Novel Parallel Method for Mining Frequent Patterns on Multi-core Shared Memory Systems

DISCS - 2013

Lan Vu, Gita Alaghband
University of Colorado Denver
1. Introduction
2. Frequent Pattern Mining Methods
3. Our Parallel Approach
4. Performance Evaluation
5. Future Work
Outline

1. Introduction
2. Frequent Pattern Mining Methods
3. Our Parallel Approach
4. Performance Evaluation
5. Future Work
Data-Intensive Applications

Data Warehouse

Data cleaning
Data integration, etc.

Database

Data mining

Knowledge Patterns

Mining Frequent Patterns, Association Rules
Classification
Clustering
Regression
Summarization
Frequent Pattern Mining (FPM)

- Find patterns frequently occurring in a large database
- Help to answer many useful questions
  - Which genes are effected by a new drug?
Applications

Market Analysis  Query Recommendation  Product Recommendation

Text Mining  Image Mining  Graph Mining
Problem Statement

<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>a, b, c</td>
</tr>
<tr>
<td>02</td>
<td>a, c</td>
</tr>
<tr>
<td>03</td>
<td>a, c</td>
</tr>
<tr>
<td>04</td>
<td>a, d</td>
</tr>
</tbody>
</table>

- Database D with N transactions & M items: \( X = \{a, b, c, \ldots\} \)
- \( Y \) (\( Y \nsubseteq X \)) is frequent pattern if its support is larger or equal to a minimum support threshold (\( \text{mins}\uparrow \))
  \[
  \text{Support}(Y) = \text{Probability}(Y) \geq \text{mins}\uparrow
  \]
- Frequent Pattern Mining = find all possible frequent patterns
- Example: \( a, c, ac \) are 3 frequent patterns if \( \text{mins}\uparrow = 50\% \)
Outline

1. Introduction
2. Frequent Pattern Mining Methods
3. Our Parallel Approach
4. Performance Evaluation
5. Future Work
How to mine frequent patterns?

- Find all combinations of items
- Check for their support to determine the frequent patterns
How to mine frequent patterns?

- Find all combinations of items
- Check for their support to determine the frequent patterns

- If $N = 55$ items
  $\Rightarrow$ # of combinations is $2^{55} = 36,028,797,018,963,968$

- What if $N \sim$ millions of items?
  $\Rightarrow$ Brute force approach is infeasible for large database
Popular Approaches

- Apriori
- Eclat
- FP-growth
- Improvements of these methods
## Popular Approaches

<table>
<thead>
<tr>
<th></th>
<th>Apriori</th>
<th>Eclat</th>
<th>FP-growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large I/O</td>
<td>▶️</td>
<td>▶️ Small I/O</td>
<td>▶️ Small I/O</td>
</tr>
<tr>
<td>Rescan data from disk</td>
<td>▶️</td>
<td>▶️ Vertical data format</td>
<td>▶️ Horizontal data format</td>
</tr>
<tr>
<td>Candidate Gen. &amp; Test</td>
<td>▶️</td>
<td>▶️ Candidate Gen. &amp; Test</td>
<td>▶️ No Candidate Generation</td>
</tr>
<tr>
<td>Breath-first</td>
<td>▶️</td>
<td>▶️ Depth first</td>
<td>▶️ Depth first</td>
</tr>
</tbody>
</table>
Mining Frequent Pattern on Large Database

- Large I/O
- Huge Memory Consumption
- Computationally Intensive

The Digital Universe: 50-fold Growth from the Beginning of 2010 to the End of 2020

1. Introduction
2. Frequent Pattern Mining Methods
3. Our Parallel Approach
4. Performance Evaluation
5. Future Work
ShaFEM: A Novel Parallel Method for Mining Frequent Patterns on Multi-core Shared Memory Systems
Motivation

Existing parallel FPM methods

- Do not efficiently perform on both sparse & dense databases

  → Majority are based on Apriori, Eclat and FP-growth

<table>
<thead>
<tr>
<th>Databases</th>
<th>Type</th>
<th>Minsup</th>
<th>Apriori (sec.)</th>
<th>Eclat (sec.)</th>
<th>FP-growth (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>Dense</td>
<td>20%</td>
<td>1924</td>
<td>77</td>
<td>89</td>
</tr>
<tr>
<td>Connect</td>
<td>Dense</td>
<td>30%</td>
<td>522</td>
<td>366</td>
<td>403</td>
</tr>
<tr>
<td>Retail</td>
<td>Sparse</td>
<td>0.003%</td>
<td>18</td>
<td>59</td>
<td>10</td>
</tr>
<tr>
<td>Kosarak</td>
<td>Sparse</td>
<td>0.08%</td>
<td>4332</td>
<td>385</td>
<td>144</td>
</tr>
</tbody>
</table>
Motivation

Existing parallel FPM mining methods

- Under-utilize the benefits of shared memory

Most are developed for distributed-memory systems

A dual socket server
ShaFEM algorithm

New method for mining frequent patterns on shared memory multi-core systems

- **Self-adapt to data characteristics dynamically**
  - Perform well on both sparse and dense databases

- **Efficiently utilize shared memory**
  - Minimize the communication and synchronization need
  - Maximize data independence for scalability
ShaFEM algorithm

Two main stages:

1. **XFP-tree Construction**
   - An extension of FP-tree shared among cores
   - Compact data of all frequent patterns into memory

