Parallel Generation of Transversal Hypergraphs

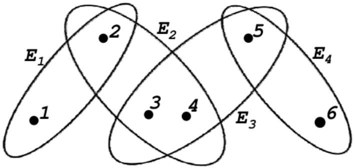
Yuzhen Xie
Computer Science Department
University of Western Ontario (UWO)
joint work with
Charles E. Leiserson (CSAIL, MIT),
Liyun Li and Marc Moreno Maza (UWO)

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

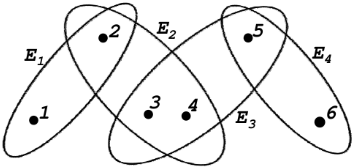


Note: a hyperedge can have more than two vertices.

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

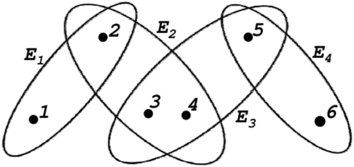
  Note: a hyperedge can have more than two vertices.

- 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. $T \cap E \neq \emptyset,\ \forall E \in \mathcal{E}$.

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.

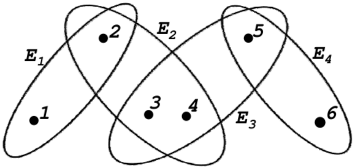
- A subset T of V is a transversal (or hitting set) of $\mathcal{H}$ if it intersects all the hyperedges of $\mathcal{H}$, i.e. $T \cap E \neq \emptyset,\ \forall E \in \mathcal{E}$.

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

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.

- 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. $T \cap E \neq \emptyset$, $\forall E \in \mathcal{E}$.

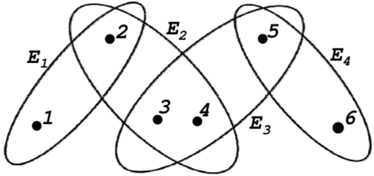
  A transversal T of $\mathcal{H}$ is minimal if no proper subset of T is a transversal of $\mathcal{H}$.

  Example: $\{25\}$ is a minimal transversal of $\mathcal{H}$; $\{235\}$ is a transversal but not minimal.

Transversal Hypergraph Generation (THG)

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

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

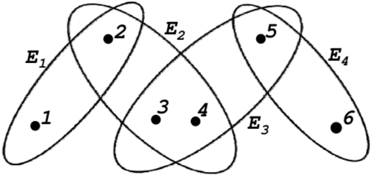


Note: the size (number of edges) of $\mathrm{Tr}(\mathcal{H})$ can be exponential in the order of $\mathcal{H}$ (number of vertices).

Transversal Hypergraph Generation (THG)

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

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



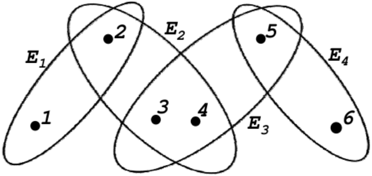
  Note: the size (number of edges) of $\mathrm{Tr}(\mathcal{H})$ can be exponential in the order of $\mathcal{H}$ (number of vertices).

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

Transversal Hypergraph Generation (THG)

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

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

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

- The **transversal hypergraph generation** problem is to compute $\mathrm{Tr}(\mathcal{H})$, given a hypergraph $\mathcal{H}$.
- **Numerous applications**: data mining, computational biology, artificial intelligence and logic, cryptography, semantic web, mobile communication systems, e-commerce, etc.

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

$$\mathrm{Tr}(\mathcal{H}' \cup \mathcal{H}'') = \mathrm{Min}(\mathrm{Tr}(\mathcal{H}') \vee \mathrm{Tr}(\mathcal{H}'')),$$

where

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

and $\mathrm{Min}(\mathcal{H}')$ 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

  $$\mathrm{Tr}(\mathcal{H}' \cup \mathcal{H}'') = \mathrm{Min}(\mathrm{Tr}(\mathcal{H}') \vee \mathrm{Tr}(\mathcal{H}'')),$$

  where

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

  and $\mathrm{Min}(\mathcal{H}')$ 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,

  $$\mathrm{Tr}(\mathcal{H}_{i+1}) = \mathrm{Min}(\mathrm{Tr}(\mathcal{H}_i) \vee (V, \{\{v\} \mid v \in E_{i+1}\})).$$

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.

