Parallel Generation of Transversal Hypergraphs

Yuzhen Xie
Computer Science Department
University of Western Ontario (UWO)
joint work with
Charles E. Leiserson (CSAIL, MIT),
Liyun Li and Marc Moreno Maza (UWO)

SIAM DM, 2012
Hypergraphs

- For a finite set $V$ of vertices, $\mathcal{H} = (V, \mathcal{E})$ is a hypergraph if $\mathcal{E}$ (called hyperedges) is a collection of subsets of $V$.

**Example:** $\mathcal{H} = (123456, \{12, 234, 345, 56\})$.

Note: a hyperedge can have more than two vertices.
Hypergraphs

- For a finite set $V$ of vertices, $\mathcal{H} = (V, \mathcal{E})$ is a hypergraph if $\mathcal{E}$ (called hyperedges) is a collection of subsets of $V$.

**Example**: $\mathcal{H} = (\{1, 2, 3, 4, 5, 6\}, \{\{1, 2\}, \{2, 3, 4\}, \{3, 4, 5\}, \{5, 6\}\})$.

- A subset $T$ of $V$ is a transversal (or hitting set) of $\mathcal{H}$ if it intersects all the hyperedges of $\mathcal{H}$, i.e. $T \cap E \neq \emptyset$, $\forall E \in \mathcal{E}$.

Note: a hyperedge can have more than two vertices.
Hypergraphs

For a finite set $V$ of vertices, $\mathcal{H} = (V, E)$ is a hypergraph if $E$ (called hyperedges) is a collection of subsets of $V$.

**Example:** $\mathcal{H} = (\{1, 2, 3, 4, 5, 6\}, \{\{1, 2\}, \{2, 3, 4\}, \{3, 4, 5\}, \{5, 6\}\})$.

Note: a hyperedge can have more than two vertices.

A subset $T$ of $V$ is a transversal (or hitting set) of $\mathcal{H}$ if it intersects all the hyperedges of $\mathcal{H}$, i.e. $T \cap E \neq \emptyset$, $\forall E \in E$.

A transversal $T$ of $\mathcal{H}$ is minimal if no proper subset of $T$ is a transversal of $\mathcal{H}$. 
Hypergraphs

For a finite set $V$ of vertices, $\mathcal{H} = (V, \mathcal{E})$ is a hypergraph if $\mathcal{E}$ (called hyperedges) is a collection of subsets of $V$.

**Example:** $\mathcal{H} = (\{1, 2, 3, 4, 5, 6\}, \{\{1, 2\}, \{2, 3, 4\}, \{3, 4, 5\}, \{5, 6\}\})$.

Note: a hyperedge can have more than two vertices.

A subset $T$ of $V$ is a transversal (or hitting set) of $\mathcal{H}$ if it intersects all the hyperedges of $\mathcal{H}$, i.e. $T \cap E \neq \emptyset$, $\forall E \in \mathcal{E}$.

A transversal $T$ of $\mathcal{H}$ is minimal if no proper subset of $T$ is a transversal of $\mathcal{H}$.

**Example:** $\{25\}$ is a minimal transversal of $\mathcal{H}$; $\{235\}$ is a transversal but not minimal.
Transversal Hypergraph Generation (THG)

- The transversal hypergraph $\text{Tr}(\mathcal{H})$ is the family of all minimal transversals of $\mathcal{H}$.

**Example:** $\mathcal{H} = (123456, \{12, 234, 345, 56\})$, $\text{Tr}(\mathcal{H}) = (123456, \{135, 136, 145, 146, 236, 246, 25\})$.

Note: the size (number of edges) of $\text{Tr}(\mathcal{H})$ can be exponential in the order of $\mathcal{H}$ (number of vertices).
The transversal hypergraph $\text{Tr}(\mathcal{H})$ is the family of all minimal transversals of $\mathcal{H}$.

**Example:** $\mathcal{H} = (123456, \{12, 234, 345, 56\})$,
$\text{Tr}(\mathcal{H}) = (123456, \{135, 136, 145, 146, 236, 246, 25\})$. 

