Massively Parallel Communication and Query Evaluation

Paul Beame
U. of Washington

Based on joint work with Paraschos Koutris and Dan Suciu
[PODS 13], [PODS 14]
Massively Parallel Systems
MapReduce

[Dean,Ghemawat 2004]  Rounds of

**Map**: Local and data parallel on \((key, value)\) pairs creating \((key_1, value_1)\)... \((key_k, value_k)\) pairs

**Shuffle**: Groups or sorts \((key, value)\) pairs by key
  • Local sorting plus global communication round

**Reduce**: Local and data parallel on keys:
  \((key, value_1)\)... \((key, value_k)\) reduces to \((key, value)\)

  – Data fits jointly in main memory of 100’s/1000’s of parallel servers each with gigabyte+ storage
  – Fault tolerance
What can we do with MapReduce? Models & Algorithms

• **Massive Unordered Distributed Data (MUD) model** [Feldman-Muthukrishnan-Sidiropoulos-Stein-Svitkina 2008]
  – 1 round can simulate data streams on symmetric functions, using Savitch-like small space simulation
  – Exact computation of frequency moments in 2 rounds of MapReduce

• **MRC model** [Karloff, Suri, Vassilvitskii 2010]
  – For $n^{1-\epsilon}$ processors and $n^{1-\epsilon}$ storage per processor, $O(t)$ rounds can simulate $t$ PRAM steps so $O(\log^k n)$ rounds can simulate $NC^k$
  – Minimum spanning trees and connectivity *on dense graphs* in 2 rounds of MapReduce
  – Generalization of parameters, sharper simulations, sorting and computational geometry applications [Goodrich, Sitchinava, Zhang 2011]
What can we do with MapReduce?
Models & Algorithms

• Communication-processor tradeoffs for 1 round of MapReduce
  – Upper bounds for database join queries [Afrati, Ullman 2010]
  – Upper and lower bounds for finding triangles, matrix multiplication, finding neighboring strings [Afrati, Sarma, Salihoglu, Ullman 2012]
More than just MapReduce

What can we do with this?

Are there limits? Lower bounds?
A simple general model?
MapReduce

[Dean,Ghemawat 2004] Rounds of

Map: Local and data parallel on \((key, value)\) pairs
creating \((key_1, value_1), \ldots, (key_k, value_k)\) pairs

Shuffle: Groups or sorts \((key, value)\) pairs by key
  • Local sorting plus global communication round

Reduce: Local and data parallel on \(keys\):
  \((key,value_1), \ldots, (key,value_m)\) reduces to \((key,value)\)

  • Data fits jointly in main memory of 100’s/1000’s of parallel servers each with gigabyte+ storage
  • Fault tolerance

\(\Omega(n \log n)\) time

unspecified time

essential for efficiency
Properties of a Simple General Model of Massively Parallel Computation

- Organized in synchronous rounds
- Local computation costs per round should be considered free, or nearly so
  - No reason to assume that sorting is special compared to other operations
- Memory per processor is the fundamental constraint
  - This also limits # of bits a processor can send or receive in a single round
Bulk Synchronous Parallel Model

[Valiant 1990] Local computations separated by global synchronization barriers

• Key notion: An \( h \)-relation, in which each processor sends and receives at most \( h \) bits

• Parameters:
  – periodicity \( L \) : time interval between synchronization barriers
  – bandwidth \( g \) : \( \max_{h \geq h_0} \frac{\text{time to deliver an } h \text{-relation}}{h} \)
Massively Parallel Communication (MPC) Model

- Total size of the input = $N$
- Number of processors = $p$
- Each processor has:
  - Unlimited computation power
  - $L \geq N/p$ bits of memory
- A round/step consists of:
  - Local computation
  - Global communication of an $L$-relation
    - i.e., each processor sends/receives $\leq L$ bits
    - $L$ stands for the communication/memory load
MPC model continued

• Wlog $\frac{N}{p} \leq L \leq N$
  – any processor with access to the whole input can compute any function

• Communication
  – processors pay *individually for receiving* the $L$ bits per round, total communication cost up to $pL \geq N$ per round.

