SEQUENTIAL PATTERN MINING

WITH

THE MICRON'S AUTOMATA PROCESSOR

Ke Wang, Elaheh Sadredini, Kevin Skadron
Center for Automata Computing
Department of Computer Science
University of Virginia
Outline

- Micron’s Automata Processor (AP)
- SPM on the AP: Opportunities and Challenges
- AP Accelerated SPM and optimizations
- Performance Evaluation
- Conclusions and the Future Work
Sequential Pattern Mining

Sequential Pattern Mining (SPM):

- discovers frequent sequence of transactions
- the order of items within a transaction doesn’t matter, but the order of transaction is important

- Customer purchase patterning analysis
- Correlation analysis of storage system
- Web log analysis
- Software bug tracking
- Software API usage tracking

<table>
<thead>
<tr>
<th>Trans.</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;{Bread, Milk}, {Coke}&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;{Bread, Milk, Diaper}, {Beer, Eggs}, {Diaper}&gt;</td>
</tr>
<tr>
<td>3</td>
<td>&lt;{Milk}, {Diaper}, {Beer, Coke}&gt;</td>
</tr>
<tr>
<td>4</td>
<td>&lt;{Bread, Milk, Diaper}, {Beer, Diaper}, {Beer, Coke, Eggs}&gt;</td>
</tr>
<tr>
<td>5</td>
<td>&lt;{Bread, Milk}, {Coke}, {Diaper}, {Eggs}&gt;</td>
</tr>
</tbody>
</table>
Seq. Pat. Mining vs. Freq. Set. Mining

Similarity

- Can be solved by Apriori-based algorithm
- Massive pattern matching is the bottleneck

Differences

- Hierarchical patterns in SPM
  - The sequence of itemsets (transactions) are considered
    - The items within an itemset could be unordered and discontinuous
    - The itemsets within an a sequence could be discontinuous but the order should be checked

- Larger search space
  - For a given number of total items in a sequential pattern - k, the total number of structures is $2^{(k-1)}$
  - Search space needs to be reduced by Apriori-based algorithm (GSP)
  - Automaton design space needs to be reduced to save reconfiguration time
Data preprocessing:
1) Filter out infrequent items
2) Recode -> 8-bit / 16-bit symbols
3) Recode transactions
4) Sort items in transactions
5) Connect transactions by an itemset delimiter (\(x_{253}\))
6) Connect sequences by a sequence delimiter (\(x_{254}\))

Encoding:
- \(freq\_item\# < 254\): 8-bit
- \(253 < freq\_item\# < 64009\): 16-bit
Automata Design for SPM: Flattened

(a) Automaton for matching sequence $< \{1, 50\}, \{15, 80\}>$
Automata Design for SPM: Multi-entry

(a) AP macro for sequential pattern

(b) Automaton for sequence $<\{12, 79, 95\}>$

(c) Automaton for sequence $<\{33, 80\}11>$

(d) Automaton for sequence $<\{17\}2\{90\}>$
Performance Evaluation – AP

- **Target hardware:** D480 X 32
- **8-bit encoding:**
  - Speed: 7.5ns/item
  - ![Table](http://example.com/table8bit.png)

One D480 AP chip has 192 AP blocks. One 32-chip AP board has 6144 blocks.

- **16-bit encoding:**
  - Speed: 15ns/item
  - ![Table](http://example.com/table16bit.png)

- **Connection reconfiguration time:** 5ms for an entire board
- **Symbol replacement time:** 45ms for an entire board
Performance Evaluation - Comparison

- Compare with other implementations
  1. Java multi-threading implementation of GSP: GSP-Java
  2. C sequential implementation of GSP: GSP-1C
  3. OpenMP multi-threading implementation of GSP: GSP-6C
  4. GPU implementation of GSP: GSP-1G
  5. Java multi-threading implementation of PrefixSpan
  6. Java multi-threading implementation of SPADE

- Testing platform
  - CPU: Intel CPU i7-5820K (6 physical cores, 3.30GHz)
  - Mem: 32GB, 1.333GHz
  - GPU: Nvidia Kepler K40, 706 MHz clock, 2888 CUDA cores, 12 GB global memory
## Performance Evaluation - Datasets

- **Six real-world datasets**

<table>
<thead>
<tr>
<th>Name</th>
<th>Sequences#</th>
<th>Aver. Len.</th>
<th>Item#</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMS1</td>
<td>59601</td>
<td>2.42</td>
<td>497</td>
<td>1.5</td>
</tr>
<tr>
<td>BMS2</td>
<td>77512</td>
<td>4.62</td>
<td>3340</td>
<td>3.5</td>
</tr>
<tr>
<td>Kosarak</td>
<td>69998</td>
<td>16.95</td>
<td>41270</td>
<td>4.0</td>
</tr>
<tr>
<td>Bible</td>
<td>36369</td>
<td>17.84</td>
<td>13905</td>
<td>5.4</td>
</tr>
<tr>
<td>Leviathan</td>
<td>5834</td>
<td>33.8</td>
<td>9025</td>
<td>1.3</td>
</tr>
<tr>
<td>FIFA</td>
<td>20450</td>
<td>34.74</td>
<td>2990</td>
<td>4.8</td>
</tr>
</tbody>
</table>

Aver. Len. = Average number of items per sequence.
Performance Evaluation – GSP implementations

BM2

Kosarak
Performance Evaluation – GSP implementations (2)

Bible

FIFA
Performance Evaluation – Analysis

BM2

Kosarak
As sup decreasing, un-parallelized candidate generation becomes a new bottleneck.

The STE symbol replacement dominates the AP processing time.

BM2

Kosarak
Performance Evaluation – Compare with PrefixSpan and SPADE
Performance Evaluation – Compare with PrefixSpan and SPADE (2)
Performance Evaluation – input size

Kosarak

Leviathan
Conclusions and Future Work

- We present a hardware-accelerated solution for sequential pattern mining (SPM), using Micron’s AP
- Our proposed solution adopts the algorithm framework of the Generalized Sequential Pattern (GSP)
- We derive a compact automaton design for matching and counting frequent sequences.
  1) flatten sequences into strings by using delimiters and place-holders
  2) multiple-entry NFA strategy is proposed to accommodate variable-structured sequences
Together, this allows a single, compact template to match any candidate sequence of a given length, so this template can be replicated to make full use of the capacity and massive parallelism of the AP

- Up to 430X, 90X, and 29X speedups are achieved by the AP-accelerated GSP on six real-world datasets, when compared with the single-threaded CPU, multicore CPU, and GPU GSP implementations, respectively
- By parallelizing candidate generation, up to 452X and 49X over PrefixSpan and SPADE
- More speedup on larger datasets
Backups
GSP Algorithm