Note: the size (number of edges) of $\text{Tr}(\mathcal{H})$ can be exponential in the order of $\mathcal{H}$ (number of vertices).

The transversal hypergraph generation problem is to compute $\text{Tr}(\mathcal{H})$, given a hypergraph $\mathcal{H}$. 

![Diagram of a hypergraph with vertices and edges labeled 1 to 6.](image)
Transversal Hypergraph Generation (THG)

- The transversal hypergraph \( \text{Tr}(\mathcal{H}) \) is the family of all minimal transversals of \( \mathcal{H} \).

**Example:** \( \mathcal{H} = (123456, \{12, 234, 345, 56\}) \), 
\( \text{Tr}(\mathcal{H}) = (123456, \{135, 136, 145, 146, 236, 246, 25\}) \).

Note: the size (number of edges) of \( \text{Tr}(\mathcal{H}) \) can be exponential in the order of \( \mathcal{H} \) (number of vertices).

- The transversal hypergraph generation problem is to compute \( \text{Tr}(\mathcal{H}) \), given a hypergraph \( \mathcal{H} \).

- Numerous applications: data mining, computational biology, artificial intelligence and logic, cryptography, semantic web, mobile communication systems, e-commerce, etc.
Berge (1987): for two hypergraphs $\mathcal{H}' = (V, E')$ and $\mathcal{H}'' = (V, E'')$ we have

$$\text{Tr}(\mathcal{H}' \cup \mathcal{H}'') = \text{Min}(\text{Tr}(\mathcal{H}') \lor \text{Tr}(\mathcal{H}'')),$$

where

$$\mathcal{H}' \lor \mathcal{H}'' = (V, \{E' \cup E'' \mid (E', E'') \in E' \times E''\}),$$

and $\text{Min}(\mathcal{H}')$ returns the edges of $\mathcal{H}'$ that are $\subseteq$-minimal.
THG: State-of-the-Art (1/3)

- Berge (1987): for two hypergraphs $\mathcal{H}' = (V, \mathcal{E}')$ and $\mathcal{H}'' = (V, \mathcal{E}'')$ we have

$$\text{Tr}(\mathcal{H}' \cup \mathcal{H}'') = \text{Min}(\text{Tr}(\mathcal{H}') \lor \text{Tr}(\mathcal{H}'')),$$

where

$$\mathcal{H}' \lor \mathcal{H}'' = (V, \{E' \cup E'' \mid (E', E'') \in \mathcal{E}' \times \mathcal{E}'\}),$$

and $\text{Min}(\mathcal{H}')$ returns the edges of $\mathcal{H}'$ that are $\subseteq$-minimal.

This algorithm suggests an incremental approach. More precisely, let $\mathcal{E} = \{E_1, \ldots, E_m\}$ and $\mathcal{H}_i = (V, \{E_1, \ldots, E_i\})$ for $i = 1 \cdots m$. Then,

$$\text{Tr}(\mathcal{H}_{i+1}) = \text{Min}(\text{Tr}(\mathcal{H}_i) \lor (V, \{\{v\} \mid v \in E_{i+1}\})).$$
Dong and Li’s border differential algorithm (DL, 1999-2005):
– reminiscent of Berge’s;
– processes edges 1-by-1, in increasing order of cardinality;
– program performs well with only a few edges of small size.
Dong and Li’s border differential algorithm (DL, 1999-2005):
– reminiscent of Berge’s;
– processes edges 1-by-1, in increasing order of cardinality;
– program performs well with only a few edges of small size.

Bailey, Manoukian and Ramamohanarao (BMR03):
– a divide-n-conquer approach, recursively partitioning the edge set by the frequency of the vertices involved;
– use DL-Algorithm to compute the transversal for small-size hypergraphs; Store intermediate minimal transversals;
– program was 9 to 29 times faster than DL’s.
THG: State-of-the-Art (2/3)

- **Dong and Li**’s border differential algorithm (DL, 1999-2005):
  - reminiscent of Berge’s;
  - processes edges 1-by-1, in increasing order of cardinality;
  - program performs well with only a few edges of small size.