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.

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 Khachiyan**'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; polylog($|V|, |\mathcal{H}|, k$) time assuming 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($|V|, |\mathcal{H}|, k$) time assuming poly($|V|, |\mathcal{H}|, k$) 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 $\mathcal{H}'$ and $\mathcal{H}''$ are of similar order.

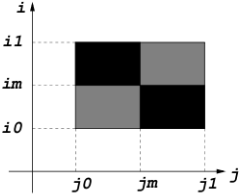
$$\mathsf{Tr}(\mathcal{H}' \cup \mathcal{H}'') = \mathsf{Min}(\mathsf{Tr}(\mathcal{H}') \vee \mathsf{Tr}(\mathcal{H}''))$$

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.

$$\mathrm{Tr}(\mathcal{H}' \cup \mathcal{H}'') = \mathrm{Min}(\mathrm{Tr}(\mathcal{H}') \vee \mathrm{Tr}(\mathcal{H}''))$$

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

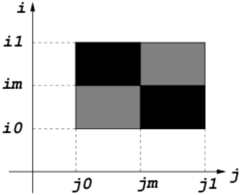


Our Parallel Transversal Algorithm: ParTran

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

$$\mathrm{Tr}(\mathcal{H}' \cup \mathcal{H}'') = \mathrm{Min}(\mathrm{Tr}(\mathcal{H}') \vee \mathrm{Tr}(\mathcal{H}''))$$

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



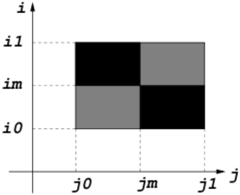
- Compute Min, again in a divide-n-conquer manner.

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.

$$\mathrm{Tr}(\mathcal{H}' \cup \mathcal{H}'') = \mathrm{Min}(\mathrm{Tr}(\mathcal{H}') \vee \mathrm{Tr}(\mathcal{H}''))$$

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



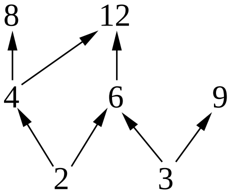
- Compute Min, again in a divide-n-conquer manner.
- Parallelism is created by the divide-n-conquer recursive calls.

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 $\mathrm{Tr}(\mathcal{H})$ and $\mathcal{H}' \vee \mathcal{H}''$ 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: $y \preceq x \;\Rightarrow y = x$.
- $\mathrm{Min}(A, \preceq)$, or simply $\mathrm{Min}(A)$ designates the set of the minimal elements of A.
- A poset example for the integer divisibility relation:



A Simple Procedure but ...

Algorithm 1: SerMinPoset

Input : a poset $A = \{a_0, \cdots, a_{n-1}\}$
Output : Min(A)

```
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 ⪯ a_i then
                    mark a_i; break inner loop
                if a_i ⪯ a_j then
                    mark a_j
A ← {unmarked elements in A}
return A
```

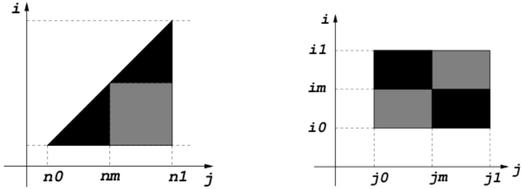
- Poor locality: A is scanned for n times, $Q(n) = \Theta(n^2/L)$.
- Parallelizing these loops require **locks**.

Challenges and Solutions

- Improve data locality, say cache complexity $Q(n) \in O(\frac{n^2}{ZL})$ 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 $Q(n) \in O(\frac{n^2}{ZL})$ 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 $n \simeq 10^7$.



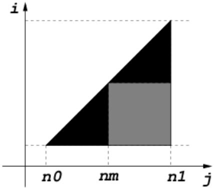
- Traverse the iteration space in a divide-n-conquer manner (Matteo Frigo's techniques for cache oblivious stencil computations and N-body problems (2005)).
- Generate A and compute Min(A) concurrently.

