Parallelizing Sequential Algorithms

Madan Musuvathi
Microsoft Research

joint work with
Saeed Maleki, Univ. of Illinios, Urbana-Champaign
Todd Mytkowicz, Microsoft Research
Hardware is Parallel

But, many important algorithms are ‘inherently sequential’
Motivating Problem

Grep terabytes in seconds

Sequentially reading terabytes from disk takes hours
Grep implementations are sequential
Breaking Sequential Dependences

c = g(b)
Two Parallel Algorithms

Finite State Machines
30x faster on 12 cores

Dynamic Programming
Fastest software Viterbi decoder
Higher throughput on 16 cores than a commercial FPGA implementation
Parallel Finite State Machines
FSM Applications

- grep (regex matching)
- lex (tokenization)
- Dictionary-based decoding (e.g. Huffman decoding)
- Text encoding/decoding (e.g. UTF8)
Data Dependence limits ILP, SIMD, and multicore parallelism

state = $a$;

\[
\text{foreach}(\text{input in})
\]

\[
\text{state} = T[in][\text{state}];
\]
Breaking Dependences with Enumeration

\[ P_0 \]

\[
\begin{array}{ccccccc}
/ & * & x & x & x & * & * \\
--- & --- & --- & --- & --- & --- & ---
\end{array}
\]

\[ P_1 \]

\[
\begin{array}{ccccccc}
/ & * & x & x \\
--- & --- & --- & ---
\end{array}
\]

\[ a \rightarrow b \rightarrow c \rightarrow \ldots \]

\[
\begin{array}{c|c|c|c}
T & / & * & x \\
\hline
a & b & a & a \\
b & b & c & a \\
c & c & d & d \\
d & a & d & c \\
\end{array}
\]

Cost of enumeration \( \propto \) number of states
Convergence Reduces the Cost of Enumeration

\[ P_0 \]

\[ / \quad * \quad x \quad x \quad x \quad * \quad * \quad / \quad x \quad x \]

\[ a \rightarrow b \rightarrow c \rightarrow \ldots \]

\[ T \quad / \quad * \quad x \]

<table>
<thead>
<tr>
<th></th>
<th>/</th>
<th>*</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>d</td>
<td>c</td>
</tr>
</tbody>
</table>

\[ P_1 \]
Convergence Reduces the Cost of Enumeration

Cost of enumeration $\propto$ number of unique states
Convergence for Worst-Case Inputs

90% of FSMs converge to <= 16 states after 10 steps

Only 65% of FSMs converge to <= 4 states even after 5000 steps
Single-Core Performance when using SIMD

Good performance

Not so good performance
Multicore Performance for Bing Tokenization
Parallelizing
Dynamic Programming
Optimization Problems Solved using Dynamic Programming

\[ p_{i,j} = \max_k (p_{i-1,k} \times t_{k,j}) \]

Viterbi

\[ C_{i,j} = \max \left\{ C_{i-1,j-1} + \delta_{i,j}, C_{i,j-1}, C_{i-1,j} \right\} \]

diff
Parallelize Across Stages

Assume recurrence is of the form

\[ s_i[j] = \min_k (C_{ijk} + s_{i-1}[k]) \]
Optimization problem = Finding shortest path in some graph
Break Dependences with All-Pair Shortest Paths

source to all boundary nodes

all boundary nodes to destination

Overhead $\propto$ stage size
All-Pair Shortest Paths Converge
Convergence in LCS
Results – Viterbi Decoder
Conclusions

Parallel algorithms for FSMs and dynamic programming
Using dynamic convergence properties

Can we break dependences for other algorithms?
Parsing
Iterative machine learning
Graph algorithms

Can we automatically parallelize across dependences?
Save the planet and return your name badge before you leave (on Tuesday)