- **Bailey, Manoukian and Ramamohanarao** (BMR03):
  - a divide-n-conquer approach, recursively partitioning the edge set by the frequency of the vertices involved;
  - use DL-Algorithm to compute the transversal for small-size hypergraphs; Store intermediate minimal transversals;
  - program was 9 to 29 times faster than DL’s.

- **Fredman and Khachiyian**’s algorithm (1996), implemented by **Boros, Elbassioni, Gurvich and Khachiyan** (BEGK03):
  - test the duality of a pair of monotone boolean functions;
  - incremental quasi-polynomial time algorithm.
THG: State-of-the-Art (3/3)

- **Kavvadias and Stavropoulos** (KS05):
  - Berge’s algorithm combined with techniques to overcome the potentially exponential memory requirement: generalized and appropriate vertices, depth-first strategy.
  - Program outperformed BEGK and BMR for small to medium size problems, and was competitive for large size problems.
THG: State-of-the-Art (3/3)

- **Kavvadias and Stavropoulos (KS05):**
  - Berge’s algorithm combined with techniques to overcome the potentially exponential memory requirement: generalized and appropriate vertices, depth-first strategy.
  - Program outperformed BEGK and BMR for small to medium size problems, and was competitive for large size problems.

- **Khachiyan et al. (2006):**
  - Theoretical study on global parallelism for hypergraphs of bounded edge size $k$;
  - CREW-PRAM model; $\text{polylog}(|V|, |\mathcal{H}|, k)$ time assuming $\text{poly}(|V|, |\mathcal{H}|, k)$ number of processors.
THG: State-of-the-Art (3/3)

- **Kavvadias and Stavropoulos (KS05):**
  - Berge’s algorithm combined with techniques to overcome the potentially exponential memory requirement: generalized and appropriate vertices, depth-first strategy.
  - Program outperformed BEGK and BMR for small to medium size problems, and was competitive for large size problems.

- **Khachiyan et al. (2006):**
  - Theoretical study on global parallelism for hypergraphs of bounded edge size $k$;
  - CREW-PRAM model; polylog($|V|$, $|H|$, $k$) time assuming poly($|V|$, $|H|$, $k$) number of processors.

- **Lower Bounds:**
  - **Takata** (2007): Berge’s algorithm is not output-polynomial;
  - **Hagen** (2008): None of BMR03, DL05 and KS05 is.
Our Parallel Transversal Algorithm: ParTran

- Apply Berge’s formula in a divide-n-conquer manner where $H'$ and $H''$ are of similar order.

$$\text{Tr}(H' \cup H'') = \text{Min}(\text{Tr}(H') \lor \text{Tr}(H''))$$
Our Parallel Transversal Algorithm: ParTran

- Apply Berge’s formula in a **divide-n-conquer** manner where $\mathcal{H}'$ and $\mathcal{H}''$ are of similar order.

\[
\text{Tr}(\mathcal{H}' \cup \mathcal{H}'') = \text{Min}(\text{Tr}(\mathcal{H}') \lor \text{Tr}(\mathcal{H}''))
\]

- Compute $\mathcal{H}' \lor \mathcal{H}''$ also in a **divide-n-conquer** manner as a Cartesian product traversal, and apply **Min** to intermediate results so as to control expression swell.
Our Parallel Transversal Algorithm: ParTran

- Apply Berge’s formula in a divide-n-conquer manner where $\mathcal{H}'$ and $\mathcal{H}''$ are of similar order.

  $$\text{Tr}(\mathcal{H}' \cup \mathcal{H}'') = \text{Min}(\text{Tr}(\mathcal{H}') \lor \text{Tr}(\mathcal{H}''))$$

- Compute $\mathcal{H}' \lor \mathcal{H}''$ also in a divide-n-conquer manner as a Cartesian product traversal, and apply Min to intermediate results so as to control expression swell.

- Compute Min, again in a divide-n-conquer manner.
Our Parallel Transversal Algorithm: ParTran

- Apply Berge’s formula in a **divide-n-conquer** manner where $\mathcal{H}'$ and $\mathcal{H}''$ are of similar order.

\[
\text{Tr}(\mathcal{H}' \cup \mathcal{H}'') = \min(\text{Tr}(\mathcal{H}') \lor \text{Tr}(\mathcal{H}''))
\]

- Compute $\mathcal{H}' \lor \mathcal{H}''$ also in a **divide-n-conquer** manner as a Cartesian product traversal, and apply $\min$ to intermediate results so as to **control expression swell**.

