A Parallel Pattern Mining Algorithm for Multi-Core Architectures

Benjamin Negrevergne

Jury:

- Marie-Christine Rousset
  LIG/Grenoble University
- Alexandre Termier
  LIG/Grenoble University
- Jean-François Méhaut
  LIG/Grenoble University
- Hiroki Arimura
  Graduate School of Science and Technology/Hokkaido University
- Bruno Crémilleux
  GREYC/University of Caen
- Anne Laurent
  LIRMM/University of Montpellier
Pattern mining is a sub-topic of Data Mining

Knowledge discovery in databases [Fayyad, 96]

"Identifying valid, novel, potentially useful, and ultimately understandable patterns in data"

Pattern mining:
- Extract knowledge as patterns representing regularities (or irregularities) in data.
Broad range of applications

- Mining frequent set of items purchased together
- Mining frequent sub molecules in a molecular database
- Mining correlations in sensors data
Broad range of applications

- Mining frequent set of items purchased together
  - Possible frequent pattern: \{milk, cereals\}
  - Dataset: set of receipts
  - Patterns: set of items

- Mining frequent sub molecules in a molecular database

- Mining correlations in sensors data
Broad range of applications

- Mining frequent set of items purchased together
  - 
- Mining frequent sub molecules in a molecular database
  - 
- Mining correlations in sensors data
  - 
Broad range of applications

- Mining frequent set of items purchased together

- Mining frequent sub molecules in a molecular database

Krammer [2001] was able to identify the following molecular pattern in a set of anti-HIV molecules

- Dataset: set of molecules (represented as graphs)
- Patterns: graphs
Broad range of applications

- Mining frequent set of items purchased together
- Mining frequent sub molecules in a molecular database
- Mining correlations in sensors data
Broad range of applications

- Mining frequent set of items purchased together
  - 
  - Mining frequent sub molecules in a molecular database
  - 
- Mining correlations in sensors data
  - Records from meteorological sensors
  - Frequent pattern:
    - Temperature ↓ Pressure ↓ Windspeed ↑
      - Dataset: list of numerical sensor records
      - Patterns: set of variations
Broad range of applications

- Mining frequent set of items purchased together
- Mining frequent sub molecules in a molecular database
- Mining correlations in sensors data
The broad range of solutions

Nowadays: many different algorithms for each pattern mining problem

- **Frequent itemset mining**
  - Apriori  Eclat  FP-Growth  DCI-Closed  LCM
    - [Agrawal 94]  [Zaki 1997]  [Han 2004]  [Lucchese 2004]  [Uno 2004]

- **Graph Mining**
  - AGM  gSpan  FSG  CloseGraph  Gaston
    - [Inokuchi 2000]  [Yan 2002]  [Kuramochi 2001]  [Yan 2003]  [Nijssen 2004]

- **Tree Mining**
  - FreqT  TreeMiner  Dryade  CMTreeMiner
    - [Asai 2002]  [Zaki 2005]  [Termier 2004]  [Chi 2004]

- **Gradual itemset mining**
  - Grite  PGP-mc  PGLCM
    - [Di Jorio 2009]  [Laurent 2010]  [Do 2010]

- **Sequence**
  - Spade  PrefixSpan  CloSpan  LCM_seq
    - [Zaki 2000]  [Pei 2000]  [Yan 2003]  [Uno]
This algorithmic diversity

1. data owners refrain using pattern mining techniques
2. use inadequate algorithms

**Challenge #1:**
Use a *generic* approach to address different pattern mining problems with a unique algorithm.
Challenge #2: Build an efficient and generic pattern mining algorithm

Pattern mining is inherently **combinatorial**:
- very large number of possible patterns

**Example**

basket market analysis: 1000 items $\rightarrow 2^{1000} (\sim 10^{300})$ possible patterns.