Parallel Min Algorithm

Algorithm 2: ParMinPoset(A)

if $|A| \leq \mathit{MIN\_BASE}$ **then**
  return SerMinPoset(A)
$(A^-, A^+) \leftarrow$ Split(A)
$A^- \leftarrow$ **spawn** ParMinPoset(A^-)
$A^+ \leftarrow$ **spawn** ParMinPoset(A^+)
sync
$(A^-, A^+) \leftarrow$ ParMinMerge(A^-, A^+)
return Union(A^-, A^+)

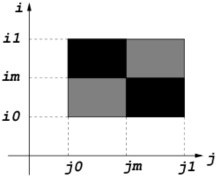
MIN_BASE* must be large enough to **reduce parallelization overheads and small enough to **increase data locality**.



Parallel Merge of Min(B) and Min(C) (1/2)

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

if $|B| \leq$ MIN_MERGE_BASE *and* $|C| \leq$ MIN_MERGE_BASE **then**
    return SerMinMerge(B, C)
else if $|B| >$ MIN_MERGE_BASE *and* $|C| >$ MIN_MERGE_BASE **then**
    $(B^-, B^+) \leftarrow$ Split(B); $(C^-, C^+) \leftarrow$ Split(C)
    $(B^-, C^-) \leftarrow$ **spawn** ParMinMerge(B^-, C^-)
    $(B^+, C^+) \leftarrow$ **spawn** ParMinMerge(B^+, C^+)
    sync
    $(B^-, C^+) \leftarrow$ **spawn** ParMinMerge(B^-, C^+)
    $(B^+, C^-) \leftarrow$ **spawn** ParMinMerge(B^+, C^-)
    sync
    return (Union(B^-, B^+), Union(C^-, C^+))
. . . .



Parallel Merge of Min(B) and Min(C) (2/2)

Algorithm 4: ParMinMerge(B, C) for Min(B) $= B$ and Min(C) $= C$

if $|B| \leq$ MIN_MERGE_BASE *and* $|C| \leq$ MIN_MERGE_BASE **then**

  . . . .

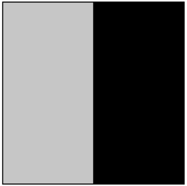
else if $|B| >$ MIN_MERGE_BASE *and* $|C| >$ MIN_MERGE_BASE **then**

  . . . .

else if $|B| >$ MIN_MERGE_BASE *and*
    $|C| \leq$ MIN_MERGE_BASE **then**

  $(B^-, B^+) \leftarrow$ Split(B)
  $(B^-, C) \leftarrow$ ParMinMerge(B^-, C)
  $(B^+, C) \leftarrow$ ParMinMerge(B^+, C)
  return (Union(B^-, B^+), C)

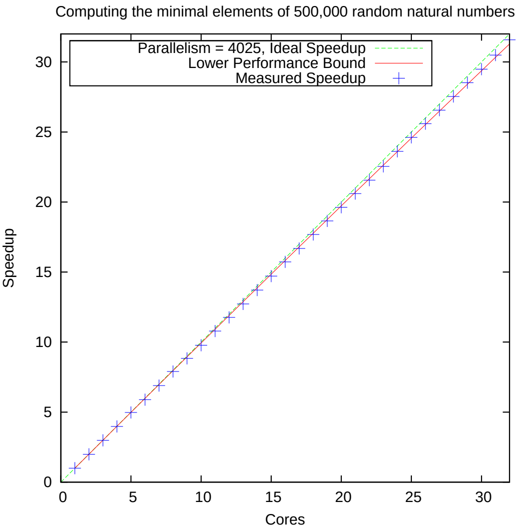
. . . .



