Scalable Large-Vocabulary Continuous Speech Recognition

Katya Gonina
with Jike Chong, Kisun You, Youngmin Yi, Kurt Keutzer & others

UC Berkeley ParLab

January 30, 2012
Parallel scalability:

The ability for an application to efficiently utilize an increasing number of processing elements.

Intel Core i7 (45nm) 4 cores

NVIDIA GTX280 and GTX480 30 and 14 cores

Parallel scalability is required for software to obtain sustained performance improvements on successive generations of processors.
Outline

- Characteristics of Manycore Architectures
- Speech Recognition Application
  - Software architecture and characteristics
  - Important parallelization concerns
  - Design space explored for application scalability
- Design Space Evaluation
- Recognition Network Structure Evaluation
- Conclusion
Outline

- Characteristics of Manycore Architectures
- Speech Recognition Application
  - Software architecture and characteristics
  - Important parallelization concerns
  - Design space explored for application scalability
- Design Space Evaluation
- Recognition Network Structure Evaluation
- Conclusion
Parallel Platform Characteristics

- Multicore/manycore design philosophy
  - **Multicore**: Devote significant transistor resources to single thread performance
  - **Manycore**: Maximizing computation throughput at the expense of single thread performance

- Architecture Trend:
  - Increasing vector unit width (SIMD)
  - Increasing numbers of cores per die

- Application Implications:
  - Must increase data access regularity
  - Must optimize synchronization cost

We explore a design space for application scalability for a speech inference engine on multicore and manycore platforms
Outline

- Characteristics of Manycore Architectures
- **Speech Recognition Application**
  - Software architecture and characteristics
  - Important parallelization concerns
  - Design space explored for application scalability
- Design Space Evaluation
- Recognition Network Structure Evaluation
- Conclusion
Outline

- Characteristics of Manycore Architectures
- Speech Recognition Application
  - Software architecture and characteristics
  - Important parallelization concerns
  - Design space explored for application scalability
- Design Space Evaluation
- Recognition Network Structure Evaluation
- Conclusion
Challenges:
- Recognizing words from a large vocabulary arranged in exponentially many possible permutations
- Inferring word boundaries from the context of neighboring words
- Viterbi algorithm on Hidden Markov Models (HMM) is currently the most popular approach
Continuous Speech Recognition

- Inference engine system
  - Used in Sphinx (CMU, USA), HTK (Cambridge, UK), and Julius (CSRC, Japan)
- Modular and flexible setup
  - Shown to be effective for Arabic, English, Japanese, and Mandarin
Recognition Network

Gaussian Mixture Model for One Phone State
- Computing distance to each mixture component
- Computing weighted sum of all components

HMM Acoustic Phone Model

Pronunciation Model
- HOP hh aa p
- ON aa n
- POP p aa p

Bigram Language Model

Compiled HMM Recognition Network
Features from one frame

Gaussian Mixture Model for One Phone State
- Mixture
- 128 Components
- Computing distance to each mixture components
- Computing weighted sum of all components

HMM Acoustic Phone Model
- C level
- 17550 Triphones

Pronunciation Model
- L level
- 58k Word Vocabulary

WFST network
- H•C•L•G
- 4 million States
- 10 millions Transition Arcs
- WFST Recognition Network

Bigram Language Model
- G level
- 168k Bigram Transitions

HOP
ON
POP
Speech Inference: Detailed Algorithm

**HMM-based inference**

1. **Forward Pass**
   - Observations: Obs 1, Obs 2, Obs 3, Obs 4
   - States: State 1, State 2, State 3, State N

2. **Backward Pass**

**Equation:**

\[ m[t][s_i] = \max_{s_{t-1}} m[t-1][s_{t-1}] \cdot P(s_i|s_{t-1}) \cdot P(x_t|s_i) \]

**Legends:**
- A State
- A Pruned State
- An Observation

**Model size for a WFST language model**
- # states: 4 million
- # arcs: 10 million
- # observations: 100/sec
- Average # active states per time step: 10,000 – 20,000
ASR: Detailed Algorithm

1. Forward Pass
   - Voice Input
   - Speech Feature Extractor
   - Observation Sequence (Obs 1, Obs 2, Obs 3, Obs 4)
   - Time
     - States (State 1, State 2, State 3, State N)
     - Speech Model
     - Recognition Network
       - Acoustic Model
       - Pronunciation Model
       - Language Model