To be efficient algorithms exploit problem properties to reduce the search space ◆ efficient but non-generic

**Challenge #2**

Design an efficient generic and pattern mining algorithm
Contributions of this thesis

We proposed ParaMiner: both *generic* and *efficient* algorithm

How:

- Extend theoretical work on pattern enumeration [Boley 2007] and [Arimura 2009]
- Tackle large real world datasets through *dataset reduction*
- Exploit parallelism for multi-core architectures
Contributions of this thesis

We proposed ParaMiner: both *generic* and *efficient* algorithm

How:

- Extend theoretical work on pattern enumeration [Boley 2007] and [Arimura 2009]
- Tackle large real world datasets through *dataset reduction*
- Exploit parallelism for multi-core architectures

Results

ParaMiner:

- solves various pattern mining problems
- is time-efficient (compete with ad-hoc algorithms)
Outline

Generic framework and problem statement

ParaMiner
   Efficient exploration of the set of candidate patterns
   Speeding up candidate pattern testing
   Parallel execution of ParaMiner

Experiments
   Parallel performance evaluation
   Comparative experiments

Conclusion and future work
Illustration: frequent itemset mining

Minimum frequency threshold = 50%

- $P_1 = \{\text{milk}\}$ is frequent (100%): $t_1, t_2, t_3 \rightarrow \text{closed}$
- $P_2 = \{\text{cereals}\}$ is frequent (66%): $t_1, t_2$
- $P_3 = \{\text{milk}, \text{cereals}\}$ is frequent (66%): $t_1, t_2 \rightarrow \text{closed}$

Receipt 1
- milk
- butter
- cereals

Receipt 2
- milk
- eggs
- cereals

Receipt 3
- milk
- beer
- bread
Illustration: frequent itemset mining

transactions: $t_1$

items
- milk
- butter
- cereals

receipt 1

$P_1 = \{\text{milk}\}$ is frequent (100%): $t_1, t_2, t_3$ → closed

$P_2 = \{\text{cereals}\}$ is frequent (66%): $t_1, t_2$

$P_3 = \{\text{milk}, \text{cereals}\}$ is frequent (66%): $t_1, t_2$ → closed

transactions: $t_2$

receipt 2

milk
eggs
cereals

transactions: $t_3$

receipt 3

milk
beer
bread

minimum frequency threshold = 50%
transactions: \( t_1 \)

- receipt 1
  - milk
  - butter
  - cereals

transactions: \( t_2 \)

- receipt 2
  - milk
  - eggs
  - cereals

transactions: \( t_3 \)

- receipt 3
  - milk
  - beer
  - bread

minimum frequency threshold = 50%

- \( P_1 = \{\textit{milk}\} \) is frequent (100%): \( t_1, t_2, t_3 \)
Minimum frequency threshold = 50%

- $P_1=\{\text{milk}\}$ is frequent (100%): $t_1, t_2, t_3$
- $P_2=\{\text{cereals}\}$ is frequent (66%): $t_1, t_2$
Illustration: frequent itemset mining

transactions: \( t_1 \)

items

\begin{itemize}
  \item \( P_1 = \{ \textit{milk} \} \) is frequent (100\%): \( t_1, t_2, t_3 \)
  \item \( P_2 = \{ \textit{cereals} \} \) is frequent (66\%): \( t_1, t_2 \)
  \item \( P_3 = \{ \textit{milk, cereals} \} \) is frequent (66\%): \( t_1, t_2 \)
\end{itemize}

minimum frequency threshold = 50\%
Illustration: frequent itemset mining

Minimum frequency threshold = 50%

- \( P_1 = \{\text{milk}\} \) is frequent (100%): \( t_1, t_2, t_3 \rightarrow \text{closed} \)
- \( P_2 = \{\text{cereals}\} \) is frequent (66%): \( t_1, t_2 \)
- \( P_3 = \{\text{milk, cereals}\} \) is frequent (66%): \( t_1, t_2 \rightarrow \text{closed} \)

Closed pattern: maximal pattern in a set of transactions
Generic framework

- Every pattern represented as a set

- A pattern mining problem defined
  - A ground set $E$
  - A dataset $D_E$
  - A selection criterion $Select$
Ground set

**Definition**

- Set of all possible elements
- Every candidate pattern is a *subset* of the ground set
**Ground set**

■ **Definition**

- Set of all possible elements
- Every candidate pattern is a *subset* of the ground set

■ **Examples**

**Frequent itemsets** ➤ *E*: all possible items

\[ E = \{ \text{milk, cereals, eggs, . . .} \} \]

e.g. candidate: \{milk, eggs\}

**Connected relational graphs** ➤ *E*: all the possible arcs

\[ E = \{ (G_1, G_2), (G_1, G_3), \ldots, (G_5, G_4) \} \]

eg. candidate:

![Connected relational graphs example]

**Gradual itemsets** ➤ *E*: all the possible variations

\[ E = \{ T \uparrow, T \downarrow, P \uparrow, \ldots, W \downarrow \} \]

eg. candidate: \{T \uparrow, W \uparrow\}
Definition

- Sequence of transactions $\mathcal{D}_E$
- Each transaction is a subset of $E$
Definition

- Sequence of transactions $\mathcal{D}_E$
- Each transaction is a subset of $E$

Examples

Frequent itemsets in $\mathcal{D}_E$ each transaction = receipt
### Definition
- Sequence of *transactions* $D_E$
- Each *transaction* is a subset of $E$

### Examples
- **Frequent itemsets**  
  - in $D_E$ each transaction = receipt
- **Relational graphs**  
  - in $D_E$ each transaction = input graph

\[
\begin{align*}
\left[ G_5, G_6, G_7, G_8 \right] & \Rightarrow \left\{ (G_1, G_2), (G_3, G_1), (G_3, G_2), (G_4, G_1) \right\}, \\
\left[ (G_1, G_2), (G_3, G_2), (G_4, G_1) \right] & \Rightarrow \left\{ (G_1, G_2), (G_3, G_2), (G_4, G_1) \right\}
\end{align*}
\]
**Gradual itemsets** in $\mathcal{D}_E$ each transaction $=$ variation of attributes between pairs of records

<table>
<thead>
<tr>
<th>Date</th>
<th>T</th>
<th>P</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day 1</td>
<td>17.6</td>
<td>1021.20</td>
<td>57</td>
</tr>
<tr>
<td>Day 2</td>
<td>18.5</td>
<td>1021.30</td>
<td>56</td>
</tr>
<tr>
<td>Day 3</td>
<td>20.4</td>
<td>1018.20</td>
<td>51</td>
</tr>
</tbody>
</table>

- $(d_1, d_2)$ \{ $T \uparrow$, $P \uparrow$, $W \downarrow$ \}
- $(d_1, d_3)$ \{ $T \uparrow$, $P \downarrow$, $W \downarrow$ \}
- $(d_2, d_1)$ \{ $T \downarrow$, $P \downarrow$, $W \uparrow$ \}
- $(d_2, d_3)$ \{ $T \uparrow$, $P \downarrow$, $W \downarrow$ \}
- $(d_3, d_1)$ \{ $T \downarrow$, $P \uparrow$, $W \uparrow$ \}
- $(d_3, d_2)$ \{ $T \downarrow$, $P \uparrow$, $W \uparrow$ \}
Definition

- User-defined predicate $2^E \rightarrow \{true, false\}$
- $Select(P, \mathcal{D}_E) \equiv P$ is a **pattern** of interest in $\mathcal{D}_E$
Selection criterion

■ Definition

- User-defined predicate $2^E \rightarrow \{\text{true, false}\}$
- $\text{Select}(P, \mathcal{D}_E) \equiv P$ is a pattern of interest in $\mathcal{D}_E$

■ Examples

Frequent itemsets $\Rightarrow \text{Select}(P, \mathcal{D}_E) \equiv P$ is frequent in $\mathcal{D}_E$

Connected relational graphs $\Rightarrow \text{Select}(P, \mathcal{D}_E) \equiv P$ is a connected graph and $P$ is frequent in $\mathcal{D}_E$

Gradual itemsets $\Rightarrow \text{Select}(P, \mathcal{D}_E) \equiv$ there exists a path $[(d_{x_1}, d_{x_2}), \ldots, (d_{x_{n-1}}, d_{x_n})]$ with $n \geq \text{minsup}$
**Closed patterns**

Closed pattern: Any subset $P$ of $E$ such that

- $P$ occurs in $D_E$ ($D_E[P] \neq \emptyset$)
- $P$ satisfies the selection criterion
- $P$ is the maximal pattern in $D_E[P]

**Problem statement**

Given a **ground set** $E$, a **dataset** $D_E$ and a **selection criterion**

- extract all the **closed patterns**
Outline

Generic framework and problem statement

ParaMiner
   Efficient exploration of the set of candidate patterns
   Speeding up candidate pattern testing
   Parallel execution of ParaMiner

Experiments
   Parallel performance evaluation
   Comparative experiments

Conclusion and future work
The standard approach

generate and test.

