Frequent Itemset Mining Using Parallel Computing
Outline

• Introduction
• Frequent Itemset Mining
• Apriori Algorithm
• GPU Architecture
• Parallel Apriori
Introduction

• Wal-Mart in 1990s

• Beer and diaper usually bought together

• Sale of beer increased 30%
Introduction

• Applications of Association Rule Mining
  • Supermarket
  • Web Mining
  • Medical analysis
  • Bioinformatics
  • Network analysis
    (e.g., Denial-of-service (DoS))
  • Programming Pattern Finding
Frequent Itemset Mining

<table>
<thead>
<tr>
<th>TID</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>t2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>t3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>t5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

A, D  
A, B, D, E  
B, C  
A, B, C, D, E  
B, C, E

Single Items (or simply items): A, B, C, D, E

Itemsets:  
\{B, C\}  \{A, B, C\}  \{B, C, D\}  \{A\}

2-itemset  3-itemset  3-itemset  1-itemset
Frequent Itemset Mining

<table>
<thead>
<tr>
<th>TID</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>t2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>t3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>t5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Single Items (or simply items): A, B, C, D, E

Itemsets: {B, C}, {A, B}, {A, B, C}, {A, B, D}, {A}

Frequent itemsets: itemsets with support >= a threshold (e.g., 3)
- Support = 3: e.g., {A}, {B}, {B, C}
- Support = 4: but NOT {A, B, C}
- 3-frequent itemset of size 2
Suppose we want to find all “large” itemsets (e.g., itemsets with support $\geq 3$).

<table>
<thead>
<tr>
<th>TID</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>t2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>t3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>t5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- $\{B, C, E\}$ is NOT large
  - Support of $\{B, C, E\} = 2$

Is $\{A, B, C, E\}$ large?

Is $\{B, C, D, E\}$ large?

**Downward Closure Property:** If an itemset $S$ is NOT large, then any proper superset of $S$ must NOT be large.
Suppose we want to find all “large” itemsets (e.g., itemsets with support $\geq 3$)

Thus, $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$ and $\{E\}$ are “large” itemsets of size 1 (or, “large” 1-itemsets).

We set $L_1 = \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}\}$
Apriori Algorithm

Suppose we want to find all “large” itemsets with support $\geq 3$.

<table>
<thead>
<tr>
<th>TID</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>t2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>t3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>t4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>t5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Thus, $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$ and $\{E\}$ are “large” itemsets of size 1 (or, “large” 1-itemsets).

We set $L_1 = \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}\}$
Suppose we know that itemset \{B, C\} and itemset \{B, E\} are large (i.e., L_2).

It is possible that itemset \{B, C, E\} is also large (i.e., C_3).

**Downward Closure Property:** If an itemset S is NOT large, then any proper superset of S must NOT be large.
Suppose we know that itemset \{B, C\} and itemset \{B, E\} are large (i.e., \(L_2\)). It is possible that itemset \{B, C, E\} is also large (i.e., \(C_3\)).

**Suppose we know that \{C, E\} is not large. We can prune \{B, C, E\} in \(C_3\).**
Apriori

Suppose we want to find all “large” itemsets (e.g., itemsets with support $\geq 3$)

Thus, $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$ and $\{E\}$ are “large” itemsets of size 1 (or, “large” 1-itemsets).

We set $L_1 = \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}\}$
GPU Architecture

GPU

Multiprocessor 1

Multiprocessor 2

Shared Memory

P_1 P_2 \ldots P_n

P_1 P_2 \ldots P_n

Device Memory

CPU

Main Memory

Main Memory
Multithreaded CUDA Program

- A GPU is build around an array of Streaming Multiprocessors (SMs)

- A multithreaded program is partitioned into blocks of threads

- All threads execute independently from each other
Parallel Apriori

• Trei-Based implementation

• Pure bitmap implementation
Trie-Based Candidate Generation

- Trie is a rooted, directed prefix tree
- Each node stores an item ID and its support
- Root – at depth 0
- Node at depth K concatenating all its ancestors represents a K-itemset
- Trie data structure implemented on CPU
**Trie-Based Candidate Generation**

- Incrementally construct the trie level by level
- Perform join for every node a depth K-1 with its right siblings
- Sorted by lexicographical order on item ID
- Prune stage: check whether (K-1)-subset is frequent by performing binary search
- Transform candidate itemsets bookkeeping data to GPU to perform support counting
Bitmap-Based Candidate Generation

- Processed by GPU
- Data stored on GPU device memory
- Using bitmap represent itemset
- Using bitwise “OR” to generate candidate
- Check whether (K-1)-subsets are frequent in prune step
Bitmap-Based Candidate Generation
Bitmap-Based Candidate Generation

• Uniform and efficient bitwise operations to perform joins on GPU

• Avoids the overhead of frequent data transfer between GPU memory and CPU memory

• Incurs excessive non-coalesced device memory access when the number of items is large
Bitmap Representation of Dataset

**Horizontal Data Layout**

<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>Item IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABCD</td>
</tr>
<tr>
<td>2</td>
<td>ABD</td>
</tr>
<tr>
<td>3</td>
<td>ACD</td>
</tr>
<tr>
<td>4</td>
<td>BCD</td>
</tr>
</tbody>
</table>

**Vertical Data Layout**

<table>
<thead>
<tr>
<th>Itemset ID</th>
<th>Transaction IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABD</td>
<td>1,2</td>
</tr>
<tr>
<td>ACD</td>
<td>1,3</td>
</tr>
<tr>
<td>BCD</td>
<td>1,4</td>
</tr>
</tbody>
</table>

**Bitmap Representation**

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABD</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ACD</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BCD</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Support Counting

• Refer to transaction bitmap
• Use build in vector data type int4
• int4 – a structure containing 4x32-bit integer
• GPU read 16byte data from device memory to shared memory in a single instruction
• Construct a lookup table mapping a integer and the number of 1’s
• Put lookup table in constant memory of GPU
Support Counting

• Each GPU Thread Block process on candidate K-itemset in parallel

• K-itemset generated by intersect 2 (K-1)-itemset

• Count added up using parallel reduce

• Assume data type int is of size 8-bits
Parallel Reduce

Values (shared memory)

Step 1
Stride 8
Thread IDs
Values

Step 2
Stride 4
Thread IDs
Values

Step 3
Stride 2
Thread IDs
Values

Step 4
Stride 1
Thread IDs
Values
Experiment Result

Experiment on synthetic dataset **T40I10D100K**

Experiment on real dataset **Retail**
Experiment Result

Experiment on synthetic dataset **T40I10D100K**

Experiment on real dataset **Retail**
Summary

• Using bitmap is inspiring

• Faster than sequential Apriori

• Two different implementation superior in different cases

• Candidate generation is still time consuming in both implementation