- Compute $\min$, again in a **divide-n-conquer** manner.

- Parallelism is created by the **divide-n-conquer** recursive calls.
The Core Operation: Min

- We describe a procedure ParMinPoset, in the following, for parallel computation of the minimal elements of a partially ordered set.

- Our computations for $\text{Tr}(\mathcal{H})$ and $\mathcal{H}' \lor \mathcal{H}''$ follow the same scheme.
Partially Ordered Set (POSET)

- \((A, \preceq)\) is a poset if \(\preceq\) is a binary relation on \(A\) which is reflexive, antisymmetric, and transitive.

- \(x \in A\) is **minimal** for \(\preceq\) if for all \(y \in A\) we have: \(y \preceq x \Rightarrow y = x\).

- \(\text{Min}(A, \preceq)\), or simply \(\text{Min}(A)\) designates the set of the minimal elements of \(A\).

- A poset example for the integer divisibility relation:

\[
\begin{array}{ccc}
8 & & 12 \\
4 & \searrow & 6 \\
2 & & 3 \\
9 & \nearrow &
\end{array}
\]
A Simple Procedure but . . .

### Algorithm 1: SerMinPoset

**Input**: a poset $A = \{a_0, \cdots, a_{n-1}\}$

**Output**: $\text{Min}(A)$

```plaintext
for $i$ from 0 to $n-2$ do
    if $a_i$ is not marked then
        for $j$ from $i+1$ to $n-1$ do
            if $a_j$ is not marked then
                if $a_j \preceq a_i$ then
                    mark $a_i$; break inner loop
                if $a_i \preceq a_j$ then
                    mark $a_j$

$A \leftarrow \{\text{unmarked elements in } A\}$
return $A$
```

- Poor locality: $A$ is scanned for $n$ times, $Q(n) = \Theta(n^2/L)$.
- Parallelizing these loops require **locks**.
Challenges and Solutions

▲ Improve data locality, say cache complexity $Q(n) \in O\left(\frac{n^2}{ZL}\right)$ instead of $\Theta(n^2/L)$; $Z$ and $L$ are the cache size and line size.

▲ Load balancing.

▲ Obtain good scalability on multi-cores.

▲ Handle very large poset, say $n \simeq 10^7$. 
Challenges and Solutions

▲ Improve data locality, say cache complexity $Q(n) \in O\left(\frac{n^2}{ZL}\right)$ instead of $\Theta(n^2/L)$; $Z$ and $L$ are the cache size and line size.

▲ Load balancing.

▲ Obtain good scalability on multi-cores.

▲ Handle very large poset, say $n \simeq 10^7$.

▲ Traverse the iteration space in a divide-n-conquer manner (Matteo Frigo’s techniques for cache oblivious stencil computations and N-body problems (2005)).

▲ Generate $A$ and compute $\text{Min}(A)$ concurrently.
Parallel Min Algorithm

Algorithm 2: ParMinPoset(A)

if |A| ≤ MIN\_BASE then
  return SerMinPoset(A)

(A^-, A^+) ← Split(A)
A^- ← spawn ParMinPoset(A^-)
A^+ ← spawn ParMinPoset(A^+)

sync
(A^-, A^+) ← ParMinMerge(A^-, A^+)
return Union(A^-, A^+)

*MIN\_BASE must be large enough to reduce parallelization overheads and small enough to increase data locality.
Parallel Merge of Min($B$) and Min($C$) (1/2)

