Logical Aspects of Massively Parallel and Distributed Systems

Frank Neven
Hasselt University
PODS Tutorial

June 29, 2016
Logical aspects of massively parallel and distributed systems

<table>
<thead>
<tr>
<th>Topics</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity of query evaluation</td>
<td><em>synchronized parallel systems</em></td>
</tr>
<tr>
<td>Foundations of query optimization</td>
<td></td>
</tr>
<tr>
<td>Monotone recursive queries</td>
<td><em>asynchronous distributed systems</em></td>
</tr>
</tbody>
</table>
synchronous setting

**Question**

How can we understand the complexity of query processing on shared-nothing parallel architectures?
How can we understand the complexity of query processing on shared-nothing parallel architectures?

Examples: MapReduce, PigLatin, Hive, Dremel, Shark, ...
Topic #1: complexity of query evaluation

Synchronous setting

Question

How can we understand the complexity of query processing on shared-nothing parallel architectures?

- **Examples**: MapReduce, PigLatin, Hive, Dremel, Shark, ...
- **Characteristics**:
  - Large numbers of servers
  - Computation proceeds in rounds consisting of a communication and computation phase
How can we understand the complexity of query processing on shared-nothing parallel architectures?

- **Examples**: MapReduce, PigLatin, Hive, Dremel, Shark, ...
- **Characteristics**:
  - Large numbers of servers
  - Computation proceeds in rounds consisting of a communication and computation phase
- **Focus**: join queries and Shares/HyperCube algorithm
Question

How can we avoid reshuffling of data when evaluating queries?
Topic #2: foundations of query optimization

synchronous setting

Question

How can we avoid reshuffling of data when evaluating queries?

- Framework for reasoning on data partitioning
- Parallel-correctness and transferability
- Focus: join queries and single-round parallel algorithms
When can we avoid synchronization barriers in asynchronous parallel systems?
Question

When can we avoid synchronization barriers in asynchronous parallel systems?

- Coordination-freeness
- CALM: consistency and logical monotonicity
- Focus: monotone and Datalog queries (and extensions)
Outline

1 Introduction

2 Complexity of parallel query processing
   - Massively Parallel Communication Model
   - Single-Round MPC: Shares/HyperCube

3 Reasoning about data partitioning

4 Avoiding coordination

5 Discussion
Outline

1 Introduction

2 Complexity of parallel query processing
   - Massively Parallel Communication Model
     - Single-Round MPC: Shares/HyperCube

3 Reasoning about data partitioning

4 Avoiding coordination

5 Discussion
Aim of MPC [Koutris, Suciu, PODS 2011]
Understand the complexity of parallel query processing on shared-nothing architectures

Examples
MapReduce, Spark, Pig, Hive, Dremel, ...
Massively Parallel Communication Model (MPC)

**Aim of MPC [Koutris, Suciu, PODS 2011]**
Understand the complexity of parallel query processing on shared-nothing architectures

**Examples**
MapReduce, Spark, Pig, Hive, Dremel, …

**Characteristics**
- Computation is performed by $p$ servers connected by a complete network of private channels
- Computation proceeds in rounds:
  - *Communication phase*: servers exchange data
  - *Computation phase*: each server performs only local computation
Massively Parallel Communication Model
Massively Parallel Communication Model

Round 1

INPUT

DB

DB

DB

1

1

1

1

1

1

1

1

1

1

1

1

…

…

…

…

…

Round 1 Round 2 Round 3 Round r

DB

DB

DB

1

1

1

1

2

2

2

2

p

p

p

p

…

…

…

…

…

Round 1 Round 2 Round 3 Round r

OUTPUT

INPUT

Complexity of parallel query processing
Massively Parallel Communication Model

![Diagram of Massively Parallel Communication Model]

- **Round 1**
  - **INPUT**
  - DB
  - DB
  - DB
  - DB

- **Round 2**
  - DB
  - DB
  - DB
  - DB

- **OUTPUT**
  - DB
  - DB
  - DB
  - DB

- **Round r**
  - DB
  - DB
  - DB
  - DB

- Complexity of parallel query processing
Massively Parallel Communication Model

Round 1 Round 2 Round 3 Round r

Round 1 Round 2 Round 3 Round r

Complexity of parallel query processing Massively Parallel Communication Model PODS June 29, 2016 9 / 62
How do we measure complexity?

Possible measures

1. **Load:**
   - The load at each server is the amount of data received by a server during a particular round.
   - Minimization targets:
     - Total load (communication cost)
     - Maximal load (load balancedness)

2. **Number of rounds**

Typically ignoring the complexity of local computations.
Outline

1. Introduction

2. Complexity of parallel query processing
   - Massively Parallel Communication Model
   - Single-Round MPC: Shares/HyperCube

3. Reasoning about data partitioning

4. Avoiding coordination

5. Discussion
Shares/HyperCube algorithm: single-round multi-join evaluation [Afrati, Ullman, EDBT 2010], [Beame, Koutris, Suciu, PODS 2013]
Shares/HyperCube algorithm: single-round multi-join evaluation [Afrati, Ullman, EDBT 2010], [Beame, Koutris, Suciu, PODS 2013]

- Triangle Query $H(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$ (3-ary query)
- Identify every server with a coordinate in the 3-dimensional hypercube $[1, \alpha_x] \times [1, \alpha_y] \times [1, \alpha_z]$.

- $\alpha_x$ is the share of variable $x$
  (similar for $y$ and $z$)
Shares/HyperCube algorithm: single-round multi-join evaluation [Afrati, Ullman, EDBT 2010], [Beame, Koutris, Suciu, PODS 2013]

- Triangle Query $H(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$ (3-ary query)
- Identify every server with a coordinate in the 3-dimensional hypercube $[1, \alpha_x] \times [1, \alpha_y] \times [1, \alpha_z]$.