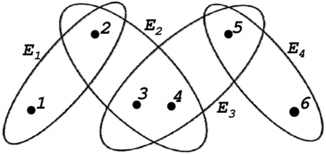
Complexity Results

- Our results are for the fork-join multi-threading parallelism (M. Frigo, C. E. Leiserson, and K. H. Randall, 1998) and the ideal cache model (M. Frigo, C. E. Leiserson, H. Prokop, & S. Ramachandran, 1999)
- The worst case occurs when $A = \mathrm{Min}(A)$ holds.
- In this case, setting all thresholds to one, we have:
  - the cache complexity $Q(n) \in \Theta(\frac{n^2}{ZL} + \frac{n}{L})$
  - 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 $\Theta(n)$

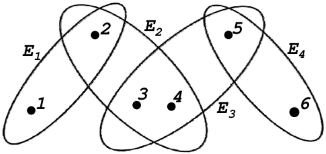
Scalability Analysis by Cilkview



ParTran: Example

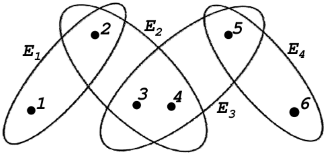


ParTran: Example



- $\mathrm{Tr}(\mathcal{H}) = \mathrm{Min}(\mathrm{Tr}(E_1 \cup E_2) \vee \mathrm{Tr}(E_3 \cup E_4))$

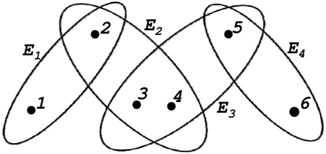
ParTran: Example



- $Tr(\mathcal{H}) = Min(Tr(E_1 \cup E_2) \vee Tr(E_3 \cup E_4))$
- $Tr(E_1 \cup E_2) = Min(Tr(E_1) \vee Tr(E_2)) = Min(\{1, 2\} \vee \{2, 3, 4\})$

  $Tr(E_3 \cup E_4) = Min(Tr(E_3) \vee Tr(E_4)) = Min(\{3, 4, 5\} \vee \{5, 6\})$

ParTran: Example



- $\mathrm{Tr}(\mathcal{H}) = \mathrm{Min}(\mathrm{Tr}(E_1 \cup E_2) \vee \mathrm{Tr}(E_3 \cup E_4))$
- $\mathrm{Tr}(E_1 \cup E_2) = \mathrm{Min}(\mathrm{Tr}(E_1) \vee \mathrm{Tr}(E_2)) = \mathrm{Min}(\{1, 2\} \vee \{2, 3, 4\})$

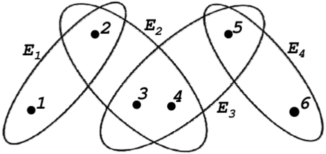
  $\mathrm{Tr}(E_3 \cup E_4) = \mathrm{Min}(\mathrm{Tr}(E_3) \vee \mathrm{Tr}(E_4)) = \mathrm{Min}(\{3, 4, 5\} \vee \{5, 6\})$
- $\mathrm{Min}(\{1, 2\} \vee \{2, 3, 4\})$

  $= \mathrm{MinMerge}(\{\mathrm{Min}(\{1\} \vee \{2, 3\}), \mathrm{Min}(\{2\} \vee \{4\})\}, \{\mathrm{Min}(\{1\} \vee \{4\}), \mathrm{Min}(\{2\} \vee \{2, 3\})\})$

  $\mathrm{Min}(\{3, 4, 5\} \vee \{5, 6\})$

  $= \mathrm{MinMerge}(\cdots)$

ParTran: Example

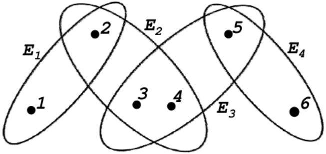


- $\mathrm{Tr}(\mathcal{H}) = \mathrm{Min}(\mathrm{Tr}(E_1 \cup E_2) \vee \mathrm{Tr}(E_3 \cup E_4))$
- $\mathrm{Tr}(E_1 \cup E_2) = \mathrm{Min}(\mathrm{Tr}(E_1) \vee \mathrm{Tr}(E_2)) = \mathrm{Min}(\{1, 2\} \vee \{2, 3, 4\})$

