Frequent Subtree Mining on the Automata Processor: Challenges and Opportunities

Elaheh Sadredini, Reza Rahimi, Ke Wang, Kevin Skadron
Department of Computer Science
University of Virginia

Presenting in:
International Conference on Supercomputing, June 13-16, 2017
Motivation
Motivation
Motivation

WEB MINING

Google  amazon

Walmart

NLP

\[ x_2 \]

\[ x_1 \]
What Is the Frequent Subtree Mining (FTM)?

- To efficiently enumerate all frequent subtrees in a forest (database of trees) according to a given minimum support
- The support of a subtree is the number of subtrees in D that contains one occurrence of S
- A subtree S is frequent if its support is more than or equal to a user specified minimum support value

Relative support threshold: 60%
What Is the Frequent Subtree Mining (FTM)?

- To efficiently enumerate all frequent subtrees in a forest (database of trees) according to a given minimum support.
- The support of a subtree is the number of subtrees in D that contains one occurrence of S.
- A subtree S is frequent if its support is more than or equal to a user specified minimum support value.

Support = 3

Is frequent: 

Relative support threshold: 60%
Preliminaries

Induced subtree

Embedded subtree

Subtree

Tree

Subtree

Tree
## Issues with the Current FTM Solutions (1)

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BFS</strong></td>
<td>Massive pruning</td>
</tr>
<tr>
<td></td>
<td>Memory efficient</td>
</tr>
<tr>
<td></td>
<td>Multi-pass of dataset</td>
</tr>
<tr>
<td></td>
<td>slow</td>
</tr>
<tr>
<td><strong>DFS</strong></td>
<td>Fast</td>
</tr>
<tr>
<td></td>
<td>Little pruning opportunity</td>
</tr>
<tr>
<td></td>
<td>Memory-hungry</td>
</tr>
</tbody>
</table>

BFS and DFS refer to candidate generation approach, not tree traversal 😊
Issues with the Current FTM Solutions (2)
Issues with the Current FTM Solutions (2)
Would that be acceptable to achieve **hundreds-X speedup** at the expense of losing a couple of percent **accuracy**?

**Contribution of this research:**

- Proposing a memory efficient and fast solution to the frequent subtree mining problem on the Automata Processor
- Achieving 350X and more speed up, when allowing 7.5% false positive subtrees
The Rest of This Talk

• Automata Processor
• FTM Challenges and Opportunities on the AP
• Experimental Evaluation
• Takeaways
The Automata Processor (1)

- The Micron Automata Processor (AP) is a reconfigurable non-von Neumann architecture, which implements non-deterministic finite automata (NFA) with Boolean logic gates and counters in hardware based on DRAM technology.

Row Address (Input Symbol)

Row Access results in **49,152** match & route operations (then Boolean AND with “active” bit-vector)
The Automata Processor (2)

- A massively parallel ‘MISD’ architecture
- 1 Gbps data processing
- Hardware resources on development board
  - State Transition Elements (STE): 1.5M
  - Reporting STEs: 200K
  - Counter Elements: 25K
  - Boolean Elements: 74K
Applications on the AP

• Data mining
  – Frequent itemset mining
  – Sequential pattern mining
• Machine learning
  – Random forest
  – Entity resolution
  – String/tree kernel
• Bioinformatics
  – Motif discovery
  – DNA alignment
• ...

International Conference on Supercomputing 2017
Challenges: Exact FTM on the AP

- The AP supports regular languages
- Tree can be represented using context-free-grammar [Ivn07]
Challenges: Exact FTM on the AP

- The AP supports regular languages.
- Tree can be represented using context-free-grammar [Ivn07].
- The AP can not efficiently implement exact FTM.
Challenges: Exact FTM on the AP

The AP supports regular languages

Tree can be represented using context-free-grammar [Ivn07]

The AP can not efficiently implement exact FTM

Exact solutions (e.g., stack implementation on the AP)