- $\alpha_x$ is the share of variable $x$
  (similar for $y$ and $z$)
- hash function $h_x$ mapping domain values to $[1, \alpha_x]$
  (similar for $y$ and $z$)
Shares/HyperCube algorithm: single-round multi-join evaluation [Afrati, Ullman, EDBT 2010], [Beame, Koutris, Suciu, PODS 2013]

Triangle Query $H(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$

Communication phase:

- $R(a, b)$ is sent to every server $(h_x(a), h_y(b), \star)$ (replicating every tuple $\alpha_z$ times)
Triangle Query $H(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$

**Communication phase:**

- $R(a, b)$ is sent to every server $(h_x(a), h_y(b), \star)$ (replicating every tuple $\alpha_z$ times)
- $S(b, c)$ is sent to every server $(\star, h_y(b), h_z(c))$ (replicating every tuple $\alpha_x$ times)
- $T(c, a)$ is sent to every server $(h_x(a), \star, h_z(c))$ (replicating every tuple $\alpha_y$ times)
Triangle Query $H(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$

Communication phase:
- $R(a, b)$ is sent to every server $(h_x(a), h_y(b), \star)$ (replicating every tuple $\alpha_z$ times)
- $S(b, c)$ is sent to every server $(\star, h_y(b), h_z(c))$ (replicating every tuple $\alpha_x$ times)
- $T(c, a)$ is sent to every server $(h_x(a), \star, h_z(c))$ (replicating every tuple $\alpha_y$ times)

Computation phase: All servers evaluate the triangle query.
Triangle Query: $H(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$

**Property**

For every valuation $V = \{ x \mapsto a, y \mapsto b, z \mapsto c \}$ the required facts $\{R(a, b), S(b, c), T(c, a)\}$ meet at server with coordinate $(h_x(a), h_y(b), h_z(c))$
Shares/HyperCube: some results

- minimize total load: solving Lagrangean equations [Afrati, Ullman, EDBT 2010]

- minimize maximal load [Beame, Koutris, Suciu, PODS 2013, 2014]

Lower bound: any 1-round algorithm requires at least load $\frac{M}{p} \frac{1}{\tau} \ast (Q)$ per server ($\tau \ast (Q)$ is optimal fractional vertex cover number of $Q$)

HyperCube provides matching upper bound

Practical? [Chu, Balazinska, Suciu, SIGMOD 2015] [Afrati, Ullman, TKDE 2011]

Rounding issues strong for queries with large intermediate results weak on queries with small output
Shares/HyperCube: some results

- **minimize total load**: solving Lagrangean equations [Afrati, Ullman, EDBT 2010]

- **minimize maximal load** [Beame, Koutris, Suciu, PODS 2013, 2014]
  - lower bound: any 1-round algorithm requires at least load

\[ \frac{M}{p^{1/\tau^*(Q)}} \] 

per server (\(\tau^*(Q)\) is optimal fractional vertex cover number of \(Q\))

- HyperCube provides matching upper bound
Shares/HyperCube: some results

- minimize **total load**: solving Lagrangean equations [Afrati, Ullman, EDBT 2010]

- minimize **maximal load** [Beame, Koutris, Suciu, PODS 2013, 2014]
  - lower bound: any 1-round algorithm requires at least load
    \[ M/p^{1/\tau^*(Q)} \]
    per server (\(\tau^*(Q)\) is optimal fractional vertex cover number of \(Q\))
  - HyperCube provides matching upper bound

- Practical? [Chu, Balazinska, Suciu, SIGMOD 2015]
  [Afrati, Ullman, TKDE 2011]
  - rounding issues
  - **strong** for queries with large intermediate results
  - **weak** on queries with small output
Intermediate wrap-up:

Summary:

- MPC as a model to study parallel query evaluation
- Shares/HyperCube fundamental algorithm
Intermediate wrap-up:

Summary:
- MPC as a model to study parallel query evaluation
- Shares/HyperCube fundamental algorithm

Research opportunities:
- Skew [Beame, Koutris, Suciu, PODS 2014]
  [Afrati, Stasinopoulos, Ullman, Vassilakopoulos, 2015]
- Multi-round
  - GYM [Afrati, Joglekar, R, Salihoglu, Ullman, 2014]
  - lower-bound: any multi-round algorithm requires at least load
    \[ M/\rho^{1/\rho^*(Q)} \]
    per server \((\rho^*(Q)\) is optimal fractional edge cover number of \(Q)\)
  [Beame, Koutris, Suciu, ICDT 2016]
- matching upper bound for binary relations [Ketsman, Suciu, 2016]
Outline

1. Introduction

2. Complexity of parallel query processing

3. Reasoning about data partitioning
   - Parallel-Correctness
   - Transferability of parallel-correctness

4. Avoiding coordination

5. Discussion
Observations on HyperCube

Distribution policies

- HyperCube reshuffles data on the granularity of facts
Observations on HyperCube

Distribution policies

- HyperCube reshuffles data on the granularity of facts
- Can be modeled by a distribution policy $P$: a function mapping every fact $f$ to a subset of servers $P(f)$
Observations on HyperCube

Distribution policies

- HyperCube reshuffles data on the granularity of facts
- Can be modeled by a distribution policy $P$: a function mapping every fact $f$ to a subset of servers $P(f)$

Generic one-round algorithm
(relative to a distribution policy $P$ and a query $Q$)

1. Reshuffle all data according to $P$
2. Evaluate $Q$ locally at every server
3. Output then is the union of all local results
Observations on HyperCube

Distribution policies

- HyperCube reshuffles data on the granularity of facts
- Can be modeled by a distribution policy $P$: a function mapping every fact $f$ to a subset of servers $P(f)$

Generic one-round algorithm
(relative to a distribution policy $P$ and a query $Q$)

1. Reshuffle all data according to $P$
2. Evaluate $Q$ locally at every server
3. Output then is the union of all local results

Potential for query optimization

Can we avoid the reshuffle step in the generic one-round algorithm?
Reasoning on data distribution

First scenario. The data is already distributed over the servers. Can we use the generic one-round algorithm to evaluate a given query using the current distribution?

- *if yes:* no data reshuffling needed
- *if no:* choose another distribution policy that works
First scenario. The data is already distributed over the servers. Can we use the generic one-round algorithm to evaluate a given query using the current distribution?

- *if yes:* no data reshuffling needed
- *if no:* choose another distribution policy that works

Decision problem: parallel-correctness
Reasoning on data distribution

**Second scenario.**
It may be unpractical to reason about distribution policies:
- too complex
- current distribution policy is unknown or hidden
- not chosen yet
Second scenario.
It may be unpractical to reason about distribution policies:
- too complex
- current distribution policy is unknown or hidden
- not chosen yet

Looking for ways to infer that two queries can *always* be evaluated correctly after another *without* an intermediate reshuffling step.
Second scenario.
It may be unpractical to reason about distribution policies:
- too complex
- current distribution policy is unknown or hidden
- not chosen yet

Looking for ways to infer that two queries can always be evaluated correctly after another without an intermediate reshuffling step.

