognition works
  - What happens in a recognizer

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

- **Results**
  - **ASIC version**: Simulation results
  - **FPGA version**: Live, running hardware-based recognizer
Design Flow: C++ Cycle Simulator → Verilog

- 2006 benchmark: 5K-word “Wall Street Journal” task
- Cycle sim 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>Hardware: Our Proposed Recognizer</td>
<td>6.725%</td>
<td>0.125 GHz</td>
<td>1.67X</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 → weeks

Speedup vs Software (over 330 WSJ utterances)
Aside: Pieces of Design = Great Class Projects

- CMU student team: Patrick Chiu, David Fu, Mark McCartney, Ajay Panagariya, Chris Thomas

A Complete Live Recognizer: FPGA Demo

- 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
FPGA Experimental Results

- Aside: as far as we know, this is the most complex recognizer architecture ever fully mapped into a running, 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
  - SoC and FPGA versions
  - ~10X realtime speedup architecture in progress at CMU

- Reflections
  - Some of the most interesting experiences happen when you get people from very different backgrounds – silicon + speech – on same team
Acknowledgements

- **Work supported by**
  - US National Science Foundation (www.nsf.gov)
  - Semiconductor Research Corporation (www.src.org)
  - FCRP Focus Research Center for Circuit & System Solutions (www.fcrp.org, www.c2s2.org)

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