1. **Generate** candidate patterns
2. **Test** if the candidate pattern is a pattern
The generate and test principle

The standard approach

*generate and test.*

1. **Generate** candidate patterns
2. **Test** if the candidate pattern is a pattern

... is not naively applicable

- **Too many** candidate patterns
- **Each test** requires costly database accesses
  (e.g. to count frequency)

Challenges

- Generic strategy to explore the set of candidate patterns
- Generic method to simplify the testing
Generic framework and problem statement

ParaMiner
   Efficient exploration of the set of candidate patterns
   Speeding up candidate pattern testing
   Parallel execution of ParaMiner

Experiments
   Parallel performance evaluation
   Comparative experiments

Conclusion and future work
Efficient exploration of the set of candidate patterns

- **Structured search space**

Most algorithms define a *pattern augmentation relation* ▶ search space is explored by repeatedly augmenting patterns

ParaMiner’s augmentation relation

\( P \) and \( Q \), two patterns:

- \( Q \) is the augmentation of \( P \) ⇔ \( Q = P \cup \{ e \} \)

\[
\begin{array}{c}
\text{ACE} \rightarrow \text{ABCE}
\end{array}
\]

**Set of patterns + Augmentation relation = DAG\(^1\) structure**

\(^1\) *directed acyclic graph*
Structured search space

DAG-structure of a set of patterns

▶ several arcs lead to the same pattern
An enumeration strategy

Is required to:
- discover all the patterns
- avoid duplicates generation

Which augmentation path must be followed?
An enumeration strategy

Is required to:

- discover all the patterns
- avoid duplicates generation

Which augmentation path must be followed?

▶ AB is the first parent of ABC
An enumeration strategy

Is required to:

- discover all the patterns
- avoid duplicates generation

Which augmentation path must be followed?

- BC is the first parent of ABC
Enumeration strategy

Follow the augmentation $P \rightarrow Q$ if:

$\text{first_parent}(Q) = P$

**DAG-structure following a tree search**
Enumeration strategy

Follow the augmentation $\text{P} \rightarrow \text{Q}$ if:

$\text{first_parent}(\text{Q}) = \text{P}$

DAG-structure following a tree search
How to compute the first parent of a pattern

- **Requirement**

A method that is:

- **Generic**
  - Sound for all our pattern mining problems

- **Adequate to parallel exploration of the search space**
  - Computable without *global computations*
How to compute the first parent of a pattern

■ Requirement
A method that is:

- **Generic**
  - Sound for all our pattern mining problems
- **Adequate to parallel exploration of the search space**
  - Computable without *global computations*

■ State of the art:
Theoretical work closed pattern enumeration strategies

- [Arimura et al. 2009]
- [Boley et al. 2010]
  - Poly-space enumeration strategies
  - first_parent do not rely on a global representation of the search space
[Boley 2010] **Strong accessibility:**

**Idea:** *We can reach every pattern by augmenting any subset that is a pattern*
[Boley 2010] **Strong accessibility:**

**Idea:** *We can reach every pattern by augmenting any subset that is a pattern*

We can detect first parent by memorizing **only the root** of the forking branches:

- Branch can be explored independently → Parallel
- FIM, CRG and GRI satisfy this property (proved) → Generic
Conclusion on pattern enumeration

• Pattern augmentation
• Structured search space
• Enumeration strategy required
• Problem of detecting the first parent
• Strong accessibility property
• Exclusion list

► generic and parallelizable enumeration strategy

Next: Identifying patterns
Outline

Generic framework and problem statement

ParaMiner
  Efficient exploration of the set of candidate patterns
  Speeding up candidate pattern testing
  Parallel execution of ParaMiner

Experiments
  Parallel performance evaluation
  Comparative experiments

Conclusion and future work
Candidate pattern testing

- Check candidate pattern occurrence in the dataset
- Computing the selection criterion
- Computing pattern closure

