Scalable String Matching on the Cell BE Processor

Daniele Scarpazza, Oreste Villa, Fabrizio Petrini
Applied Computer Science Group
Pacific Northwest National Laboratory
fabrizio.petrini@pnl.gov

Georgia Tech, Sony/Toshiba/IBM Workshop on Software and Applications for the Cell BE Processor
Atlanta, GA, June 19 2007
The problem
- Network Intrusion Detection Systems (NIDS) are becoming an essential part of data centers
- At the heart of a NIDS there is a string matching algorithm

The Aho-Corasick algorithm
- A Deterministic Finite Automaton (DFA)

Multicore Processors
- An interesting opportunity to accelerate keyword scanning
- Most of existing work done on FPGAs/specialized processors

Goals and challenges
- Scalability of the dictionary and the network speed

DFAs with very high speed
- Two SPEs can handle a 10 Gbit/sec rate with a transition table of less than 200KB
The advent of teraflop-scale, many-core processors.

![Diagram showing the progression from Medieval Times to the Industrial Age through the Renaissance Period, highlighting the development of cores and threads.](image-url)

- **Medieval Times**: Small Number Of Traditional Cores
- **Renaissance Period**: Arrays of Throughput Cores
- **Industrial Age**: SMT

Courtesy of Doug Carmean, Intel
Set Pattern Matching Problem

- Find patterns in text
- $P = \{P_1, P_2, \ldots, P_q\}$, in $T$
- Aho and Corasick proposed an interesting algorithm for multi-pattern string matching
- Uses a state machine
- Important problem in a number of fields
  - Text processing, biology, network security, etc.
Aho Corasick - Example

\[ P = \{\text{her, iris, he, is}\} \]

\[ T = \text{“the iris for her”} \]
Aho Corasick - Example

\[ P = \{ \text{her, iris, he, is} \} \]

\[ T = \text{“the iris for her”} \]

Travel along the Goto function, which is a trie of all patterns.
Aho Corasick - Example

\[ P = \{ \text{her, iris, he, is} \} \]

\[ T = \text{“the iris for her”} \]

Travel along the Goto function, which is a trie of all patterns.
First Step: Keyword Tree

![Diagram of a keyword tree with nodes labeled t, m, a, h, e, and edges connecting them.](image)
Second Step: Failed Transitions (Non-deterministic Finite Automaton NFA)
Extend Failed Transitions for Each Character
Build an Optimized Deterministic Finite Automaton (DFA)
Design Challenges: Speed vs Size of the Dictionary

Throughput (Gbps)

Dictionary size (patterns)

(hard performance boundary)

~200 ~1,500 ~100,000 ~millions

(?)

Pacific Northwest National Laboratory
U.S. Department of Energy
Mapping the Aho-Corasick Algorithm on the Cell Processor: Data Streaming and SIMD parallelism
Aho-Corasick: A Multi-level Parallelization

General approach

- Multithreaded parallelism within a Synergistic Processing Unit (SPU), using multiple segments/connections of the input stream
- SIMD parallelism, pipeline parallelism (even/odd pipelines of the SPU)
- An arsenal of techniques: loop unrolling, removing speculation, restricted pointers, etc.

Using multiple SPUs to increase processing bandwidth/dictionary size

Dynamic loading of dictionaries
Aggregate Main Memory Bandwidth: Memory Access Traffic Explicitly Orchestrated at User-Level

![Graph showing aggregate memory bandwidth with different object sizes](image-url)
SIMD and Pipeline Parallelism

16 Interleaved input streams
16 input characters
16 input symbols
16 offsets to the transition table cells

Current state pointers for the 16 DFAs

Addresses to the cells containing the next state pointers

State Transition Table

Next state pointers for the 16 DFA
Final state flags for the 16 DFA

16 loads
16 SISD add
16 SISD ands
16 SISD ands

SIMD shl
SIMD shr

0xFFFFFFFE
0x00000001

16 SISD add
16 input characters
16 input symbols
16 offsets to the transition table cells

Current state pointers for the 16 DFAs
Addresses to the cells containing the next state pointers