Inefficient
Database dependent
Impractical
Opportunities: Pruning

- Four-stage pruning strategy
  - Subset pruning
  - Intersection pruning
  - Downward pruning
  - Connectivity pruning

- Kernel properties
  - Complementary pruning
  - Avoiding false negatives

**Diagram:**

1. **Generate candidates of k-subtree**
2. **Prune candidates on the AP**
   - If $K < \text{max K} \land K$-subtree not empty
     - **Yes**
       - **Find final frequent subtree on the CPU**
       - **Write results**
     - **No**
       - **Is exact solution needed?**
         - **Yes**
           - **Find final frequent subtree on the CPU**
           - **Write results**
         - **No**
           - **K = K+1**
   - **No**

**Flowchart Symbols:**
- **CPU**
- **AP**
- **Generate candidates of k-subtree**
- **Prune candidates on the AP**
- **Find final frequent subtree on the CPU**
- **Write results**
Background

• Frequent itemset mining (FIM)

<table>
<thead>
<tr>
<th>Trans.</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Coke, Diaper</td>
</tr>
</tbody>
</table>

\[ \text{sup} \{ \{ \text{Diaper, Milk} \} \} = 3 \]

• Sequential pattern mining (SPM)

<table>
<thead>
<tr>
<th>Trans.</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{ { \text{Bread, Milk} }, { \text{Coke} } }</td>
</tr>
<tr>
<td>2</td>
<td>{ { \text{Bread, Milk, Diaper} }, { \text{Beer, Eggs} }, { \text{Diaper} } }</td>
</tr>
<tr>
<td>3</td>
<td>{ { \text{Milk} }, { \text{Diaper} }, { \text{Beer, Coke} } }</td>
</tr>
<tr>
<td>4</td>
<td>{ { \text{Bread, Milk, Diaper} }, { \text{Beer, Diaper} }, { \text{Beer, Coke, Eggs} } }</td>
</tr>
<tr>
<td>5</td>
<td>{ { \text{Bread, Milk} }, { \text{Coke} }, { \text{Diaper} }, { \text{Eggs} } }</td>
</tr>
</tbody>
</table>
Kernel 1: Subset Pruning

- **Main goal**: checks downward closure property

\[
\begin{align*}
C_{5i} &= \{C_{4j}, C_{4k}, C_{4l}\} \\
C_4 &= \{C_{40}, C_{41}, \ldots, C_{4m}\}
\end{align*}
\]

Kernel 2: Intersection Pruning

- **Main goal**: checks if all the subsets of a candidate happens in the same tree

<table>
<thead>
<tr>
<th>CPU implementation</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_{4i}</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C_{4j}</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C_{4k}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C_{4l}</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Minimum support = 50%

C_{5i} = 25%

<table>
<thead>
<tr>
<th>AP implementation</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_{4i}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_{4j}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_{4k}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_{4l}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIM**

\[
\text{itemset} = C_{5i} = \{C_{4i}, C_{4j}, C_{4k}, C_{4l}\}
\]

**Input** = \{C_{4i}, C_{4j}, C_{4k}, C_{4l}, \ldots, C_{4i}, C_{4j}, C_{4k}, \ldots, C_{4i}, C_{4j} C_{4l}, \ldots\}
Kernel 3: Downward Pruning

- **Main goal**: checks if ancestor descendant relationship is met

```
Tree
  A
 /|
B E
C D

Downward representation: ABC, ABD, AE
```

```
Subtree1 = \{AC, AD, AE\}
Tree = \{ABC, ABD, AE\}
```

```
```
Kernel 4: Connectivity Pruning

- **Main goal**: checks if sibling relationship is met

![Connectivity representation: A-1 B C D -2 E -3, A B -1 C -2 D -3](image)

**Subtree1** = \{A B -1 C -2 B D\}

**Tree** = \{A -1 B C D -2 E -3, A B -1 C -2 D -3\}

**Special-case SPM**
- Itemset
- Input transaction
1. Making ARM and SPM automata for each kernel
2. Creating appropriate input stream
3. Configuring the automata for each kernel on the AP and streaming the corresponding input stream
4. Getting the potential frequent subtrees from the AP output after applying the kernels
Performance Evaluation