2. Backward Pass

In each iteration, perform beam search algorithm

Iterative through inputs one time step at a time

In each step, consider alternative interpretations

Speech: I think therefore I am
Inference Engine Architecture

- A highly hierarchical structure
  - An iterative outer loop over time steps
  - A pipeline of operations in each time step
  - A set of alternative hypothesis to advance

Sequential operation with iteration dependencies

- Compute Intensive
- Communication Intensive

Extensive fine-grained parallelism at the inner most level
Recognition Process

- **Phase 1:**
  - Observation probability computation
  - Highly compute intensive step

- **Phase 2:**
  - Traverse out-going arcs from active states
  - Write contention must be resolved at the destination states
  - Destination state is updated with most-likely in-coming arc

Recognition is a process of graph traversal
Inference Engine Challenges

- **Application Challenges**
  - Irregularity of network
  - Input-dependent, dynamically changing working set

- **Scalability Goals**
  - Expose sufficient concurrency
    1) Efficiently synchronize between an increasing number of concurrent tasks
    2) Effectively utilize all levels of parallel resources, including SIMD parallelism
Outline

- Characteristics of Manycore Architectures
- Speech Recognition Application
  - Software architecture and characteristics
  - Important parallelization concerns
  - Design space explored for application scalability
- Design Space Evaluation
- Recognition Network Structure Evaluation
- Conclusion
### Core Level Synchronization

**Challenge:**
- The cost for write conflict resolution can dominate runtime

**Experiment:**
- Allow traversal to either **propagate** from source or **aggregate** at destination for write conflict resolution

<table>
<thead>
<tr>
<th></th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Traversal by Propagation</strong></td>
<td>Easy to program, HW handles write conflicts transparently</td>
<td>Sensitive to atomic operation latency</td>
<td><img src="https://via.placeholder.com/150" alt="Figure" /></td>
</tr>
<tr>
<td><strong>Traversal by Aggregation</strong></td>
<td>Explicit resolution of write conflicts, no atomics</td>
<td>Overhead in building lists of to-be-updated destination states</td>
<td><img src="https://via.placeholder.com/150" alt="Figure" /></td>
</tr>
</tbody>
</table>
The fixed cost (overhead) of **aggregation** technique is significant.

Relative gradient of **propagation** and **aggregation** techniques depend on the efficiency of the platform in resolving write conflicts.

If no hardware atomics are available, using spin locks and semaphores will be costly.

If data structure requires multiple writes to the same destination states, significant contention can occur.
- **Challenge:**
  - Vector unit efficiency can quickly drop off with increased vector width

- **Experiment:**
  - Traverse the recognition network based on **active states** or **active arcs**

<table>
<thead>
<tr>
<th></th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active States</strong></td>
<td>Easy to program, all active arcs emit from active states</td>
<td>Load-imbalance, number of arcs varies per state</td>
<td><img src="image1.png" alt="Diagram of Current States and Next States" /></td>
</tr>
<tr>
<td><strong>Active Arcs</strong></td>
<td>Finer granularity, Load balance</td>
<td>More information to maintain more arcs than states</td>
<td><img src="image2.png" alt="Diagram of Current States and Next States" /></td>
</tr>
</tbody>
</table>
### Design Space

<table>
<thead>
<tr>
<th></th>
<th>Traversal by Propagation</th>
<th>Traversal by Aggregation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active States</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Current States</strong></td>
<td>Maintain active source states, propagate out-arc computation results to destination state</td>
<td>Maintain active destination states, determine all potential destination states and aggregate incoming arcs</td>
</tr>
<tr>
<td><strong>Next States</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Active Arcs</strong></td>
<td>Maintain active arcs, propagate active arc computation results to destination state</td>
<td>Maintain active arcs, group arcs with same destination states and aggregate active arcs locally to resolve write conflicts</td>
</tr>
</tbody>
</table>
# Hardware Platform

<table>
<thead>
<tr>
<th>Specifications</th>
<th>Core i7920</th>
<th>GTX280</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Elements</td>
<td>4 cores (SMT), 4 way SIMD @2.66 GHz</td>
<td>30 cores, 8 way physical, 32 way logical SIMD @1.3 GHz</td>
</tr>
<tr>
<td>SP GFLOP/s</td>
<td>85.1</td>
<td>933</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>25.6 GB/s</td>
<td>141 GB/s</td>
</tr>
<tr>
<td>Register File</td>
<td>-</td>
<td>1.875 MB</td>
</tr>
<tr>
<td>Local Store</td>
<td>-</td>
<td>480 kB</td>
</tr>
</tbody>
</table>
Outline

- Characteristics of Manycore Architectures
- Speech Recognition Application
  - Software architecture and characteristics
  - Important parallelization concerns
  - Design space explored for application scalability
- Design Space Evaluation
- Recognition Network Structure Evaluation
- Conclusion
### Efficiency vs Platform

**[Table 3]** Recognition performance normalized for 1 s of speech for different algorithm styles. Speedup reported over optimized sequential version of the propagation-by-states style.