**Algorithm 3:** ParMinMerge($B, C$) for Min($B) = B$ and Min($C) = C$

\[
\text{if } |B| \leq \text{MIN\_MERGE\_BASE} \text{ and } |C| \leq \text{MIN\_MERGE\_BASE} \text{ then}
\]
\[
\quad \text{return SerMinMerge}(B, C)
\]
\[
\text{else if } |B| > \text{MIN\_MERGE\_BASE} \text{ and } |C| > \text{MIN\_MERGE\_BASE} \text{ then}
\]
\[
\quad (B^-, B^+) \leftarrow \text{Split}(B); (C^-, C^+) \leftarrow \text{Split}(C)
\]
\[
\quad (B^-, C^-) \leftarrow \text{spawn} \text{ ParMinMerge}(B^-, C^-)
\]
\[
\quad (B^+, C^+) \leftarrow \text{spawn} \text{ ParMinMerge}(B^+, C^+)
\]
\[
\quad \text{sync}
\]
\[
\quad (B^-, C^+) \leftarrow \text{spawn} \text{ ParMinMerge}(B^-, C^+)
\]
\[
\quad (B^+, C^-) \leftarrow \text{spawn} \text{ ParMinMerge}(B^+, C^-)
\]
\[
\quad \text{sync}
\]
\[
\quad \text{return} (\text{Union}(B^-, B^+), \text{Union}(C^-, C^+))
\]

..............
Parallel Merge of $\text{Min}(B)$ and $\text{Min}(C)$ (2/2)

**Algorithm 4**: $\text{ParMinMerge}(B, C)$ for $\text{Min}(B) = B$ and $\text{Min}(C) = C$

if $|B| \leq \text{MIN\_MERGE\_BASE}$ and $|C| \leq \text{MIN\_MERGE\_BASE}$ then

else if $|B| > \text{MIN\_MERGE\_BASE}$ and $|C| > \text{MIN\_MERGE\_BASE}$ then

else if $|B| > \text{MIN\_MERGE\_BASE}$ and $|C| \leq \text{MIN\_MERGE\_BASE}$ then

  $(B^-, B^+) \leftarrow \text{Split}(B)$

  $(B^-, C) \leftarrow \text{ParMinMerge}(B^-, C)$

  $(B^+, C) \leftarrow \text{ParMinMerge}(B^+, C)$

return $(\text{Union}(B^-, B^+), C)$


Complexity Results


- The worst case occurs when $A = \text{Min}(A)$ holds.

- In this case, setting all thresholds to one, we have:
  - the cache complexity $Q(n) \in \Theta\left(\frac{n^2}{ZL} + \frac{n}{L}\right)$
  - the work $T_1(n) \in \Theta(n^2)$
  - the critical path (or span) $T_\infty(n) \in \Theta(n)$
  - and thus the parallelism is $\Theta(n)$
Scalability Analysis by Cilkview

Computing the minimal elements of 500,000 random natural numbers

Parallelism = 4025, Ideal Speedup
Lower Performance Bound
Measured Speedup
ParTran: Example
ParTran: Example

\[ \text{Tr}(\mathcal{H}) = \text{Min}(\text{Tr}(E_1 \cup E_2) \lor \text{Tr}(E_3 \cup E_4)) \]
ParTran: Example

\[ \text{Tr}(\mathcal{H}) = \text{Min}(\text{Tr}(E_1 \cup E_2) \lor \text{Tr}(E_3 \cup E_4)) \]

\[ \text{Tr}(E_1 \cup E_2) = \text{Min}(\text{Tr}(E_1) \lor \text{Tr}(E_2)) = \text{Min}(\{1, 2\} \lor \{2, 3, 4\}) \]

\[ \text{Tr}(E_3 \cup E_4) = \text{Min}(\text{Tr}(E_3) \lor \text{Tr}(E_4)) = \text{Min}(\{3, 4, 5\} \lor \{5, 6\}) \]
ParTran: Example

\[ \text{Tr}(\mathcal{H}) = \text{Min}(\text{Tr}(E_1 \cup E_2) \lor \text{Tr}(E_3 \cup E_4)) \]

\[ \text{Tr}(E_1 \cup E_2) = \text{Min}(\text{Tr}(E_1) \lor \text{Tr}(E_2)) = \text{Min}([1, 2] \lor [2, 3, 4]) \]
\[ \text{Tr}(E_3 \cup E_4) = \text{Min}(\text{Tr}(E_3) \lor \text{Tr}(E_4)) = \text{Min}([3, 4, 5] \lor [5, 6]) \]