Decision problem: parallel-correctness transfer
Outline

1. Introduction

2. Complexity of parallel query processing

3. Reasoning about data partitioning
   - Parallel-Correctness
   - Transferability of parallel-correctness

4. Avoiding coordination

5. Discussion
Parallel-correctness

Reasoning about data partitioning
Parallel-correctness

\[ [Q, P](I) = \bigcup_j Q(I_j) \]

\[ Q(I) = [Q, P](I) \]

centralized evaluation
distributed evaluation
Parallel-correctness: a bit more formal

Definition
A query $Q$ is parallel-correct on $I$ w.r.t. $P$ iff $Q(I) = [Q, P](I)$.

If $Q$ is monotone, then always $Q(I) \supseteq [Q, P](I)$
Parallel-correctness: a bit more formal

**Definition**

A query $Q$ is parallel-correct on $I$ w.r.t. $P$ iff $Q(I) = [Q, P](I)$.

If $Q$ is monotone, then always $Q(I) \supseteq [Q, P](I)$

**Definition**

A query $Q$ is parallel-correct w.r.t. $P$ iff $Q$ is parallel-correct w.r.t. $P$ on every $I$. 
Parallel-correctness for conjunctive queries

A valuation \( V \) for a conjunctive query \( Q \):

\[
H(x, z) \quad \iff \quad \text{head } Q \leftarrow R(x, y), S(y, z),
\]

is a mapping from the variables of \( Q \) to domain elements. If \( V(\text{body } Q) \subseteq I \) then \( V(\text{head } Q) \in Q(I) \).

Observation

If for every valuation \( V \) for \( Q \), there is a node \( \kappa \) in the network for which \( V(\text{body } Q) \subseteq P_{\text{out}}(\kappa) \), then \( Q \) is parallel-correct w.r.t. \( P \).

\( \star \) implies that for every \( I \),

\[
Q(I) \subseteq [Q, P](I)
\]

follows by monotonicity. Every CQ is parallel-correct w.r.t. its HyperCube distribution.
Parallel-correctness for conjunctive queries

A valuation $V$ for a conjunctive query

$$Q : H(x, z) \leftarrow R(x, y), S(y, z)$$

is a mapping from the variables of $Q$ to domain elements.
Parallel-correctness for conjunctive queries

A valuation $V$ for a conjunctive query

$$Q : \begin{cases} H(x, z) \leftarrow R(x, y), S(y, z) \end{cases}$$

is a mapping from the variables of $Q$ to domain elements. If $V(\text{body}_Q) \subseteq I$ then $V(\text{head}_Q) \in Q(I)$.
Parallel-correctness for conjunctive queries

A valuation $V$ for a conjunctive query

$$Q : \begin{array}{c}
H(x, z) \
\text{head}_Q
\end{array} \leftarrow \begin{array}{c}
R(x, y), S(y, z) \
\text{body}_Q
\end{array}$$

is a mapping from the variables of $Q$ to domain elements.
If $V(body_Q) \subseteq I$ then $V(head_Q) \in Q(I)$.

Observation

If for every valuation $V$ for $Q$, there is a node $\kappa$ in the network for which

$$V(body_Q) \subseteq P^{-1}(\kappa), \quad (\dagger)$$

then $Q$ is parallel-correct w.r.t. $P$. 
Parallel-correctness for conjunctive queries

A valuation $V$ for a conjunctive query

$$Q : H(x, z) \leftarrow R(x, y), S(y, z)$$

is a mapping from the variables of $Q$ to domain elements.
If $V(body_Q) \subseteq I$ then $V(head_Q) \in Q(I)$.

Observation

If for every valuation $V$ for $Q$, there is a node $\kappa$ in the network for which

$$V(body_Q) \subseteq P^{-1}(\kappa), \quad (\dagger)$$

then $Q$ is parallel-correct w.r.t. $P$

$$(\dagger)$$ implies that for every $I$, $Q(I) \subseteq [Q, P](I)$$
Parallel-correctness for conjunctive queries

A valuation $V$ for a conjunctive query

$$Q : H(x, z) \leftarrow R(x, y), S(y, z)$$

is a mapping from the variables of $Q$ to domain elements.
If $V(body_Q) \subseteq I$ then $V(head_Q) \in Q(I)$.

Observation

If for every valuation $V$ for $Q$, there is a node $\kappa$ in the network for which

$$V(body_Q) \subseteq P^{-1}(\kappa), \quad (\dagger)$$

then $Q$ is parallel-correct w.r.t. $P$

- $(\dagger)$ implies that for every $I$, $Q(I) \subseteq [Q, P](I)$
- $Q(I) \supseteq [Q, P](I)$ follows by monotonicity
Parallel-correctness for conjunctive queries

A valuation $V$ for a conjunctive query

$$Q : H(x, z) ← R(x, y), S(y, z)$$

is a mapping from the variables of $Q$ to domain elements. If $V(\text{body}_Q) \subseteq I$ then $V(\text{head}_Q) \in Q(I)$.

**Observation**

If for every valuation $V$ for $Q$, there is a node $\kappa$ in the network for which

$$V(\text{body}_Q) \subseteq P^{-1}(\kappa), \quad (\dagger)$$

then $Q$ is parallel-correct w.r.t. $P$

- $(\dagger)$ implies that for every $I$, $Q(I) \subseteq [Q, P](I)$
- $Q(I) \supseteq [Q, P](I)$ follows by monotonicity
- Every CQ is parallel-correct w.r.t. its HyperCube distribution
Sufficient but not a necessary condition: counterexample

Distribution policy $P$

\[ all - \{ R(a, b) \} \]

\[ all - \{ R(b, a) \} \]

Query $Q$: $H(x, z) \leftarrow R(x, y), R(y, z), R(x, x)$
Reasoning about data partitioning
Parallel-Correctness
PODS June 29, 2016
25 / 62