<table>
<thead>
<tr>
<th>SECONDS (%)</th>
<th>CORE i7</th>
<th></th>
<th>CORE i7</th>
<th></th>
<th>GTX280</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQUENTIAL</td>
<td>PROP. BY</td>
<td>PROP.</td>
<td>AGGR. BY</td>
<td>PROP.</td>
<td>AGGR.</td>
<td>AGGR.</td>
</tr>
<tr>
<td>PHASE 1</td>
<td>STATES</td>
<td>STATES</td>
<td>STATES</td>
<td>STATES</td>
<td>STATES</td>
<td>STATES</td>
</tr>
<tr>
<td></td>
<td>2.623 (83%)</td>
<td>0.732 (79%)</td>
<td>0.737 (73%)</td>
<td>0.148 (49%)</td>
<td>0.147 (12%)</td>
<td>0.148 (16%)</td>
</tr>
<tr>
<td>PHASE 2</td>
<td>0.474 (15%)</td>
<td>0.157 (17%)</td>
<td>0.242 (24%)</td>
<td>0.103 (34%)</td>
<td>0.770 (64%)</td>
<td>0.469 (51%)</td>
</tr>
<tr>
<td>PHASE 3</td>
<td>0.073 (2%)</td>
<td>0.035 (4%)</td>
<td>0.026 (3%)</td>
<td>0.043 (14%)</td>
<td>0.272 (23%)</td>
<td>0.281 (31%)</td>
</tr>
<tr>
<td>SEQUENTIAL</td>
<td>OVERHEAD</td>
<td></td>
<td>OVERHEAD</td>
<td></td>
<td>OVERHEAD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.001</td>
<td>0.001</td>
<td>0.008 (1%)</td>
<td>0.008 (2.5%)</td>
<td>0.014 (1.2%)</td>
<td>0.014 (1.6%)</td>
</tr>
<tr>
<td>TOTAL</td>
<td>3.171</td>
<td>0.925</td>
<td>2.593</td>
<td>0.776</td>
<td>1.203</td>
<td>0.912</td>
</tr>
<tr>
<td>SPEEDUP</td>
<td>1</td>
<td>3.43</td>
<td>1.22</td>
<td>4.08</td>
<td>2.64</td>
<td>3.48</td>
</tr>
</tbody>
</table>
## Recognition Accuracy

<table>
<thead>
<tr>
<th>Avg. # of Active States</th>
<th>32820</th>
<th>20000</th>
<th>10139</th>
<th>3518</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word Error Rate</td>
<td>41.6</td>
<td>41.8</td>
<td>42.2</td>
<td>44.5</td>
</tr>
<tr>
<td>RTF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential</td>
<td>4.36</td>
<td>3.17</td>
<td>2.29</td>
<td>1.2</td>
</tr>
<tr>
<td>Multicore</td>
<td>1.23</td>
<td>0.93</td>
<td>0.70</td>
<td>0.39</td>
</tr>
<tr>
<td>Manycore</td>
<td>0.40</td>
<td>0.30</td>
<td>0.23</td>
<td>0.18</td>
</tr>
</tbody>
</table>
### Overall Speedup

- **Speed up varies between phases**
  - 4-20x for compute intensive phases
  - 3-4x for communication intensive phases
  - Communication intensive phases becoming proportionally more important
Synchronization Cost in Inference Engine Graph Traversal

- **Propagate** with Global Contention
- **Aggregate**

Graph showing the relationship between the number of arcs synchronized and the total synchronization cost in seconds. The graph includes data points indicating that the cost increases linearly with the number of arcs synchronized.
SIMD Utilization Efficiency

<table>
<thead>
<tr>
<th></th>
<th>State Based</th>
<th>Arc Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time taken</td>
<td>756.79 ms</td>
<td>81.74 ms</td>
</tr>
<tr>
<td>Speedup</td>
<td>1x</td>
<td>9.25x</td>
</tr>
</tbody>
</table>
• Characteristics of Manycore Architectures
• Speech Recognition Application
  ▪ Software architecture and characteristics
  ▪ Important parallelization concerns
  ▪ Design space explored for application scalability
• Design Space Evaluation
• Recognition Network Structure Evaluation
• Conclusion
Recognition Network Representation