• Platform
  – CPU: Intel(R) Core™ i7-5820k CPU @ 3.30GHz, Memory: 32GB
  – GPU: Tesla K80, Memory 24GB

• Dataset

<table>
<thead>
<tr>
<th>Name</th>
<th>#Trees</th>
<th>Ave_Node</th>
<th>SD_Node</th>
<th>#Items</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1M</td>
<td>1,000,000</td>
<td>5.5</td>
<td>6.2</td>
<td>500</td>
<td>49.3</td>
</tr>
<tr>
<td>T2M</td>
<td>2,000,000</td>
<td>2.95</td>
<td>3.3</td>
<td>100</td>
<td>60.1</td>
</tr>
<tr>
<td>CSLOGS</td>
<td>59691</td>
<td>12.94</td>
<td>22.47</td>
<td>13361</td>
<td>6.3</td>
</tr>
<tr>
<td>TREEBANK</td>
<td>52581</td>
<td>68.03</td>
<td>32.46</td>
<td>1387266</td>
<td>27.3</td>
</tr>
</tbody>
</table>

Ave_Node = Average number of nodes per tree.
SD_Node = Standard deviation of number of nodes per tree.

• Apples-to-apples comparison
GPU Implementation

• BFS Approach, because:
  – Not be bound by the finite GPU global memory
  – Exposes many ready-to-process candidates and provide parallelism

• FTM-GPU
  – Candidate generation on the CPU
  – Subset pruning on the CPU
  – Enumeration on the GPU
    • Trees in shared memory
    • Candidate in constant memory

• Sorting the input trees
  – Decrease divergence
Algorithmic & Architectural Contributions

AP kernel over CPU kernel speedup:
  Subset = up to 163X
  Intersection: up to 19X
  Downward: up to 3144X
  Connectivity: up to 2635X
Algorithmic & Architectural Contributions

AP kernel over CPU kernel speedup:
- Subset = up to 163X
- Intersection: up to 19X
- Downward: up to 3144X
- Connectivity: up to 2635X
Algorithmic & Architectural Contributions

AP kernel over CPU kernel speedup:
  Subset = up to 163X
  Intersection: up to 19X
  Downward: up to 3144X
  Connectivity: up to 2635X

Red bar over black bar: up to 1.6X
Black bars: up to 215X
Red bars: up to 353X
Pruning Efficiency

Kernel effectiveness:
- Subset = 80%
- Intersection: 0.5%
- Downward: 3.5%
- Connectivity: 4.8%
Pruning Efficiency

Kernel effectiveness:
- Subset = 80%
- Intersection: 0.5%
- Downward: 3.5%
- Connectivity: 4.8%

Removing intersection kernel:
- AP over PatternMatcher: 353X → 2190X
- Accuracy: 86% → 83%
FTM-AP vs Other FTM Algorithms

Trade-off between *speed* and *accuracy* of the AP solution vs the existing FTM implementation.

Dataset: CSLOGS
FTM-AP vs Other FTM Algorithms

Trade-off between speed and accuracy of the AP solution vs the existing FTM implementation

Dataset: CSLOGS
FTM-AP vs Other FTM Algorithms

Trade-off between *speed* and *accuracy* of the AP solution vs the existing FTM implementation

Dataset: CSLOGS
FTM-AP vs Other FTM Algorithms

Trade-off between speed and accuracy of the AP solution vs the existing FTM implementation

**Speedup**

\[
\frac{\text{FTM\_AP}}{\text{PatternMatcher}} = \text{up to 353X}
\]

**Memory usage**

\[
\frac{\text{TreeMinerD}}{\text{FTM\_AP}} = \text{up to 5000X}
\]