Sufficient but not a necessary condition: counterexample

Distribution policy $P$

$all - \{R(a, b)\}$

$all - \{R(b, a)\}$

$Q : H(x, z) \leftarrow R(x, y), R(y, z), R(x, x)$

$V = \{x, z \rightarrow a, y \rightarrow b\}$

**Derives:**

$H(a, a)$

**$V(\text{body}_Q)$ requires:**

$R(a, b) \quad R(b, a) \quad R(a, a)$

$\exists \kappa : V(\text{body}_Q) \subseteq P^{-1}(\kappa)$
Sufficient but not a necessary condition: counterexample

Distribution policy $P$

$all - \{R(a, b)\}$

Example

Query $Q$: $T(x, z) \leftarrow R(x, y), R(y, z), R(x, x)$

$V = \{x, z \rightarrow a, y \rightarrow b\}$

Derives:

$H(a, a)$

$V(\text{body}_Q)$ requires:

$R(a, b), R(b, a), R(a, a)$

$\not\exists \kappa: V(\text{body}_Q) \subseteq P^{-1}(\kappa)$

$V' = \{x, y, z \rightarrow a\}$

Derives:

$H(a, a)$

$V'(\text{body}_Q)$ requires:

$R(a, a)$

$\exists \kappa: V(\text{body}_Q) \subseteq P^{-1}(\kappa)$
Parallel-correctness for CQs: characterization

**Lemma**

Q is parallel-correct w.r.t. P iff for every minimal valuation V for Q, there is a node \( \kappa \) in the network for which

\[
V(body_Q) \subseteq P^{-1}(\kappa).
\]

**Definition**

V is minimal if no \( V' \) exists, where

- \( V'(head_Q) = V(head_Q) \); and,
- \( V'(body_Q) \subset V(body_Q) \).
Lemma

$Q$ is parallel-correct w.r.t. $P$ iff for every minimal valuation $V$ for $Q$, there is a node $\kappa$ in the network for which

$$V(\text{body}_Q) \subseteq P^{-1}(\kappa).$$
Parallel-correctness for CQs: complexity

**Lemma**

Q is parallel-correct w.r.t. P iff

for every *minimal* valuation V for Q,

there is a node κ in the network for which

\[ V(body_Q) \subseteq P^{-1}(\kappa). \]

- bound on the size of representations of nodes κ
- bound on the complexity of the test \( V(body_Q) \subseteq P^{-1}(\kappa). \)
Parallel-correctness for CQs: complexity

**Lemma**

Q is parallel-correct w.r.t. P iff

for every *minimal* valuation V for Q,

there is a node κ in the network for which

\[ V(\text{body}_Q) \subseteq P^{-1}(\kappa). \]

- bound on the size of representations of nodes κ
- bound on the complexity of the test \( V(\text{body}_Q) \subseteq P^{-1}(\kappa). \)

**Theorem (Ameloot, Geck, Ketsman, N., Schwentick, 2015)**

Deciding whether a CQ is parallel-correct w.r.t. P is \( \Pi^P_2 \)-complete.
Lemma

Q is parallel-correct w.r.t. P iff for every minimal valuation V for Q, there is a node κ in the network for which

\[ V(\text{body}_Q) \subseteq P^{-1}(\kappa). \]

- bound on the size of representations of nodes κ
- bound on the complexity of the test \( V(\text{body}_Q) \subseteq P^{-1}(\kappa). \)

Theorem (Ameloot, Geck, Ketsman, N., Schwentick, 2015)

Deciding whether a CQ is parallel-correct w.r.t. P is \( \Pi_2^P \)-complete.

Reduces to reasoning on properties of minimal valuations.
Parallel-correctness beyond CQs: adding unions, inequalities, negation

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a UCQ ≠ is parallel-correct is \( \Pi_2^P \)-complete.

Reduces again to reasoning on minimal valuations.
Parallel-correctness beyond CQs: adding unions, inequalities, negation

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a \( UCQ \neq \) is parallel-correct is \( \Pi_2^P \)-complete.

Reduces again to reasoning on minimal valuations.

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a \( CQ^\neg \) is parallel-correct is \( \text{coNEXPTIME} \)-complete.
Parallel-correctness beyond CQs: adding unions, inequalities, negation

Theorem (Geck, Ketsman, N., Schwentick, 2016)

**Deciding whether a UCQ \( \neq \) is parallel-correct is \( \Pi_2^p \)-complete.**

Reduces again to reasoning on minimal valuations.

Theorem (Geck, Ketsman, N., Schwentick, 2016)

**Deciding whether a CQ \( \neg \) is parallel-correct is coNEXPTIME-complete.**

- Non-monotone: parallel-soundness, parallel-completeness
Parallel-correctness beyond CQs: adding unions, inequalities, negation

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a \( UCQ \neq \) is parallel-correct is \( \Pi^P_2 \)-complete.

Reduces again to reasoning on minimal valuations.

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a \( CQ^- \) is parallel-correct is coNEXPTIME-complete.

- Non-monotone: parallel-soundness, parallel-completeness
- Upper bound: bound on size of smallest counterexample.
- Lower bound: reduction from \( CQ^- \) containment
Parallel-correctness beyond CQs: adding unions, inequalities, negation

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a UCQ $\neq$ is parallel-correct is $\Pi_2^P$-complete.

Reduces again to reasoning on minimal valuations.

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a CQ $\neg$ is parallel-correct is coNEXPTIME-complete.

- Non-monotone: parallel-soundness, parallel-completeness
- Upper bound: bound on size of smallest counterexample.
- Lower bound: reduction from CQ $\neg$ containment
- Requires reasoning on possible counterexamples that can be of exponential size.
Outline

1. Introduction

2. Complexity of parallel query processing

3. Reasoning about data partitioning
   - Parallel-Correctness
   - Transferability of parallel-correctness

4. Avoiding coordination

5. Discussion
Evaluating $Q'$ after $Q$ (for free)

$Q'$ can not be evaluated for free after $Q$
Evaluating $Q'$ after $Q$ (for free)

$Q'$ can sometimes be evaluated for free after $Q$
Evaluating $Q'$ after $Q$ (for free)

$Q'$ can always be evaluated for free after $Q$
Transferability of parallel-correctness

Definition

Parallel-correctness **transfers** from $Q$ to $Q'$, denoted $Q \rightarrow_T Q'$, iff

$Q'$ is parallel-correct on *every* distribution policy for which $Q$ is parallel-correct.
Transferability of parallel-correctness