- Significant effort put into optimizing recognition networks
- Starting at baseline Linear Lexical Models
  - One chain of states per word
- Tree-lexical
- Finite state machine techniques to construct WFST

What implications does the structure have on efficiency of parallel speech inference algorithms?
Linear-Lexical Model vs WFST

A sample Linear Lexical Model Network

A section of a Weighed Finite State Transducer Network
Explicitly handles two types of transitions
- Within-word
- Across-word

Optimized data layout for each type
- First states for each word stored consecutive for across-word transitions
  - Chains of within-word states stored as a chain
- Across-word transitions – all-to-all dense computation
  - Extremely efficient on the GPU
Results

- Wall Street Journal 5K Corpus

![Graph showing comparison between LLM and WFST for speed and error rate. The graph indicates a 22X improvement in speed with LLM.]
Results

GTX285 Results

WER (%) vs Real Time Factor
Execution Time

Execution speed measured at 8.90% WER

<table>
<thead>
<tr>
<th>Real Time Factor</th>
<th>LLM</th>
<th>WFST</th>
<th>GTX285</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.14</td>
<td>8.23x</td>
<td>7.78x</td>
<td></td>
</tr>
<tr>
<td>0.12</td>
<td>18%</td>
<td>51%</td>
<td></td>
</tr>
<tr>
<td>0.10</td>
<td>45%</td>
<td>10%</td>
<td>26%</td>
</tr>
<tr>
<td>0.08</td>
<td>23%</td>
<td>12%</td>
<td></td>
</tr>
<tr>
<td>0.06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data Gathering | Observation Prob | Graph Traversal | Sequential Overhead
Outline

- Characteristics of Manycore Architectures
- Speech Recognition Application
  - Software architecture and characteristics
  - Important parallelization concerns
  - Design space explored for application scalability
- Design Space Evaluation
- Recognition Network Structure Evaluation
- Conclusion
Conclusions

- Scalable software architecture for speech recognition inference engine
  - 2.5% sequential overhead
- Explored algorithmic design space
  - Fastest algorithm depends on platform
  - Core synchronization and SIMD optimization are important for scalability
- Explored recognition network representation
  - Simpler, more regular LLM representation very competitive with highly-optimized, more irregular WFST
Current and Future Work

- Efficient training of acoustic models (GMMs)
- Productive parallel computing for application writers
  - Not have to go through this process every time
- Automating parallelization techniques
  - High-level code transformation
  - Just-in-time compilation
  - Code variant selection
- What is the best (parallel) platform for a particular algorithm?
Thank you!

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, Nokia, NVIDIA, Oracle, and Samsung.
References


In the Hidden Markov Model, states are *hidden*, because phones are *indirectly observed*.

One must infer the *most likely interpretation* of the signal while taking the model of the *underlying language* into account.
Detailed Speedup: Multicore

**Sequential**
RTF: 3.17; 1x

- 2.623
- 0.474
- 0.073

**Arc-based Propagation**
RTF: 1.006; 3.2x

- 0.737
- 0.242
- 0.026
- 0.001

**State-based Propagation**
RTF: 0.925; 3.4x

- 0.732
- 0.157
- 0.035
- 0.001

**State-based Aggregation**
RTF: 2.593; 1.2x

- 0.754
- 1.356
- 0.482
- 0.001

**Overhead**

<table>
<thead>
<tr>
<th>Phase</th>
<th>Seq.</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase 1</td>
<td>2.6</td>
<td>0.073</td>
</tr>
<tr>
<td>Phase 2</td>
<td>0.47</td>
<td>0.026</td>
</tr>
<tr>
<td>Phase 3</td>
<td>0.07</td>
<td>0.001</td>
</tr>
</tbody>
</table>

**Real Time Factor**

- RTF: 3.17
- Speedup: 3.2x vs Seq.

