Parallel HMMs

Parallel Implementation of Hidden Markov Models for Wireless Applications

Authors
Shawn Hymel (Wireless@VT, Virginia Tech)
Ihsan Akbar (Harris Corporation)
Jeffrey Reed (Wireless@VT, Virginia Tech)
Agenda

• Overview of GPGPU
• Overview of HMMs
• Parallelization
• Results
• Applications
• Why Is This Useful?
General-Purpose Processing on GPUs

• CUDA-specific
• Important Terms:
  – Threads
  – Blocks
  – Grid
CUDA Code Flow

1. Copy processing data
2. Instruct the processing
3. Execute parallel in each core
4. Copy the result
Hidden Markov Model

\[ \lambda = (A, B, \pi) \]

\[ A = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \]

\[ B = \begin{bmatrix} 0.1 & 0.6 \\ 0.4 & 0.3 \end{bmatrix} \]

\[ \pi = [0.6 \ 0.4] \]
HMM Canonical Problems

• Evaluation: $P(O|\lambda)$
  – Forward Algorithm
  – Backward Algorithm

• Find the most likely state sequence
  – Viterbi Algorithm

• Training (maximize $P(O|\lambda)$)
  – Baum-Welch Algorithm
Forward Algorithm

Given a model and an observation sequence, calculate $P(O | \lambda)$

- $T =$ number of observations
- $N =$ number of states
- $M =$ number of possible symbols

Initiation:

$$\alpha_1(i) = \pi_i b_i(O_1), \ i = 1,2,...N$$

Induction:

$$\alpha_{t+1}(j) = \sum_{i=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1})$$

Termination

$$P(O | \lambda) = \sum_{i=1}^{N} \alpha_T(i)$$
Example of Parallelization

\[
\alpha_{t+1}(j) = \sum_{i=1}^{N} \alpha_{t}(i) a_{ij} b_{j}(O_{t+1})
\]

For all \( j \), matrix multiplication

For all \( j \), element-by-element multiplication

We can perform this step in parallel!
\( O(TN^2) \rightarrow O(T \log N) \)
## Computational Complexity

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Serial</th>
<th>Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward Algorithm</td>
<td>$O(TN^2)$</td>
<td>$O(T \log N)$</td>
</tr>
<tr>
<td>Viterbi Algorithm</td>
<td>$O(TN^2)$</td>
<td>$O(T \log N)$</td>
</tr>
<tr>
<td>Baum-Welch Algorithm</td>
<td>$O(TN^2)$ or $O(TMN)$</td>
<td>$O(T \log N)$</td>
</tr>
</tbody>
</table>
Test Procedures

- Time execution of each algorithm (C vs. CUDA)
  - Vary states
  - Vary symbols
  - Vary sequence length
- Calculate total energy consumption (C vs. CUDA)
  - PowerTOP software

<table>
<thead>
<tr>
<th>Test Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Component</strong></td>
</tr>
<tr>
<td>CPU</td>
</tr>
<tr>
<td>GPU</td>
</tr>
<tr>
<td>GPU Core Speed</td>
</tr>
<tr>
<td>GPU Shader Speed</td>
</tr>
<tr>
<td>GPU Memory Speed</td>
</tr>
<tr>
<td>CUDA Cores</td>
</tr>
</tbody>
</table>
# Speed Results

<table>
<thead>
<tr>
<th>Number of States</th>
<th>CPU Runtime (s)</th>
<th>GPU Runtime (s)</th>
<th>Speed Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Forward Algorithm</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.001</td>
<td>0.1531</td>
<td>0.007x</td>
</tr>
<tr>
<td>40</td>
<td>0.04</td>
<td>0.1393</td>
<td>0.287x</td>
</tr>
<tr>
<td>400</td>
<td>4.2816</td>
<td>0.2379</td>
<td>17.99x</td>
</tr>
<tr>
<td>4000</td>
<td>534.2028</td>
<td>2.9495</td>
<td>181.12 x</td>
</tr>
<tr>
<td><strong>Viterbi Algorithm</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.0033</td>
<td>0.1605</td>
<td>0.021x</td>
</tr>
<tr>
<td>40</td>
<td>0.0436</td>
<td>0.1801</td>
<td>0.242x</td>
</tr>
<tr>
<td>400</td>
<td>4.2684</td>
<td>1.6595</td>
<td>2.57x</td>
</tr>
<tr>
<td>4000</td>
<td>534.5543</td>
<td>116.2531</td>
<td>4.60 x</td>
</tr>
<tr>
<td><strong>Baum-Welch Algorithm</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.0021</td>
<td>0.4142</td>
<td>0.005x</td>
</tr>
<tr>
<td>40</td>
<td>0.1946</td>
<td>0.4299</td>
<td>0.453x</td>
</tr>
<tr>
<td>400</td>
<td>17.6719</td>
<td>0.7502</td>
<td>23.56x</td>
</tr>
<tr>
<td>4000</td>
<td>1834.672</td>
<td>28.1271</td>
<td>65.23 x</td>
</tr>
</tbody>
</table>
Energy Consumption

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Power (W)</th>
<th>States to Break Even</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
<td>CUDA</td>
</tr>
<tr>
<td>Forward</td>
<td>18.5</td>
<td>26.5</td>
</tr>
<tr>
<td>Viterbi</td>
<td>18.5</td>
<td>29.1</td>
</tr>
<tr>
<td>BWA</td>
<td>18.3</td>
<td>26.1</td>
</tr>
</tbody>
</table>

Energy Consumption for Forward Algorithm

Energy Consumed (kWh)

Number of States

0 50 100 150 200 250
Applications

• Pattern Recognition
  – Spectrum Sensing
  – Signal Classification
  – Specific Emitter Identification
  – Geolocation

• Modeling
  – Channel Fading
  – Call Drop Prediction
Why Is This Useful?

- Evolution of GPUs and multi-core processors
  - Smart phones, tablets, SDR
  - Co-processor
- Utilize existing hardware for HMM applications
  - Large number of states
  - 2D/3D HMMs
- Uses in other fields (speech recognition, computer vision)
- Extrapolation to other algorithms (pattern recognition)
Questions?

Contact Information
Email: hymelsr@vt.edu
Blog: http://sgmustadio.wordpress.com/
Code: http://code.google.com/p/hmm-cuda/

Other Good Resources
cuHMM: http://code.google.com/p/chmm/
HTK: http://htk.eng.cam.ac.uk/
Supporting Slide: Reductions

MATLAB example:

```matlab
>> sum(A)
```

C Implementation:

```c
sum = 0;
for (i = 0; i < length; i++) {
    sum = sum + A[i];
}
```

Parallelization:

Reducing arrays to a single value (e.g. sum) go from O(N) to O(log N)
Supporting Slide: Timing Results (Forward)

**Execution Time for Forward Algorithm**

- **Vary States**
- **Vary Symbols**

**Execution Time for Forward Algorithm on GPU**

**Execution Time for Forward Algorithm**
Supporting Slide: Timing Results (Viterbi)

Vary States

Vary Symbols

Execution Time for Viterbi Algorithm

- CPU
- GPU

Execution Time for Viterbi on GPU

Execution Time for Viterbi Algorithm

- CPU
- GPU
Supporting Slide: Timing Results (BWA)

Vary States

Vary Symbols