Definition

Parallel-correctness transfers from $Q$ to $Q'$, denoted $Q \rightarrow_T Q'$, iff $Q'$ is parallel-correct on every distribution policy for which $Q$ is parallel-correct.

$$Q() \leftarrow R(x, y), R(y, z), R(z, w) \xrightarrow[?]{T} Q'() \leftarrow R(x, y), R(y, x)$$
Transferability of parallel-correctness

**Definition**

Parallel-correctness transfers from $Q$ to $Q'$, denoted $Q \rightarrow_T Q'$, iff $Q'$ is parallel-correct on every distribution policy for which $Q$ is parallel-correct.

$$Q() \leftarrow R(x, y), R(y, z), R(z, w) \quad \rightarrow_T^? \quad Q'() \leftarrow R(x, y), R(y, x)$$

The diagrams illustrate the facts required by $V(\text{body}_Q)$.
Transferability of parallel-correctness

**Definition**

Parallel-correctness transfers from $Q$ to $Q'$, denoted $Q \rightarrow_T Q'$, iff $Q'$ is parallel-correct on every distribution policy for which $Q$ is parallel-correct.

$$Q() \leftarrow R(x, y), R(y, z), R(z, w) \quad \rightarrow_T \quad Q'(()) \leftarrow R(x, y), R(y, x)$$

facts required by $V(\text{body}_Q)$

facts required by $V(\text{body}_{Q'})$
Lemma

For conjunctive queries $Q$ and $Q'$, $Q \rightarrow_T Q'$ iff for every minimal valuation $V'$ for $Q'$ there is a minimal valuation $V$ for $Q$ such that

$$V'(\text{body}_{Q'}) \subseteq V(\text{body}_Q).$$
Lemma

For conjunctive queries $Q$ and $Q'$, $Q \rightarrow_T Q'$ iff for every minimal valuation $V'$ for $Q'$ there is a minimal valuation $V$ for $Q$ such that $V'(body_{Q'}) \subseteq V(body_Q)$.

Theorem (Ameloot, Geck, Ketsman, N., Schwentick, 2015)

Deciding $Q \rightarrow_T Q'$ is $\Pi^P_3$-complete.

Only depends on the size of queries! Independent on representation of distribution policies.
Lowering complexity

Lemma

For conjunctive queries $Q$ and $Q'$, $Q \rightarrow_T Q'$ iff for every minimal valuation $V'$ for $Q'$ there is a minimal valuation $V$ for $Q$ such that $V'(\text{body}_{Q'}) \subseteq V(\text{body}_{Q})$. 
**Lowering complexity**

**Lemma**

For conjunctive queries \( Q \) and \( Q' \), \( Q \rightarrow_T Q' \) iff for every *minimal* valuation \( V' \) for \( Q' \), there is a *minimal* valuation \( V \) for \( Q \) such that

\[
V'(body_{Q'}) \subseteq V(body_Q).
\]

**Strongly minimal CQs**

- A CQ is **strongly** minimal if every valuation is minimal
- Example: full CQs, CQs without self-joins, ...
Lowering complexity

Lemma

For conjunctive queries $Q$ and $Q'$, $Q \rightarrow^T Q'$ iff
for every minimal valuation $V'$ for $Q'$
there is a minimal valuation $V$ for $Q$ such that
\[ V'(body_{Q'}) \subseteq V(body_Q). \]

Strongly minimal CQs

- A CQ is strongly minimal if every valuation is minimal
- Example: full CQs, CQs without self-joins, ...

Theorem

Let $Q$ be a strongly minimal CQ and $Q'$ be a CQ.

- Deciding $Q \rightarrow^T Q'$ is NP-complete.
- Deciding parallel-correctness of $Q$ w.r.t. a distribution policy $P$ is in coNP.
Intermediate wrap-up: parallel-correctness and transferability

Summary:

- Framework for reasoning about correctness of arbitrary distribution policies
- Intuitive characterization for CQs in terms of minimal valuations
Intermediate wrap-up: parallel-correctness and transferability

Summary:

- Framework for reasoning about correctness of *arbitrary* distribution policies
- Intuitive characterization for CQs in terms of *minimal valuations*

Research opportunities:

- Current setting:
  - no tractable restriction
  - decidability frontier for more expressive query languages
Intermediate wrap-up: parallel-correctness and transferability

Summary:
- Framework for reasoning about correctness of arbitrary distribution policies
- Intuitive characterization for CQs in terms of minimal valuations

Research opportunities:
- Current setting:
  - no tractable restriction
  - decidability frontier for more expressive query languages
- Extended setting:
  - more liberal single-round algorithms
  - multi-round algorithms
Intermediate wrap-up: parallel-correctness and transferability

Summary:

- Framework for reasoning about correctness of *arbitrary* distribution policies
- Intuitive characterization for CQs in terms of *minimal valuations*

Research opportunities:

- Current setting:
  - no tractable restriction
  - decidability frontier for more expressive query languages
- Extended setting:
  - more liberal single-round algorithms
  - multi-round algorithms
- Practical considerations
  - e.g., if not-parallel correct, minimal way to move to a PC one
Outline

1. Introduction

2. Complexity of parallel query processing

3. Reasoning about data partitioning

4. Avoiding coordination
   - CALM-conjecture by example
   - Coordination-free computation
   - CALM-theorems

5. Discussion
Synchronous versus asynchronous

- MPC makes synchronization explicit by modeling computation as a sequence of **rounds** (communication/computation sequence)

Asynchronous computation model:
- computation nodes are allowed to communicate freely
- synchronization can be implicit

Declarative distributed computing:
- approach where distributed computations are programmed using extensions of Datalog

Examples: Webdamlog [Abiteboul et al. 2013], Netlog [Grumbach, Wang, 2010], Dedalus [Hellerstein 2010], NDlog [Loo et al, 2006], ... 

Our focus:
- Which fragments/extensions of Datalog can be asynchronously evaluated in a coordination-free manner?

Setting:
- messages can never be lost but can be arbitrarily delayed.