Intensive access to the dataset

**Problem:** How to reduce dataset accesses?
- dataset reduction [Han 2000, Uno 2004]
**Motivation:** Every element in the dataset are not required to test each candidate pattern

**Principle:**
1. build a dataset **for each** new pattern $P$
2. **filter** unnecessary elements to $P$ and its descendants
3. use it to test $P$ and its descendants

**In ParaMiner:** EL-based filter $\rightarrow$ generic
EL-based reduction (Proved)

**Dataset:**
\( \mathcal{D}_E \)

**Pattern:**
\( P = \{D, E\} \)

**EL:**
\( EL = \{A, B, C\} \)

\( \mathcal{D}_P \) reduced ?

<table>
<thead>
<tr>
<th>( \mathcal{D}_E )</th>
<th>A, B, D, E, F, G, D, E, F, G, A, B, D, E, F, H,</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>A, B, D, E, F, G,</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>A, B, D, E, F, G,</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>A, B, D, E, F, H,</td>
</tr>
</tbody>
</table>
EL-based reduction (Proved)

Dataset:
\( D_E \)

Pattern:
\( P = \{D, E\} \)

EL:
\( EL = \{A, B, C\} \)

\( ▶ D_P^{reduced} ? \)

<table>
<thead>
<tr>
<th>( D_E )</th>
<th>( \in EL )</th>
<th>( P )</th>
<th>augm. (( \notin EL ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>A, B,</td>
<td>D, E,</td>
<td>F, G,</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>A, D,</td>
<td>D, E,</td>
<td>F, G,</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>A, B,</td>
<td>D, E,</td>
<td>F, H,</td>
</tr>
</tbody>
</table>

- Element required: Any \( e \) that can belongs to a closed pattern including \( P \)  
  ▶ ensure that first parent is sound
EL-based reduction (Proved)

**Dataset:**
\( \mathcal{D}_E \)

**Pattern:**
\( P = \{D, E\} \)

**EL:**
\( EL = \{A, B, C\} \)

\[ D \] reduced \( P \)?

<table>
<thead>
<tr>
<th>( \mathcal{D}_E )</th>
<th>( \in EL )</th>
<th>( P )</th>
<th>Augm. ( (\notin EL) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>A, B,</td>
<td>D, E,</td>
<td>F, G,</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>A, D,</td>
<td>D, E,</td>
<td>F, G,</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>A, B,</td>
<td>D, E,</td>
<td>F, H,</td>
</tr>
</tbody>
</table>

- Element required: Any \( e \) that can belong to a closed pattern including \( P \) ➤ ensure that first parent is sound
EL-based reduction (Proved)

**Dataset:**
\[ \mathcal{D}_E \]

**Pattern:**
\[ P = \{D, E\} \]

**EL:**
\[ EL = \{A, B, C\} \]

\[ \mathcal{D}_P^{\text{reduced}} ? \]

- Element required: Any \( e \) that can belongs to a closed pattern including \( P \)  
  \( \xrightarrow{\text{ensure that first parent is sound}} \)

**How to detect them:**
- grouping transactions per augmentations supported
- Apply reduction
EL-based reduction (Proved)

Dataset:
\( \mathcal{D}_E \)

Pattern:
\( P = \{D, E\} \)

EL:
\( EL = \{A, B, C\} \)

\[ \begin{array}{c|c|c|c|c}
\mathcal{D}_E & \in EL & P & \text{augm.} & (\notin EL) \\
\hline
D & A, B, & D, E, & F, G, & \\
E & A, D, & D, E, & F, G, & \{P_1\} \\
\hline
\mathcal{D}_P \text{ reduced} & \cap & = & = & \\
\hline
D' & A, D, E, F, G, & \\
E' & A, B, D, E, F, H, & \\
\end{array} \]

\( \mathcal{D}_P \text{ reduced} \) ?