State Transition Table

Next state pointers for the 16 DFA
Final state flags for the 16 DFA

16 loads
16 SISD add
16 SISD ands
16 SISD ands

SIMD shl
SIMD shr

0xFFFFFFFE
0x00000001
## Local Storage Usage

<table>
<thead>
<tr>
<th>Case</th>
<th>DFA State Transition Table</th>
<th>Input Buffer 0</th>
<th>Input Buffer 1</th>
<th>Code and Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1520 states, 32 input symbols</td>
<td>190 k</td>
<td>16 k</td>
<td>34 k</td>
</tr>
<tr>
<td>2</td>
<td>1648 states, 32 input symbols</td>
<td>206 k</td>
<td>8 k</td>
<td>34 k</td>
</tr>
<tr>
<td>3</td>
<td>1712 states, 32 input symbols</td>
<td>214 k</td>
<td>4 k</td>
<td>34 k</td>
</tr>
</tbody>
</table>

Total local store size: 256 k
Overlapping Computation with Communication

Time

Computation

- Process buffer 0 (25.64 us)
- Process buffer 1 (25.64 us)
- Process buffer 0 (25.64 us)

Data transfer

- Load buffer 0 (5.94 us)
- Load buffer 1 (5.94 us)
- Load buffer 0 (5.94 us)
- Load buffer 1 (5.94 us)
## Schedule of a Dynamic State Transition Table (STT) Replacement

<table>
<thead>
<tr>
<th>Time</th>
<th>Computation</th>
<th>Data transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Process buffer 0 (match against STT 0) (25.64 us)</td>
<td>Load input to buffer 0 (5.94 us)</td>
</tr>
<tr>
<td></td>
<td>Process buffer 1 (match against STT 0) (25.64 us)</td>
<td>Load input to buffer 1 (5.94 us)</td>
</tr>
<tr>
<td></td>
<td>Process buffer 0 (match against STT 1) (25.64 us)</td>
<td>Load next STT into STT 1 chunk 1/2 (48 kbyte) (17.83 us)</td>
</tr>
<tr>
<td></td>
<td>Process buffer 1 (match against STT 1) (25.64 us)</td>
<td>Load next STT into STT 1 chunk 2/2 (47 kbyte) (17.46 us)</td>
</tr>
<tr>
<td></td>
<td>Process buffer 0 (match against STT 0) (25.64 us)</td>
<td>Load input to buffer 1 (5.94 us)</td>
</tr>
<tr>
<td></td>
<td>Process buffer 1 (match against STT 1) (25.64 us)</td>
<td>Load next STT into STT 1 chunk 1/2 (48 kbyte) (17.83 us)</td>
</tr>
<tr>
<td></td>
<td>Process buffer 0 (match against STT 0) (25.64 us)</td>
<td>Load input to buffer 0 (5.94 us)</td>
</tr>
<tr>
<td></td>
<td>Process buffer 1 (match against STT 1) (25.64 us)</td>
<td>Load next STT into STT 0 chunk 2/2 (47 kbyte) (17.46 us)</td>
</tr>
<tr>
<td></td>
<td>Process buffer 0 (match against STT 0) (25.64 us)</td>
<td>Load input to buffer 1 (5.94 us)</td>
</tr>
<tr>
<td></td>
<td>Process buffer 1 (match against STT 1) (25.64 us)</td>
<td>Load next STT into STT 1 chunk 1/2 (48 kbyte) (17.83 us)</td>
</tr>
</tbody>
</table>
Throughput Provide by the STT replacement with a variable number of tiles (1 to 8)
Conclusion

- Multi-core processors competitive with FPGAs and specialized network processors
- Multiple data streaming options to perform string matching
- Performance from 40 Gbits/sec to 5 Gbits/sec
  - With small dictionaries
- Future work includes
  - Addressing larger dictionaries
  - Compression of the STT