Requirement:
- output can never be retracted, should eventually be complete
Synchronous versus asynchronous

- MPC makes synchronization explicit by modeling computation as a sequence of **rounds** (communication/computation sequence)
- **Asynchronous** computation model:
  - computation nodes are allowed to communicate freely
  - synchronization can be implicit
Synchronous versus asynchronous

- MPC makes synchronization explicit by modeling computation as a sequence of rounds (communication/computation sequence)
- Asynchronous computation model:
  - computation nodes are allowed to communicate freely
  - synchronization can be implicit
- Declarative distributed computing:
  - approach where distributed computations are programmed using extensions of Datalog
  - Examples: Webdamlog [Abiteboul et al. 2013], Netlog [Grumbach, Wang, 2010], Dedalus [Hellerstein 2010], NDlog [Loo et al, 2006], $D$ [Ma et al, 2013], ...
Synchronous versus asynchronous

- MPC makes synchronization explicit by modeling computation as a sequence of **rounds** (communication/computation sequence)
- **Asynchronous** computation model:
  - computation nodes are allowed to communicate freely
  - synchronization can be implicit
- **Declarative distributed computing:**
  - approach where distributed computations are programmed using extensions of Datalog
  - Examples: Webdamlog [Abiteboul et al. 2013], Netlog [Grumbach, Wang, 2010], Dedalus [Hellerstein 2010], NDlog [Loo et al, 2006], D [Ma et al, 2013], ...

- **Our focus:**
  - Which fragments/extensions of Datalog can be asynchronously evaluated in a coordination-free manner?
  - **Setting:** messages can never be lost but can be arbitrarily delayed.
  - **Requirement:** output can never be retracted, should eventually be complete
Outline

1 Introduction

2 Complexity of parallel query processing

3 Reasoning about data partitioning

4 Avoiding coordination
   - CALM-conjecture by example
     - Coordination-free computation
     - CALM-theorems

5 Discussion
Example

Query: select all triangles

Avoiding coordination

CALM-conjecture by example

PODS June 29, 2016 38 / 62
Example

Query: select all triangles

Avoiding coordination

CALM-conjecture by example

PODS  June 29, 2016  38 / 62
Naive algorithm:

- Gather all data at all nodes
- Output triangles whenever new data arrives

Query: select all triangles
Example

Query: select all triangles

Naive algorithm:
- Gather all data at all nodes
- Output triangles whenever new data arrives

- eventually consistent
- no coordination

Avoiding coordination
CALM-conjecture by example
Example

Query: select all open triangles

Avoiding coordination
CALM-conjecture by example
PODS June 29, 2016 39 / 62
Example

Query: select all *open* triangles
Example

Query: select all open triangles

requires global coordination

Avoiding coordination
CALM-conjecture by example
PODS June 29, 2016 39 / 62
CALM-conjecture [Hellerstein, PODS 2010]

A query has a \textit{coordination-free} and eventually consistent execution strategy iff the query is \textit{monotone}.
CALM-conjecture [Hellerstein, PODS 2010]

A query has a coordination-free and eventually consistent execution strategy iff the query is monotone.

- [Ameloot, N., Van den Bussche, PODS 2011]: TRUE (relational transducer networks)
CALM: Consistency And Logical Monotonicity

CALM-conjecture [Hellerstein, PODS 2010]

A query has a coordination-free and eventually consistent execution strategy iff the query is monotone

- [Ameloot, N., Van den Bussche, PODS 2011]: TRUE (relational transducer networks)
- [Zinn, Green, Ludäscher, ICDT 2012]: FALSE when nodes have more knowledge on how data is distributed (through distribution policies)
CALM-conjecture [Hellerstein, PODS 2010]

A query has a coordination-free and eventually consistent execution strategy iff the query is monotone

- [Ameloot, N., Van den Bussche, PODS 2011]: TRUE (relational transducer networks)
- [Zinn, Green, Ludäscher, ICDT 2012]: FALSE when nodes have more knowledge on how data is distributed (through distribution policies)
- [Ameloot, Ketsman, N., Zinn, PODS 2014]: TRUE when also refining monotonicity
Outline

1. Introduction

2. Complexity of parallel query processing

3. Reasoning about data partitioning

4. Avoiding coordination
   - CALM-conjecture by example
   - Coordination-free computation
   - CALM-theorems

5. Discussion
Computation model: relational transducer networks
[Ameloot, N., Van den Bussche, 2011]

- based on relational transducers
  [Abiteboul, Vianu, Fordham, Yesha, PODS 1998]

Avoiding coordination
Coordination-free computation
PODS June 29, 2016 42 / 62
Computation model: relational transducer networks
[Ameloot, N., Van den Bussche, 2011]

- based on relational transducers
  [Abiteboul, Vianu, Fordham, Yesha, PODS 1998]
- differences with MPC
Computation model: relational transducer networks
[Ameloot, N., Van den Bussche, 2011]

- based on relational transducers
  [Abiteboul, Vianu, Fordham, Yesha, PODS 1998]
- differences with MPC
  - initial start distribution \( H \) (can be modelled by a distribution policy \( P_H \))

- based on relational transducers [Abiteboul, Vianu, Fordham, Yesha, PODS 1998]

- differences with MPC
  - initial start distribution $H$ (can be modelled by a distribution policy $P_H$)
  - every node runs same program $\Pi$
Computation model: relational transducer networks
[Ameloot, N., Van den Bussche, 2011]

- based on relational transducers
  [Abiteboul, Vianu, Fordham, Yesha, PODS 1998]
- differences with MPC
  - initial start distribution $H$ (can be modelled by a distribution policy $P_H$)
  - every node runs same program $\Pi$
  - no rounds, nodes can send messages to each other (to further reshuffle data)

- based on relational transducers [Abiteboul, Vianu, Fordham, Yesha, PODS 1998]
- differences with MPC
  - initial start distribution $H$ (can be modelled by a distribution policy $P_H$)
  - every node runs same program $\Pi$
  - no rounds, nodes can send messages to each other (to further reshuffle data)
  - message can never get lost, but can take arbitrarily long to arrive
Computation model: relational transducer networks
[Ameloot, N., Van den Bussche, 2011]

- based on relational transducers
  [Abiteboul, Vianu, Fordham, Yesha, PODS 1998]

- differences with MPC
  - initial start distribution $H$ (can be modelled by a distribution policy $P_H$)
  - every node runs same program $\Pi$
  - no rounds, nodes can send messages to each other (to further reshuffle data)
  - message can never get lost, but can take arbitrarily long to arrive
  - computes queries: inflationary and eventually consistent semantics
Computing all queries

Observation

Relational transducer networks compute every query
Computing all queries

Observation
Relational transducer networks compute every query

Idea.
- Broadcast all data. Needs coordination (messages can be delayed).
- Locally evaluate query.
Computing all queries

Observation

Relational transducer networks compute every query

Idea.