\[ \text{Min}([1, 2] \lor [2, 3, 4]) \]
\[ = \text{MinMerge}([\text{Min}([1] \lor [2, 3]), \text{Min}([2] \lor [4])], \]
\[ \{\text{Min}([1] \lor [4]), \text{Min}([2] \lor [2, 3])\}) \]

\[ \text{Min}([3, 4, 5] \lor [5, 6]) \]
\[ = \text{MinMerge}(\cdots) \]
ParTran: Example

\[ \text{Tr}(\mathcal{H}) = \min(\text{Tr}(E_1 \cup E_2) \lor \text{Tr}(E_3 \cup E_4)) \]

\[ \text{Tr}(E_1 \cup E_2) = \min(\text{Tr}(E_1) \lor \text{Tr}(E_2)) = \min(\{1, 2\} \lor \{2, 3, 4\}) \]
\[ \text{Tr}(E_3 \cup E_4) = \min(\text{Tr}(E_3) \lor \text{Tr}(E_4)) = \min(\{3, 4, 5\} \lor \{5, 6\}) \]

\[ \min(\{1, 2\} \lor \{2, 3, 4\}) \]
\[ = \min\text{Merge}(\{\min(\{1\} \lor \{2, 3\}), \min(\{2\} \lor \{4\})\}, \{\min(\{1\} \lor \{4\}), \min(\{2\} \lor \{2, 3\})\}) \]
\[ = \min\text{Merge}(\{12, 13, 24\}, \{14, 2\}) = \{13, 14, 2\} \]

\[ \min(\{3, 4, 5\} \lor \{5, 6\}) \]
\[ = \min\text{Merge}(\cdots) = \cdots = \{36, 46, 5\} \]
ParTran: Example

\[\text{Tr}(\mathcal{H}) = \text{Min}(\text{Tr}(E_1 \cup E_2) \lor \text{Tr}(E_3 \cup E_4))\]

\[\text{Tr}(E_1 \cup E_2) = \text{Min}(\text{Tr}(E_1) \lor \text{Tr}(E_2)) = \text{Min}([1, 2] \lor \{2, 3, 4\})\]
\[\text{Tr}(E_3 \cup E_4) = \text{Min}(\text{Tr}(E_3) \lor \text{Tr}(E_4)) = \text{Min}([3, 4, 5] \lor \{5, 6\})\]

\[\text{Min}([1, 2] \lor \{2, 3, 4\})\]
\[= \text{MinMerge}([\text{Min}([1] \lor \{2, 3\}), \text{Min}([2] \lor \{4\})),\]
\[\{\text{Min}([1] \lor \{4\}), \text{Min}([2] \lor \{2, 3\})\}]\]
\[= \text{MinMerge}([12, 13, 24], [14, 2]) = \{13, 14, 2\}\]

\[\text{Min}([3, 4, 5] \lor \{5, 6\})\]
\[= \text{MinMerge}(\cdots) = \cdots = \{36, 46, 5\}\]

\[\text{Tr}(\mathcal{H}) = \text{Min}(\text{Tr}(E_1 \cup E_2) \lor \text{Tr}(E_3 \cup E_4))\]
\[= \text{Min}([13, 14, 2] \lor \{36, 46, 5\}) = \text{MinMerge}(\cdots)\]
\[= \{135, 136, 145, 146, 236, 246, 25\}\]
Solving some Well-known Problems

<table>
<thead>
<tr>
<th>Parameters</th>
<th>BEGK</th>
<th>BMR</th>
<th>*KS</th>
<th>ParTran</th>
<th>ParTran’s Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>m</td>
<td>t</td>
<td></td>
<td>1P(s)</td>
<td>32P(s)</td>
</tr>
<tr>
<td>140</td>
<td>4900</td>
<td>71</td>
<td>22</td>
<td>194</td>
<td>11</td>
</tr>
<tr>
<td>160</td>
<td>6400</td>
<td>81</td>
<td>40</td>
<td>460</td>
<td>23</td>
</tr>
<tr>
<td>180</td>
<td>8100</td>
<td>91</td>
<td>75</td>
<td>1000</td>
<td>44</td>
</tr>
<tr>
<td>200</td>
<td>10000</td>
<td>101</td>
<td>289</td>
<td>1968</td>
<td>82</td>
</tr>
</tbody>
</table>