- Element required: Any \( e \) that can belong to a closed pattern including \( P \) ensure that first parent is sound

How to detect them:
- grouping transactions per augmentations supported
- Apply reduction
Evaluation of dataset reduction

**Metric**

\[
\text{reduction\_factor} = \frac{|D_E|}{|D_{P\text{\_reduced}}|}
\]

\[
\text{average\_reduction\_factor} = \frac{\sum_{P \in C} |D_E| / |D_{P\text{\_reduced}}|}{|C|}
\]

**Frequent itemset mining, mushroom dataset**

<table>
<thead>
<tr>
<th>dataset name</th>
<th>ground set size</th>
<th># transactions</th>
<th>dataset size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mushroom</td>
<td>119</td>
<td>8,124</td>
<td>186,852</td>
</tr>
</tbody>
</table>
Evaluation of dataset reduction

**Metric**

\[
\text{reduction\_factor} = \frac{|D_E|}{|D_{\text{reduced}}|}
\]

\[
\text{average\_reduction\_factor} = \frac{\sum_{P \in C} |D_E|}{|D_{\text{reduced}}|} / \frac{|C|}{|C|}
\]

---

**Problem:** FIM, dataset: Mushroom

---

**PARAMINER**

**NODSR**

---

Laboratoire d'Informatique de Grenoble
ParaMiner: algorithm

1: procedure expand\((P, \mathcal{D}_P^{\text{reduced}}, EL)\)
2: for all \(e\) such that \(e\) occurs in \(\mathcal{D}_P^{\text{reduced}}\) do
3: \(\text{if } \text{Select}(P \cup \{e\}, \mathcal{D}_P^{\text{reduced}}) \text{ then}\)
4: \(Q \leftarrow \text{Clo}(P \cup \{e\}, \mathcal{D}_P^{\text{reduced}})\)
5: \(\text{if } \text{is\_first\_parent}(P, EL, Q) \text{ then}\)
6: \(\text{output } Q\)
7: \(\mathcal{D}_Q^{\text{reduced}} \leftarrow \text{reduce}(\mathcal{D}_P^{\text{reduced}}, e, EL)\)
8: \(\text{spawn expand}(Q, \mathcal{D}_Q^{\text{reduced}}, EL)\)
9: \(EL \leftarrow EL \cup \{e\}\)
10: \(\text{end if}\)
11: \(\text{end if}\)
12: \(\text{end for}\)
13: end procedure
ParaMiner: main achievements

• **Proved sound and complete** under some constraints:
  ▶ the set of patterns satisfying the **strong accessibility** property
  ▶ the selection criterion is **decomposable**

Problems FIM, CRG and GRI satisfy these properties

• **Parallelized**: space exploration is divided into independent tasks
  ▶ non-trivial for **closed** pattern mining
ParaMiner: main achievements

- **Proved sound and complete** under some constraints:
  - the set of patterns satisfying the **strong accessibility** property
  - the selection criterion is **decomposable**

Problems FIM, CRG and GRI satisfy these properties

- **Parallelized**: space exploration is divided into independent tasks
  - non-trivial for **closed** pattern mining

- Melinda a parallel execution engine for **data-driven** algorithms
  - can be extended with task scheduling strategies
  - execute tasks spawned by ParaMiner
  - also used in PLCM [Negrevergne 2010], PGLCM [Do 2010]
Generic framework and problem statement

ParaMiner
- Efficient exploration of the set of candidate patterns
- Speeding up candidate pattern testing
- Parallel execution of ParaMiner

Experiments
- Parallel performance evaluation
- Comparative experiments

Conclusion and future work
Melinda: Main concepts

- **Tuples** \((A, B, C)\) ➤ block of data with a fixed structure
- **Tuple Space** ➤ synchronized memory space
  - can contain tuples
  - supports `put()` and `get()`

Parallelizing ParaMiner with Melinda

ParaMiner parallelization scheme: A new tuple for each recursive call

Tuple structure: Parameters of recursive calls

\((Pattern, Reduce\_dataset, Exclusion\_list)\)
Melinda: Main concepts

- **Tuples** \((A, B, C)\) ➤ block of data with a fixed structure
- **Tuple Space** ➤ synchronized memory space
  - can contain tuples
  - supports `put()` and `get()`

Parallelizing ParaMiner with Melinda

ParaMiner parallelization scheme: A new tuple for each recursive call

Tuple structure: Parameters of recursive calls
\((Pattern, Reduce\_dataset, Exclusion\_list)\)

- Tuples are **produced** when a pattern is discovered
- Tuples are **consumed** when a core is idle
Parallel execution