- Broadcast all data. Needs coordination (messages can be delayed).
- Locally evaluate query.

\[ \text{node} \quad \ldots, (R(\bar{d}_2), x), (R(\bar{d}_1), x) \quad \text{node} \]

\[ x \quad y \]
Computing all queries

Observation
Relational transducer networks compute every query

Idea.
- Broadcast all data. Needs coordination (messages can be delayed).
- Locally evaluate query.
Computing all queries

Observation

Relational transducer networks compute every query

Idea.

- Broadcast all data. Needs coordination (messages can be delayed).
- Locally evaluate query.

Avoiding coordination

Coordination-free computation
Computing all queries

Observation
Relational transducer networks compute every query

Idea.
- Broadcast all data. Needs coordination (messages can be delayed).
- Locally evaluate query.

node node node node node

global coordination

need to know every node in the network
Coordination: intuition

Coordination
- Global consensus
- Reduces parallelism
Coordination: intuition

Coordination

- Global consensus
- Reduces parallelism

Coordination-free

- Embarrassingly parallel execution
- Need communication to transfer data
Coordination: intuition

Coordination
- Global consensus
- Reduces parallelism

Coordination-free
- Embarrassingly parallel execution
- Need communication to transfer data

Goal
Separate data-communication from coordination-communication
Coordination-freeness

Informal Definition

A relational transducer network \( \Pi \) is \textit{coordination-free} if for all input databases \( I \) there is an \textit{ideal} initial distribution \( P_{H,I} \) on which \( \Pi \) computes the query \textit{without reading input messages}.
Informal Definition

A relational transducer network $\Pi$ is coordination-free if for all input databases $I$ there is an ideal initial distribution $P_{H,I}$ on which $\Pi$ computes the query without reading input messages.

Intuition:

- On ideal distribution: communication but no coordination
- On non-ideal distribution: only communication for data transfer, not for coordination
Example of coordination-freeness: output all triangles

Transducer program

- Broadcast local edges
- Output a triangle when one can be derived

Coordination-free:

- Ideal distribution: whole database on one node
- Query is computed by only heartbeat transitions
Outline

1. Introduction
2. Complexity of parallel query processing
3. Reasoning about data partitioning
4. Avoiding coordination
   - CALM-conjecture by example
   - Coordination-free computation
   - CALM-theorems
5. Discussion
Theorem (Ameloot, N., Van den Bussche, PODS 2011)

The following are equivalent. Query $Q$ is

- computable by a coordination-free relational transducer network;
- computable by an oblivious relational transducer network; and, 
- monotone.

**oblivious:** not knowing the identities of all other nodes in the network
The following are equivalent. Query $Q$ is

- computable by a *coordination-free* relational transducer network;
- computable by an *oblivious* relational transducer network; and,
- *monotone*.

**oblivious**: not knowing the identities of all other nodes in the network

**Corollary:**
- coordination-free computations can only define monotone queries
The following are equivalent. Query $Q$ is

- computable by a coordination-free relational transducer network;
- computable by an oblivious relational transducer network; and,
- monotone.

**Corollary:**

- coordination-free computations can only define monotone queries
- Complete language for coordination-free computation: language defining all monotone queries
  - e.g., Datalog with inequalities and value invention [Hull, Yoshikawa, 1990] [Cabibbo, 1998]
Weaker forms of monotonicity

**Definition**

A query $Q$ is **monotone** if $Q(I) \subseteq Q(I \cup J)$ for all instances $I$ and $J$.

**Observation**

If $Q$ is monotone then $Q(I_s) \subseteq Q(I)$ for any $I_s \subseteq I$. 
Weaker forms of monotonicity

Definition
A query $Q$ is domain-distinct-monotone if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances $I$ and $J$ for which $J$ is domain distinct from $I$. 
Weaker forms of monotonicity

**Definition**
A query $Q$ is **domain-distinct-monotone** if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances $I$ and $J$ for which $J$ is domain distinct from $I$.

**Example**
Selecting open triangles in a graph is domain-distinct-monotone.
Weaker forms of monotonicity

Definition
A query $Q$ is domain-distinct-monotone if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances $I$ and $J$ for which $J$ is domain distinct from $I$.

Example
Selecting open triangles in a graph is domain-distinct-monotone.

![Diagram of a graph showing domain-distinct-monotonicity]

I

not domain distinct from I

Q(I)
Weaker forms of monotonicity

**Definition**

A query $Q$ is **domain-distinct-monotone** if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances $I$ and $J$ for which $J$ is domain distinct from $I$.

**Example**

Complement of the transitive closure of a graph is **not** domain-distinct-monotone.

---

**Diagram**

```
(a)  b
I

Q(I)
```

(a)  b

---

**Weaker forms of monotonicity**

**Definition**

A query $Q$ is **domain-distinct-monotone** if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances $I$ and $J$ for which $J$ is domain distinct from $I$.

**Example**

Complement of the transitive closure of a graph is **not** domain-distinct-monotone.
Weaker forms of monotonicity

**Definition**

A query \( Q \) is **domain-distinct-monotone** if

\[
Q(I) \subseteq Q(I \cup J)
\]

for all instances \( I \) and \( J \) for which \( J \) is domain distinct from \( I \).

**Example**

Complement of the transitive closure of a graph is **not** domain-distinct-monotone.
Weaker forms of monotonicity

**Definition**

A query $Q$ is **domain-distinct-monotone** if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances $I$ and $J$ for which $J$ is domain distinct from $I$.

**Observation (preserved under extensions)**

When $Q$ is domain-distinct-monotone, then

$$Q(I_{|C}) \subseteq Q(I)$$

for all subsets of the domain $C$.

Induced subinstance: $I_{|C} = \{ f \in I \mid \text{adom}(f) \subseteq C \}$
The following are equivalent. Query $Q$ is

1. computable by a coordination-free relational transducer network that has access to the initial distribution;
2. domain-distinct-monotone.
**CALM-Theorem (1)**

**Theorem (Ameloot, Ketsman, N., Zinn, PODS 2014)**

The following are equivalent. Query $Q$ is

1. **computable by a coordination-free relational transducer network** that has access to the initial distribution;
2. **domain-distinct-monotone.**

**Crux (2)→(1): (very rough)**

- every transducer has access to the initial distribution:
  - broadcast both occurring and missing facts
- transducer outputs $Q(J|C)$ for every set $C$ on which they have complete knowledge
The following are equivalent. Query Q is

1. computable by a coordination-free relational transducer network that has access to the initial distribution;
2. domain-distinct-monotone.