  $\mathrm{Tr}(E_3 \cup E_4) = \mathrm{Min}(\mathrm{Tr}(E_3) \vee \mathrm{Tr}(E_4)) = \mathrm{Min}(\{3, 4, 5\} \vee \{5, 6\})$
- $\mathrm{Min}(\{1, 2\} \vee \{2, 3, 4\})$
  $= \mathrm{MinMerge}(\{\mathrm{Min}(\{1\} \vee \{2, 3\}), \mathrm{Min}(\{2\} \vee \{4\})\},$
  $\{\mathrm{Min}(\{1\} \vee \{4\}), \mathrm{Min}(\{2\} \vee \{2, 3\})\})$
  $= \mathrm{MinMerge}(\{12, 13, 24\}, \{14, 2\}) = \{\mathbf{13}, \mathbf{14}, \mathbf{2}\}$

  $\mathrm{Min}(\{3, 4, 5\} \vee \{5, 6\})$
  $= \mathrm{MinMerge}(\cdots) = \cdots = \{\mathbf{36}, \mathbf{46}, \mathbf{5}\}$

ParTran: Example



- $\mathrm{Tr}(\mathcal{H}) = \mathrm{Min}(\mathrm{Tr}(E_1 \cup E_2) \ \vee \ \mathrm{Tr}(E_3 \cup E_4))$
- $\mathrm{Tr}(E_1 \cup E_2) = \mathrm{Min}(\mathrm{Tr}(E_1) \vee \mathrm{Tr}(E_2)) = \mathrm{Min}(\{1, 2\} \vee \{2, 3, 4\})$

  $\mathrm{Tr}(E_3 \cup E_4) = \mathrm{Min}(\mathrm{Tr}(E_3) \vee \mathrm{Tr}(E_4)) = \mathrm{Min}(\{3, 4, 5\} \vee \{5, 6\})$
- $\mathrm{Min}(\{1, 2\} \vee \{2, 3, 4\})$
  $= \mathrm{MinMerge}(\{\mathrm{Min}(\{1\} \vee \{2, 3\}), \mathrm{Min}(\{2\} \vee \{4\})\},$
  $\{\mathrm{Min}(\{1\} \vee \{4\}), \mathrm{Min}(\{2\} \vee \{2, 3\})\})$
  $= \mathrm{MinMerge}(\{12, 13, 24\}, \{14, 2\}) = \{\mathbf{13}, \mathbf{14}, \mathbf{2}\}$

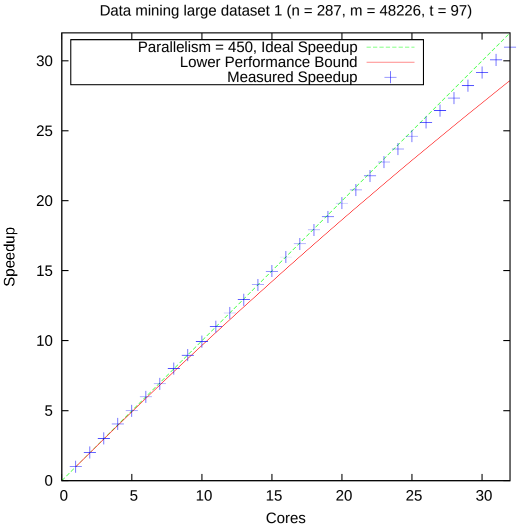
  $\mathrm{Min}(\{3, 4, 5\} \vee \{5, 6\})$
  $= \mathrm{MinMerge}(\cdots) = \ \cdots \ = \{\mathbf{36}, \mathbf{46}, \mathbf{5}\}$
- $\mathrm{Tr}(\mathcal{H}) = \mathrm{Min}(\mathrm{Tr}(E_1 \cup E_2) \ \vee \ \mathrm{Tr}(E_3 \cup E_4))$
  $= \mathrm{Min}(\{13, 14, 2\} \vee \{36, 46, 5\}) = \mathrm{MinMerge}(\cdots)$
  $= \{\mathbf{135}, \mathbf{136}, \mathbf{145}, \mathbf{146}, \mathbf{236}, \mathbf{246}, \mathbf{25}\}$