*Threshold hypergraphs*

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Dual Matching hypergraphs*

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Data Mining hypergraphs*

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Scalability Analysis by Cilkview

Data mining large dataset 1 (n = 287, m = 48226, t = 97)

Parallelism = 450, Ideal Speedup
Lower Performance Bound
Measured Speedup

ParTran for data mining problem #1
Scalability Analysis by Cilkview

Data mining large dataset 3 (n = 287, m = 108721, t = 99)

Parallelism = 1474, Ideal Speedup
Lower Performance Bound
Measured Speedup

ParTran for data mining problem #3
Solving some Classical Hypergraphs

Kuratowski Hypergraphs \( (K_r^n) \)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>KS ((s))</th>
<th>ParTran ((s))</th>
<th>Speedup</th>
<th>ParTran ((s))</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n ), (r ), (m ), (t )</td>
<td>1P ((s))</td>
<td>16P ((s))</td>
<td>32P ((s))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30, 5, 142506, 27405</td>
<td>6500</td>
<td>88</td>
<td>6</td>
<td>14.7</td>
<td>3.5</td>
</tr>
<tr>
<td>40, 5, 658008, 91390</td>
<td>&gt;15 hr</td>
<td>915</td>
<td>58</td>
<td>15.8</td>
<td>30</td>
</tr>
<tr>
<td>30, 7, 2035800, 593775</td>
<td>&gt;15 hr</td>
<td>72465</td>
<td>4648</td>
<td>15.6</td>
<td>2320</td>
</tr>
</tbody>
</table>

Lovasz Hypergraphs

<table>
<thead>
<tr>
<th>Parameters</th>
<th>KS ((s))</th>
<th>ParTran ((s))</th>
<th>Speedup</th>
<th>ParTran ((s))</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n ), (r ), (m ), (t )</td>
<td>1P ((s))</td>
<td>16P ((s))</td>
<td>32P ((s))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36, 8, 69281, 69281</td>
<td>8000</td>
<td>119</td>
<td>13</td>
<td>8.9</td>
<td>10</td>
</tr>
<tr>
<td>45, 9, 623530, 623530</td>
<td>&gt;15 hr</td>
<td>8765</td>
<td>609</td>
<td>14.2</td>
<td>347</td>
</tr>
<tr>
<td>55, 10, 6235301, 6235301</td>
<td>&gt;15 hr</td>
<td>-</td>
<td>60509</td>
<td>-</td>
<td>30596</td>
</tr>
</tbody>
</table>
Conclusion and Work in Progress

- We provide a parallel algorithm and an implementation for computing the transversal of hypergraphs targeting multi-cores.
Conclusion and Work in Progress

- We provide a parallel algorithm and an implementation for computing the transversal of hypergraphs targeting multi-cores.
- Our program performs well on a number of large problems.
Conclusion and Work in Progress

- We provide a parallel algorithm and an implementation for computing the transversal of hypergraphs targeting multi-cores.
- Our program performs well on a number of large problems.
- We have identified the *computation of the minimal elements of a poset* as a core routine in many applications. Up to our knowledge, we provide the first parallel and cache-efficient algorithm for this task.
Conclusion and Work in Progress

- We provide a parallel algorithm and an implementation for computing the transversal of hypergraphs targeting multi-cores.

- Our program performs well on a number of large problems.

- We have identified the computation of the minimal elements of a poset as a core routine in many applications. Up to our knowledge, we provide the first parallel and cache-efficient algorithm for this task.

- Work in progress:
  - apply the techniques of Kavvadias and Stavropoulos (and others) to improve the performance of our program for some small size hypergraphs.
  - attack other graph-theoretic algorithms and their applications.
Acknowledgements

Sincere thanks to our colleagues Dimitris J. Kavvadias and Elias C. Stavropoulos for providing us with their program (implementing the KS algorithm) and their test suits in a timely manner.

We are grateful to Matteo Frigo for fruitful discussions on cache-oblivious algorithms and Cilk++.