**Detailed Analysis**

- Phase 1: 2.623
- Phase 2: 0.474
- Phase 3: 0.073

**Speedup**

- Phase 1: 3.2x
- Phase 2: 3.2x
- Phase 3: 3.2x

**Overhead**

- Seq.: 0.073
- Phase 1: 0.026
- Phase 2: 0.001
- Phase 3: 0.001
Detailed Speedup: Manycore

Sequential
RTF: 3.17; 1x

Arc-based Propagation
RTF: 0.302; 10.5x
- 0.148
- 0.103
- 0.043
- 0.008

State-based Propagation
RTF: 0.776; 4.1x
- 0.148
- 0.512
- 0.108
- 0.008

Arc-based Aggregation
RTF: 0.912; 3.5x
- 0.148
- 0.469
- 0.281
- 0.014

State-based Aggregation
RTF: 1.203; 2.6x
- 0.147
- 0.77
- 0.272
- 0.014

Overhead

<table>
<thead>
<tr>
<th>RTF: Real Time Factor</th>
<th>Speedup: Seq vs Seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase 1</td>
<td>Phase 2</td>
</tr>
<tr>
<td>Phase 3</td>
<td>Seq.</td>
</tr>
</tbody>
</table>
Next Steps

- Experiment on two more sets of models
  - Telephone conversations (optimizing for batch model processing)
  - News Broadcast (optimizing for real time processing)

- Construct the application framework for domain experts to develop speech applications
  - Search for industry use cases to substantiate usage scenarios
LVCSR Application Framework

**Top Level Attributes**

Customizable attributes:
- Recognition network structure
- Input waveform format
- Output word sequence format

Data Structure:
- Feature vector format

Fixed Structure:
- Feature extractor
- Inference engine

**Feature Extractor**

Customizable Function:
- Feature extraction algorithm

**Recognition Network**

- Acoustic Model
- Pronunciation Model
- Language Model

**Inference Engine**

Customizable Functions:
- Observation/Arc probability computation
- Pruning heuristics
- Track back data logging

Framework architecture customization:
- States vs arc based traversal
- Propagate vs aggregate traversal techniques

**Inference Engine**

I think therefore I am
Frameworks for Parallel Programming

- End User
- Application Developer
- Application Framework Developer
- Algorithmic Framework Developer
- Programming Framework Developer
- Hardware Architect

LVCSR Framework

Graph Traversal Framework

<table>
<thead>
<tr>
<th>Traversal by Propagation</th>
<th>Active States</th>
<th>Active Arcs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Current States Next States</td>
<td>Current States Next States</td>
</tr>
<tr>
<td></td>
<td>Maintain active source states, propagate out-arc computation results to destination state</td>
<td>Maintain active arcs, propagate active arc computation results to destination state</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Traversal by Aggregation</th>
<th>Active States</th>
<th>Active Arcs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Current States Next States</td>
<td>Current States Next States</td>
</tr>
<tr>
<td></td>
<td>Maintain active destination states, determine all potential destination states and aggregate incoming arcs</td>
<td>Maintain active arcs, group arcs with same destination states and aggregate active arcs locally to resolve write conflicts</td>
</tr>
</tbody>
</table>
Discussion: Load Balancing

- Core level load balancing is an important issue
  - Many prior work has been limited by across core work load imbalance
- Application developers want to expose parallelism, not managing the detail
  - Best solved by implementation platform support

- Multicore:
  - Task queue abstraction with distributed queue and lazy work stealing [15]

- Manycore:
  - Hardware managed dynamic load balancing based on the CUDA runtime environment [16]


Currently, the memory hierarchy differs significantly between Intel multicore and NVIDIA manycore

- Requires different data structure for optimal performance

**Multicore:**
- Reference data in main memory, working set mostly cached in L3

**Manycore:**
- Create temporary coalesced array for working set, stored in GDDR, streaming access
Speech Inference Engine Implementation

<table>
<thead>
<tr>
<th>GPU Data</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Set</td>
<td>R</td>
</tr>
<tr>
<td>LM</td>
<td>W</td>
</tr>
<tr>
<td>HMM</td>
<td>W</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU Control</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Files</td>
<td></td>
</tr>
<tr>
<td>Initialize data structures</td>
<td></td>
</tr>
</tbody>
</table>

Phase 0: Prepare ActiveSet

Phase 1: Compute Observation Probability

Phase 2: For each active arc:
- Compute arc transition probability

Copy results back to CPU

Collect Backtrack Info

Backtrack

Output Results
Recognition Network Representation

- **Linear-Lexical Model (LLM) – baseline implementation**
  - Models each word as a chain of triphone states
  - Highly redundant
  - Language model from word-to-word transitions

- **Weighted Finite State Transducer (WFST)**
  - Combines pronunciation and language models
  - Takes advantage of sparsity of natural languages
  - Remove redundant states and arcs
  - Faster recognition speed on *sequential* processors
Software Must Use Hardware Parallelism

Hardware Trends

Software Trends

Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)

Voice Input

Recognition Output

r eh k ax g n ay z s p i y ch

ASR

Recognize

Speech

Transistors (000)
Clock Speed (MHz)
Power (W)
Perf/Clock (LP)