Solving some Well-known Problems

Parameters			BEGK	BMR	*KS	ParTran		ParTran's Gain	
n	m	t	(s)	(s)	(s)	1P(s)	32P(s)	KS/1P	KS/32P
Threshold hypergraphs									
140	4900	71	22	194	11	0.01	-	1000	-
160	6400	81	40	460	23	0.01	-	2000	-
180	8100	91	75	1000	44	0.01	-	4000	-
200	10000	101	289	1968	82	0.02	-	4000	-
Dual Matching hypergraphs									
34	131072	17	911	2360	57	9	0.6	6	100
36	262144	18	2188	12463	197	23	1.8	9	110
38	524288	19	8756	36600	655	56	3.5	12	186
40	1048576	20	35171	201142	2167	131	7.1	17	304
Data Mining hypergraphs									
287	48226	97	1332	1241	1648	92	3	18	549
287	92699	99	4388	4280	6672	651	21	10	318
287	108721	99	5898	7238	9331	1146	36	8	259

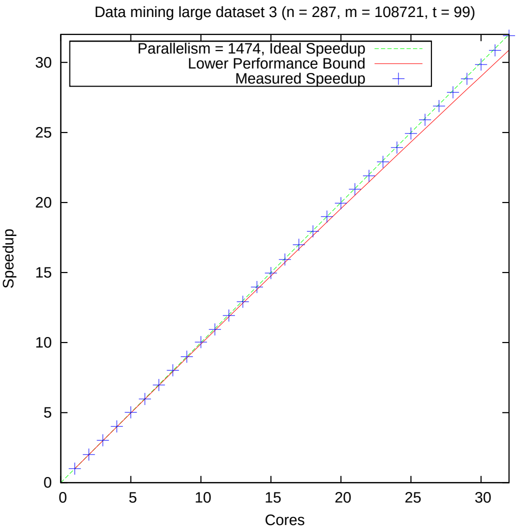
*KS: Kavvadias and Stavropoulos, http://lca.ceid.upatras.gr/estavrop/transversal/.
(Journal of Graph Algorithms and Applications, 9(2):239-264, 2005).

Scalability Analysis by Cilkview



ParTran for data mining problem #1

Scalability Analysis by Cilkview



ParTran for data mining problem #3

Solving some Classical Hypergraphs

Kuratowski Hypergraphs (K_n^r)

Parameters				KS	ParTran				
n	r	m	t	(s)	1P	16P		32P	
					(s)	(s)	Speedup	(s)	Speedup
30	5	142506	27405	6500	88	6	14.7	3.5	25.0
40	5	658008	91390	$>$15 hr	915	58	15.8	30	30.5
30	7	2035800	593775	$>$15 hr	72465	4648	15.6	2320	31.2

Lovasz Hypergraphs

Parameters				KS	ParTran				
n	r	m	t	(s)	1P	16P		32P	
					(s)	(s)	Speedup	(s)	Speedup
36	8	69281	69281	8000	119	13	8.9	10	11.5
45	9	623530	623530	$>$15 hr	8765	609	14.2	347	25.3
55	10	6235301	6235301	$>$15 hr	-	60509	-	30596	-

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

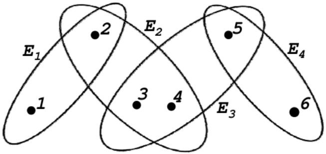
Sincere thanks to our colleagues Dimitris J. Kavvadias and Elias C. Stavropoulos for providing us with their program (implementing the KS algorithm) and their test suits in a timely manner.

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

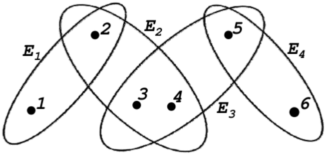
Our benchmarks were made possible by the dedicated resource program of SHARCNET.

Thank you!

Incremental Approach: Example

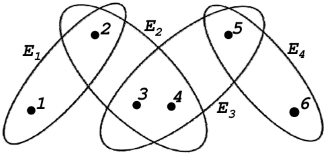


Incremental Approach: Example



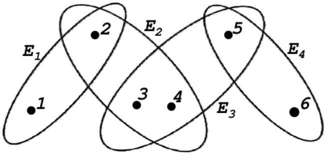
- $\mathrm{Tr}(\mathcal{H}_1) = \{1,\, 2\}$