Tuple Space

Core #1
Core #2
Core #3
Parallel execution
Parallel execution

Tuple Space

Laboratoire d'Informatique de Grenoble
Parallel execution
Parallel execution

Tuple Space

Core #1

Core #2

Core #3
Parallel execution

Tuple Space

Core #1

Core #2

Core #3
Parallel execution

Tuple Space

Core #1

Core #2

Core #3

- def
- df
- e
- d
- c
- b
- a
- abe
- abd
- ad
- ab
Parallel execution
Generic framework and problem statement

ParaMiner
  Efficient exploration of the set of candidate patterns
  Speeding up candidate pattern testing
  Parallel execution of ParaMiner

Experiments
  Parallel performance evaluation
  Comparative experiments

Conclusion and future work
Outline

Generic framework and problem statement

ParaMiner
  Efficient exploration of the set of candidate patterns
  Speeding up candidate pattern testing
  Parallel execution of ParaMiner

Experiments
  Parallel performance evaluation
  Comparative experiments

Conclusion and future work
Parallel performance evaluation

- ParaMiner instances
  - frequent itemset mining
  - frequent relational graph mining
  - gradual itemset mining

- Computing platforms
  
  **Laptop:**
  - 4-cores (4 core i7)
  - 8 GB memory

  **Server:**
  - 32-cores (4 core i7 with each core each)
  - 64 GB memory

- Metric
  
  \[
  \text{speedup} = \frac{\text{time using one core}}{\text{time using n cores}}
  \]
1. Laptop

- speedup on 4 cores: 3 to 4
## 2. Server

![ParaMiner Speedups Graph](image)

- Itemsets BMS-WebView 0.02%
- Itemsets Accidents 20%
- Graphs Hughes 65%
- Graduals: C1000-A100 80%
- Graduals: I4408 80%

Graph shows the speedup measurements for different datasets and configurations. The x-axis represents the number of cores, and the y-axis represents the speedup.
In [Buerhrer 2006, Tatikonda 2008], authors exhibit two important issues:

- load imbalance
- memory issues
In [Buerhrer 2006, Tatikonda 2008], authors exhibit two important issues:

- load imbalance
- memory issues

**Bus contention**

Memory bus: connect cores to memory
In [Buerhrer 2006, Tatikonda 2008], authors exhibit two important issues:

- load imbalance
- memory issues

**Bus contention**

Memory bus: connect cores to memory

too many memory access ➤ subject to contention
Bus contention in ParaMiner

■ % hi-lat. memory operation VS # of cores

ggradual itemset mining

( good speedup)

frequent itemset mining

( bounded speedup)

dark slices = high latencies memory operations

• **GRI**: more cores ▶ % hi-lat. memory operations remains constant

• **FIM**: more cores ▶ increasing % of hi-lat. memory operations
Melinda’s approach

[Tatikonda 2008] proposed **architecture conscious** pattern mining algorithm

Requires deep modifications in the algorithm

Inadequate to ParaMiner:
▶ could penalize executions that perform well

■ **Melinda’s tuple distribution strategies**

  ● distribute tuples to available cores
  ● user-defined
    ▶ can exploit **algorithmic level** information (in tuples)
    ▶ can exploit **architectural level** information (available in Melinda)
**Structured tuplespace**

Tuple space is divided into **internals**
- Internals can be used to classify tuples in the tuple space

**Defining new strategies**

- `distribute()` defines in which internal to put the tuple
- `retrieve()` from which internal the tuple has to be taken

- supports heavy usage
1. **Distribute**: Tuples sharing datastructures go in the same internal

2. **Retrieve**: Cores of the same processor retrieve tuples from the same internal
1. **Distribute**: Tuples sharing datastructures go in the same internal

2. **Retrieve**: Cores of the same processor retrieve tuples from the same internal

- Improve cache usage and reduce bus contention

- Frequent itemset mining, BMS-WebView/Accidents dataset

<table>
<thead>
<tr>
<th>dataset</th>
<th># items</th>
<th># transactions</th>
<th>dataset size</th>
<th>Density (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMS-WebView-2</td>
<td>3,340</td>
<td>77,512</td>
<td>320,601</td>
<td>0.14</td>
</tr>
<tr>
<td>Accidents</td>
<td>468</td>
<td>340,184</td>
<td>11,500,870</td>
<td>7.22</td>
</tr>
</tbody>
</table>
1. **Distribute**: Tuples sharing datastructures go in the same internal.