Dataset: CSLOGS
Exact Solution: AP + TreeMinerD
Exact Solution: AP + TreeMinerD
Performance Evaluation: Exact Solution: AP + TreeMinerD

Dataset: TREEBANK

Intel Xeon CPU, 2.30GHz, 512 GB memory, 2.133GHz
Summary

• Propose a multi-stage pruning framework on the AP
  – The first work to use the AP as a pruning media
  – Novel pruning kernels
  – A better scalability and stable behavior
  – Propose an exact solution

Takeaways

• Rethinking the algorithm when having a new hardware architecture
• Applies to spatial automata computing architecture such as FPGAs
• This approach can be adopted for other complex pattern mining problems
• Provides some insight for the architectural changes
References


• Elaheh Sadredini, Reza Rahimi, Ke Wang, and Kevin Skadron. "Frequent Subtree Mining on the Automata Processor: Opportunities and Challenges." ACM International Conference on Supercomputing (ICS), Chicago, June 2017


Thank you 😊

Questions?
Backup Slides
PDA-based Subtree Mining: An Example

String Encoding:

A A – C D – B D A – C – B D – A – B D

Embedded subtree

String Encoding:

A B C – B A – – – D
Input tree: A A – C D – B D A – C – B D – A ——— B D

Deterministic PDA
Input tree: A A – C D – B D A – C – – B D – A – – – – B D
Input tree: A A – C D – B D A – C – B D A – – B D

A

< A, 0 >

*
Input tree: \( A \to C \to D \to B \to D \to A \to C \to B \to D \to A \to B \to D \)
Input tree: A A – C D – B D A – C –– B D – A ––––– B D
Input tree: A A – C D – B D A – C – B D – A – – – – B D
Input tree: A A – C D – B D A – C – – B D – A – – – – – – – B D
Input tree: \( A \rightarrow C \rightarrow D \rightarrow B \rightarrow D \rightarrow A \rightarrow \cdots \rightarrow B \rightarrow D \)
**Input tree:** A A – C D – B D A – C – B D – A – – – – – B D
Input tree: A A – C D – B D A – C — B D – A ——— B D

Diagram: [Diagram of a DFA or NFA with states labeled q0, q1, q2, ..., q10, transitions labeled with symbols, and a red arrow pointing to q10.5]
Input tree: A A – C D – B D A – C – B D – A – – – – B D
Input tree: A A – C D – B D A – C – B D A – C B D
Input tree: A A – C D – B D A – C – B D – A – – – – – – B D
Input tree: A A – C D – B D A – C – B D – A – – – – – B D
Input tree: A A – C D – B D A – C – – B D – A – – – – B D
Input tree: A A – C D – B D A – C –— B D – A –—– B D
Input tree: A A – C D – B D A – C – – B D – A – – – – – B D
Input tree: A A – C D – B D A – C – B D – A – B D
Input tree: A A – C D – B D A – C – B D – A – – – B D
Input tree: A A – C D – B D A – C –– B D – A –– – B D
Input tree: A A C D B D A C B D A
Input tree: A A – C D – B D A – C – – B D – A – – – – – – B D

A, */<A,0>*/ B, */<B,1>*/ C, */<C,2>*/

A, */<A,5>*/

B, */<B,4>*/

C, */<C,*>/

\( q_0,0 \)

\( A, */<A,0>*/ \)

\( B, */<B,1>*/ \)

\( C, */<C,2>*/ \)

\( q_2,2 \)

\( q_3,3 \)

\( q_4,2 \)

\( \lambda \backslash A, */ \lambda \backslash A */ \)

\( -, */ \epsilon \)

\( \lambda \backslash B, */ \lambda \backslash B */ \)

\( -, */ \epsilon \)

\( \lambda \backslash C, */ \lambda \backslash C */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda \backslash D, */ \lambda \backslash D */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)

\( \lambda, */ \lambda */ \)

\( -, */ \epsilon \)
Input tree: A A – C D – B D A – C -- B D – A -- -- B D