Our benchmarks were made possible by the dedicated resource program of SHARCNET.

Thank you!
Incremental Approach: Example

\[ E_1, E_2, E_3, E_4 \]

1, 2, 3, 4, 5, 6
Incremental Approach: Example

\[ \text{Tr}(\mathcal{H}_1) = \{1, 2\} \]
Incremental Approach: Example

\[ \text{Tr}(\mathcal{H}_1) = \{1, 2\} \]

\[ \text{Tr}(\mathcal{H}_2) = \text{Min}(\{1, 2\} \vee \{2, 3, 4\}) = \text{Min}(\{12, 13, 14, 2, 23, 24\}) = \{13, 14, 2\} \]
Incremental Approach: Example

◮ \( \text{Tr}(\mathcal{H}_1) = \{1, 2\} \)

◮ \( \text{Tr}(\mathcal{H}_2) = \text{Min}(\{1, 2\} \lor \{2, 3, 4\}) \)
   \(= \text{Min}(\{12, 13, 14, 2, 23, 24\}) = \{13, 14, 2\} \)

◮ \( \text{Tr}(\mathcal{H}_3) = \text{Min}(\{13, 14, 2\} \lor \{3, 4, 5\}) \)
   \(= \text{Min}(\{13, 134, 135, 143, 14, 145, 23, 24, 25\}) \)
   \(= \{13, 14, 23, 24, 25\} \)
Incremental Approach: Example

- \( \text{Tr}(\mathcal{H}_1) = \{1, 2\} \)

- \( \text{Tr}(\mathcal{H}_2) = \text{Min}(\{1, 2\} \lor \{2, 3, 4\}) \)
  \[ = \text{Min}(\{12, 13, 14, 2, 23, 24\}) = \{13, 14, 2\} \]

- \( \text{Tr}(\mathcal{H}_3) = \text{Min}(\{13, 14, 2\} \lor \{3, 4, 5\}) \)
  \[ = \text{Min}(\{13, 134, 135, 143, 14, 145, 23, 24, 25\}) \]
  \[ = \{13, 14, 23, 24, 25\} \]

- \( \text{Tr}(\mathcal{H}_3) = \text{Min}(\{13, 14, 23, 24, 25\} \lor \{5, 6\}) \)
  \[ = \text{Min}(\{135, 136, 145, 146, 235, 236, 245, 246, 25, 256\}) \]
  \[ = \{135, 136, 145, 146, 236, 246, 25\} \]
Incremental Approach: Example

\[ \text{Tr}(\mathcal{H}_1) = \{1, 2\} \]

\[ \text{Tr}(\mathcal{H}_2) = \text{Min}(\{1, 2\} \lor \{2, 3, 4\}) = \text{Min}(\{12, 13, 14, 2, 23, 24\}) = \{13, 14, 2\} \]

\[ \text{Tr}(\mathcal{H}_3) = \text{Min}(\{13, 14, 2\} \lor \{3, 4, 5\}) = \text{Min}(\{13, 134, 135, 143, 14, 145, 23, 24, 25\}) = \{13, 14, 23, 24, 25\} \]

\[ \text{Tr}(\mathcal{H}_3) = \text{Min}(\{13, 14, 23, 24, 25\} \lor \{5, 6\}) = \text{Min}(\{135, 136, 145, 146, 235, 236, 245, 246, 25, 256\}) = \{135, 136, 145, 146, 236, 246, 25\} \]

Note: the growth of the intermediate expression!
Parallel $\text{Tr}(\mathcal{H})$ Top Algorithm

Algorithm 5: ParTran

if $|\mathcal{H}| \leq \text{TR\_BASE}$ then
    return $\text{SerTran}(\mathcal{H})$;

$(\mathcal{H}^-, \mathcal{H}^+) \leftarrow \text{Split}(\mathcal{H})$

$\mathcal{H}^- \leftarrow \text{spawn ParTran}(\mathcal{H}^-)$

$\mathcal{H}^+ \leftarrow \text{spawn ParTran}(\mathcal{H}^+)$

sync

return $\text{ParHypMerge}(\mathcal{H}^-, \mathcal{H}^+)$