2. **Retrieve**: Cores of the same processor retrieve tuples from the same internal.

No source code modification ➤ 30% performance gain
Outline

Generic framework and problem statement

ParaMiner
- Efficient exploration of the set of candidate patterns
- Speeding up candidate pattern testing
- Parallel execution of ParaMiner

Experiments
- Parallel performance evaluation
- Comparative experiments

Conclusion and future work
Comparative experiments: frequent itemset mining

■ Algorithms

- **PLCM** [Negrevergne 2010] parallel implementation of LCM [Uno 2004] FIMI’04 Award
- **MT-Closed** [Lucchese 2007] parallel implementation of DCI-Closed [Lucchese 2004]

■ Datasets

<table>
<thead>
<tr>
<th>dataset</th>
<th># items</th>
<th># transactions</th>
<th>dataset size</th>
<th>Density (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMS-WebView-2</td>
<td>3,340</td>
<td>77,512</td>
<td>320,601</td>
<td>0.14</td>
</tr>
<tr>
<td>Accidents</td>
<td>468</td>
<td>340,184</td>
<td>11,500,870</td>
<td>7.22</td>
</tr>
</tbody>
</table>

\[
density(\mathcal{D}_E) = \frac{\#\text{items} \times \#\text{transactions}}{|\mathcal{D}_E|} (\times100)
\]
Comparative experiments frequent itemset mining (2)

- BMS-WebView-2 (sparse)

![Graph showing time vs. relative frequency threshold for different algorithms.](image-url)
Accidents (dense)

Problem: FIM, dataset: Accidents
ParaMiner 32 cores
PLCM 32 cores
MT-Closed 32 cores
Comparative experiments: gradual itemset mining

- **Algorithms**
  - PGLCM [Do 2010]
  - PGP-mc (non-closed) [Laurent 2010]

- **_datasets**

<table>
<thead>
<tr>
<th>dataset name</th>
<th>ground set size</th>
<th># transactions</th>
<th>dataset size</th>
<th>Density (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I4408</td>
<td>8824</td>
<td>11,556</td>
<td>50,985,072</td>
<td>50</td>
</tr>
</tbody>
</table>
Comparative experiments gradual itemset mining (2)

**I4408 (Real)**

![Graph showing gradual itemset mining for I4408](image)

- **ParaMiner 32 cores**
- **PGLCM**

Relative frequency threshold (%) vs. time (s, logscale)

---

Laboratoire d'Informatique de Grenoble
Generic framework and problem statement

ParaMiner
  Efficient exploration of the set of candidate patterns
  Speeding up candidate pattern testing
  Parallel execution of ParaMiner

Experiments
  Parallel performance evaluation
  Comparative experiments

Conclusion and future work
Conclusion

- **ParaMiner**
  - based on state of the art on pattern enumeration
  - useful to parallel pattern mining
  - designed an efficient generic dataset reduction
  - exploit multi-core architectures

- **Melinda**
  - parallel engine of ParaMiner (and other algorithms)
  - extensible through strategies
Conclusion

- **ParaMiner**
  - based on state of the art on pattern enumeration
  - useful to parallel pattern mining
  - designed an efficient generic dataset reduction
  - exploit multi-core architectures

- **Melinda**
  - parallel engine of ParaMiner (and other algorithms)
  - extensible through strategies

**Novelty**

- ParaMiner: an efficient solution for new pattern mining problems
  New pattern mining problem can benefit from 15 years of research

- ParaMiner/Melinda: tool to learn about parallel pattern mining
Future works

- **Extend ParaMiner’s genericity**
  - Generalize strong accessibility properties to other structures
    - handle sequences, and general graphs
  - Study partial strong accessibility of set of patterns
    - bridge with constraint pattern mining

- **Extend ParaMiner/Melinda’s efficiency**
  - Improve Melinda’s strategies
    - More flexible and expressive querying tuples
  - Target larger computing platforms
    - cluster of computers
Thank you !