Incremental Approach: Example



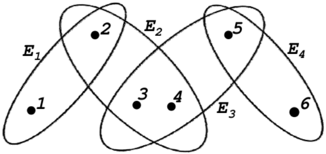
- $Tr(\mathcal{H}_1) = \{1, 2\}$
- $Tr(\mathcal{H}_2) = Min(\{1, 2\} \vee \{2, 3, 4\})$
  $= Min(\{12, 13, 14, 2, 23, 24\}) = \{13, 14, 2\}$

Incremental Approach: Example



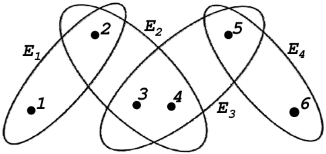
- $\mathrm{Tr}(\mathcal{H}_1) = \{1, 2\}$
- $\mathrm{Tr}(\mathcal{H}_2) = \mathrm{Min}(\{1, 2\} \vee \{2, 3, 4\})$
  $= \mathrm{Min}(\{12, 13, 14, 2, 23, 24\}) = \{13, 14, 2\}$
- $\mathrm{Tr}(\mathcal{H}_3) = \mathrm{Min}(\{13, 14, 2\} \vee \{3, 4, 5\})$
  $= \mathrm{Min}(\{13, 134, 135, 143, 14, 145, 23, 24, 25\})$
  $= \{13, 14, 23, 24, 25\}$

Incremental Approach: Example



- $\text{Tr}(\mathcal{H}_1) = \{1, 2\}$
- $\text{Tr}(\mathcal{H}_2) = \text{Min}(\{1, 2\} \vee \{2, 3, 4\})$
  $= \text{Min}(\{12, 13, 14, 2, 23, 24\}) = \{13, 14, 2\}$
- $\text{Tr}(\mathcal{H}_3) = \text{Min}(\{13, 14, 2\} \vee \{3, 4, 5\})$
  $= \text{Min}(\{13, 134, 135, 143, 14, 145, 23, 24, 25\})$
  $= \{13, 14, 23, 24, 25\}$
- $\text{Tr}(\mathcal{H}_3) = \text{Min}(\{13, 14, 23, 24, 25\} \vee \{5, 6\})$
  $= \text{Min}(\{135, 136, 145, 146, 235, 236, 245, 246, 25, 256\})$
  $= \{\mathbf{135}, \mathbf{136}, \mathbf{145}, \mathbf{146}, \mathbf{236}, \mathbf{246}, \mathbf{25}\}$

Incremental Approach: Example



- $\mathrm{Tr}(\mathcal{H}_1) = \{1, 2\}$
- $\mathrm{Tr}(\mathcal{H}_2) = \mathrm{Min}(\{1, 2\} \vee \{2, 3, 4\})$
  $= \mathrm{Min}(\{12, 13, 14, 2, 23, 24\}) = \{13, 14, 2\}$
- $\mathrm{Tr}(\mathcal{H}_3) = \mathrm{Min}(\{13, 14, 2\} \vee \{3, 4, 5\})$
  $= \mathrm{Min}(\{13, 134, 135, 143, 14, 145, 23, 24, 25\})$
  $= \{13, 14, 23, 24, 25\}$
- $\mathrm{Tr}(\mathcal{H}_3) = \mathrm{Min}(\{13, 14, 23, 24, 25\} \vee \{5, 6\})$
  $= \mathrm{Min}(\{135, 136, 145, 146, 235, 236, 245, 246, 25, 256\})$
  $= \{\mathbf{135}, \mathbf{136}, \mathbf{145}, \mathbf{146}, \mathbf{236}, \mathbf{246}, \mathbf{25}\}$

Note: the growth of the intermediate expression!

Parallel $\mathrm{Tr}(\mathcal{H})$ Top Algorithm

Algorithm 5: ParTran

if $|\mathcal{H}| \leq$ TR_BASE **then**
  return SerTran($\mathcal{H}$);
$(\mathcal{H}^-, \mathcal{H}^+) \leftarrow$ Split($\mathcal{H}$)
$\mathcal{H}^- \leftarrow$ **spawn** ParTran($\mathcal{H}^-$)
$\mathcal{H}^+ \leftarrow$ **spawn** ParTran($\mathcal{H}^+$)
sync
return ParHypMerge($\mathcal{H}^-, \mathcal{H}^+$)