2. **Mining frequent patterns**
   - Apply **two mining strategies** and dynamically switch between them based on characteristics of data during the execution
Stage 1: XFP-tree construction

1\textsuperscript{st} scan

Data Partition

Compute local count lists

<table>
<thead>
<tr>
<th></th>
<th>P₀₁</th>
<th></th>
<th>P₀₂</th>
<th></th>
<th>P₀₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>Counts</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P₀₁</th>
<th></th>
<th>P₀₂</th>
<th></th>
<th>P₀₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>Counts</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P₀₁</th>
<th></th>
<th>P₀₂</th>
<th></th>
<th>P₀₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>Counts</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Compute the global count list

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>Counts</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Identify the frequent items based on minsup
Stage 1: XFP-tree construction

Data Partition

2^{nd} scan

Construct local FP-trees
Stage 1: XFP-tree construction

Data Partition

Link local FP-trees into XFP-tree
Stage 2: Mining frequent patterns

Data Partition

Link local FP-trees into XFP-tree

Each process mines freq. patterns independently
Stage 2: Mining frequent patterns

Data Partition

Link local FP-trees into XFP-tree

Each process mines freq. patterns independently
Stage 2: Mining frequent patterns

- Dynamic job scheduling for load balancing
- Apply two mining strategies to work efficiently on both sparse and dense databases
- Switching between MineFPtree and MineBitVector for each subset based on its characteristics
Switching between two mining strategies
Switching between two mining strategies

Child FP-tree

XFP-tree
Switching between two mining strategies

Child FP-tree

XFP-tree
Switching between two mining strategies

Child FP-tree

XFP-tree
Switching between two mining strategies

Child FP-tree

XFP-tree

Bit Vector
Switching between two mining strategies

Child FP-tree

XFP-tree

Bit Vector
Switching between two mining strategies

MineFPTree
Switching between two mining strategies

MineBitVector
Compare to the related works

**Related methods:** Build FP-tree & need locks on nodes of the tree

**Our method:** Build XFP-tree & **NOT** require locks

- Minimize the communication and synchronization need
- Maximize data independence for scalability
Compare to the related works

**Related methods:** Apply only **one mining strategy** based on vertical or horizontal data format

**Our method:** Apply **two mining strategies**
Utilize both FP-tree and Bit Vector structures

Adapt better to data characteristics to perform efficiently on both dense and sparse databases.
Outline

1. Introduction
2. Frequent Pattern Mining Methods
3. Our Parallel Approach
4. Performance Evaluation
5. Future Work
### Experimental Setup

**Environment:**
- Altus Machine with dual six-core AMD Opteron 2747 processors (12 cores), 24GB shared memory and 160GB hard drive.
- Linux-based operating system (CentOS 5.8)

**Benchmarks:**
- Parallel Benchmark: ShaFEM & FP-array
- Implementations use C/C++ & OpenMP
## Experimental Setup

### Datasets:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type</th>
<th># of Items</th>
<th>Average Length</th>
<th># of Trans.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>Dense</td>
<td>76</td>
<td>37</td>
<td>3196</td>
</tr>
<tr>
<td>Connect</td>
<td>Dense</td>
<td>129</td>
<td>43</td>
<td>67557</td>
</tr>
<tr>
<td>Accidents</td>
<td>Moderate</td>
<td>468</td>
<td>33.8</td>
<td>340183</td>
</tr>
<tr>
<td>Retail</td>
<td>Sparse</td>
<td>16470</td>
<td>10.3</td>
<td>88126</td>
</tr>
<tr>
<td>Kosarak</td>
<td>Sparse</td>
<td>41271</td>
<td>8.1</td>
<td>990002</td>
</tr>
</tbody>
</table>
Time Comparison

FP-array: an FPM method by Intel for multicore shared-memory systems
ShaFEM: our FPM method

The lower the better
Relative Speedup on 12 cores

![Bar chart showing relative speedup for different applications with FP-array and ShaFEM.]
ShaFEM Speedup

- **Chess** *(dense, minsup=5%)*
  - [Graph showing speedup vs. cores]

- **Connect** *(dense, minsup=15%)*
  - [Graph showing speedup vs. cores]

- **Accidents** *(moderate, minsup=1%)*
  - [Graph showing speedup vs. cores]

- **Retail** *(sparse, minsup=0.001%)*
  - [Graph showing speedup vs. cores]

- **Kosarak** *(sparse, minsup=0.05%)*
  - [Graph showing speedup vs. cores]
Comparison with Sequential Algorithms

The lower the better

Kosarak ($sparse$)

Chess ($dense$)
ShaFEM - a novel parallel method for FPM on shared memory systems

ShaFEM performs well on both sparse and dense databases

Test cases show savings of up to 4.9 days (1 cores) and 12.8 hours (12 cores) of execution time over the compared competing method.
Outline

1. Introduction
2. Frequent Pattern Mining Methods
3. Our Parallel Approach
4. Performance Evaluation
5. Future Work
Future Work

- Integrate ShaFEM into our hybrid model of shared & distributed memory for multi-core cluster
- Combine CPU-GPU for mining frequent patterns
- Develop a high performance framework for FPM

Questions?

Our contact
Lan Vu - lan.vu@ucdenver.edu
Gita Alaghband - gita.alaghband@ucdenver.edu
Our Parallel Distributed System Lab: http://pds.ucdenver.edu

University of Colorado Denver