Corollary:

- Every language defining all domain-disjoint-monotone queries is a complete language for coordination-free computation
- Semi-positive datalog with value invention [Cabibbo, 1998]
Even weaker forms of monotonicity

### Definition

A query $Q$ is **domain-disjoint-monotone** if

\[ Q(I) \subseteq Q(I \cup J) \]

for all instances $I$ and $J$ for which $J$ is domain disjoint from $I$. 

Avoiding coordination
Even weaker forms of monotonicity

**Definition**
A query $Q$ is *domain-disjoint-monotone* if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances $I$ and $J$ for which $J$ is domain disjoint from $I$.

**Example**
Complement of the transitive closure of a graph is domain-disjoint-monotone.
Even weaker forms of monotonicity

**Definition**

A query $Q$ is **domain-disjoint-monotone** if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances $I$ and $J$ for which $J$ is domain disjoint from $I$.

**Observation (preserved under disjoint extension)**

When $Q$ is domain-disjoint-monotone, then

$$Q(C) \subseteq Q(I)$$

for all instances $I$ and **connected components** $C$ of $I$. 
Theorem (Ameloot, Ketsman, N., Zinn, PODS 2014)

The following are equivalent. Query $Q$ is

1. computable by a coordination-free relational transducer network under domain-guided distribution policies; and,
2. domain-disjoint-monotone.
The following are equivalent. Query $Q$ is

1. computable by a coordination-free relational transducer network under domain-guided distribution policies; and,

2. domain-disjoint-monotone.

Corollary:

- Every language defining all domain-disjoint-monotone queries is a complete language for coordination-free computation

- Semi-connected Datalog with value invention [Ameloot, Ketsman, N., Zinn, PODS 2014]
A Datalog rule is **connected** when the graph formed by the positive atoms is connected.

\[
\varphi \equiv O(x, y, z) \leftarrow E(x, y), E(y, z), E(z, x) \quad \text{is connected}
\]

\[
\text{graph}^+(\varphi)
\]
Connected Datalog rule

Example

\[ O(x, y, z) \leftarrow E(x, y), E(z, u) \] is not connected

\[ O(x, y, z) \leftarrow E(x, y), \neg E(y, z) \] is not connected
Semi-connected Datalog

Definition
A Datalog program is semi-connected if all rules are connected except (possibly) those of the last stratum.

Example

\[
\begin{align*}
TC(x, y) & \leftarrow E(x, y) \\
TC(x, y) & \leftarrow E(x, z), TC(z, y) \\
O(x, y) & \leftarrow \neg TC(x, y), x \neq y
\end{align*}
\]

- Semi-positive Datalog is strictly included in semi-connected Datalog
- Semi-connected Datalog defines only domain-distinct-monetone queries
Connected Datalog

**Win-move:** \( \text{win}(x) \leftarrow \text{move}(x, y), \neg \text{win}(y) \)
**Win-move:** \( \text{win}(x) \leftarrow \text{move}(x, y), \neg \text{win}(y) \)

**Theorem (Zinn, Green, Ludäscher, ICDT 2012)**

The query computing the won positions of the win-move game is in \( \mathcal{F}_2 \).
Connected Datalog

**Win-move:** \( \text{win}(x) \leftarrow \text{move}(x, y), \neg \text{win}(y) \)

---

**Theorem (Zinn, Green, Ludäscher, ICDT 2012)**

*The query computing the won positions of the win-move game is in \( F_2 \)*

**Proof.** Simulation in semi-monotone fragment of Datalog\( \neg
\neg \) with a novel disorderly semantics [Abiteboul, Vianu, 1991].
Connected Datalog

**Win-move:** \( \text{win}(x) \leftarrow \text{move}(x, y), \neg \text{win}(y) \)

---

**Theorem (Zinn, Green, Ludäscher, ICDT 2012)**

The query computing the won positions of the win-move game is in \( F_2 \)

**Proof.** Simulation in semi-monotone fragment of Datalog\( _{\neg \neg} \) with a novel disorderly semantics [Abiteboul, Vianu, 1991].

---

**Theorem (Ameloot, Ketsman, N., Zinn, ICDT 2015)**

Queries expressed in connected Datalog under the well-founded semantics are in \( F_2 \).
CALM overview

\[
\begin{align*}
\text{Datalog}(\neq) & \subset \text{wILOG}(\neq) = \mathcal{M} = \mathcal{F}_0 = \mathcal{A}_0 \\
\cap & \cap \\
\text{SP-Datalog} & \subset \text{SP-wILOG} = \mathcal{M}_{\text{distinct}} = \mathcal{F}_1 = \mathcal{A}_1 \\
\cap & \cap \\
\text{semicon-Datalog} & \subset \text{semicon-wILOG} = \mathcal{M}_{\text{disjoint}} = \mathcal{F}_2 = \mathcal{A}_2
\end{align*}
\]
Intermediate wrap-up

Summary:

- CALM-theorems link Datalog, monotonicity, and coordination-freeness on different levels
- Connected Datalog
Intermediate wrap-up

Summary:
- CALM-theorems link Datalog, monotonicity, and coordination-freeness on different levels
- Connected Datalog

Research issues:
- Broadcast algorithms are not efficient (initial steps in [Ketsman, N., ICDT 2015])
- Quantify coordination [Alvaro et al., CIDR 2011, ICDE 2014]
- General theory of asynchronous query evaluation to capture efficient evaluation
Outline

1. Introduction
2. Complexity of parallel query processing
3. Reasoning about data partitioning
4. Avoiding coordination
5. Discussion
Discussion

- Overview
  - MPC & Shares/HyperCube
  - Parallel-correctness and transferability

- Coordination-freeness and degrees of monotonicity

- Fascinating research area:
  - exciting new machinery
    - e.g., optimal fractional node/edge coverings
  - old wine in new bottles
    - e.g., decision problems relating to CQs
    - e.g., Datalog and monotonicity
Thank you for your attention

Questions?