• Input distributed uniformly
  – Adversarially or random input distribution also

• Access to random bits (possibly shared)
Relation to other communication models

• Message-passing (private messages) model
  – each costs per processor receiving it
  – wlog one player is a Coordinator who sends and receives every message
    • Many recent results improving $\Omega(N/p)$ lower bounds to $\Omega(N)$
      [WZ 12], [PVZ12], WZ13], [BEOPV13],...
    • Complexity is never larger than $N$ bits independent of rounds

  – No limit on bits per processor, unlike MPC model

• \textit{CONGEST} model
  – Total communication bounds $> N$ possible but depends on network diameter and topology
    • MPC corresponds to a complete graph for which largest communication bound possible is $\leq N$
Complexity in the MPC model

• Tradeoffs between rounds $r$, processors $p$, and load $L$

• Try to minimize load $L$ for each fixed $r$ and $p$
  – Since $N/p \leq L \leq N$, the range of variation in $L$ is a factor $p^\varepsilon$ for $0 \leq \varepsilon \leq 1$

• 1 round
  – still interesting theoretical/practical questions
  – many open questions

• Multi-round computation more difficult
  – e.g. PointerJumping, i.e., st-connectivity in out-degree 1 graphs.
    • Can achieve load $O(N/p)$ in $r=O(\log_2 p)$ rounds by pointer doubling
Database Join Queries

- Given input relations $R_1, R_2, \ldots, R_m$ as tables of tuples, of possibly different arities, produce the table of all tuples answering the query

$$Q(x_1, x_2, \ldots, x_k) = R_1(x_1, x_2, x_3), R_2(x_2, x_4), \ldots, R_m(x_4, x_k)$$

  - Known as full conjunctive queries since every variable in the RHS appears in the query (no variables projected out)

- Our examples: Connected queries only
The Query Hypergraph

- One vertex per variable
- One hyper-edge per relation

\[ Q(x_1, x_2, x_3, x_4, x_5) = R(x_1, x_2, x_3), S(x_2, x_4), T(x_3, x_5), U(x_4, x_5) \]
k-partite data graph/hypergraph

Query Hypergraph

Data Hypergraph

$n$ possible values per variable

$nk$ vertices total
k-partite data graph/hypergraph

Query Hypergraph

Data Hypergraph

$n$ possible values per variable

$nk$ vertices total
Some Hard Inputs

• **Matching Databases**
  – Number of relations $R_1, R_2, \ldots$ and size of query is constant
  – Each $R_j$ is a perfect $a_j$-dimensional *matching* on $[n]^{a_j}$ where $a_j$ is the arity of $R_j$
    • i.e. among all the $a_j$-tuples $(k_1, \ldots, k_{a_j}) \in R_j$, each value $k \in [n]$ appears exactly once in each coordinate.
    • No skew (all degrees are the same)
    • Number of output tuples is at most $n$

  – Total input size is $N = O(\log(n!)) = O(n \log n)$
Example in two steps

Find all triangles

\[ C_3(x, y, z) = R_1(x, y), R_2(y, z), R_3(z, x) \]

Algorithm 1:

For each server \( 1 \leq u \leq p \):

**Input:** \( n/p \) tuples from each of \( R_1, R_2, R_3 \)

**Step 1:**
- Send \( R_1(x, y) \) to server \((y \mod p)\)
- Send \( R_2(y, z) \) to server \((y \mod p)\)

**Step 2:**
- Join \( R_1(x, y) \) with \( R_2(y, z) \)
- Send \([R_1(x, y), R_2(y, z)]\) to server \((z \mod p)\)
- Send \( R_3(z, x) \) to server \((z \mod p)\)

**Output**
- Join \([R_1(x, y), R_2(y, z)]\) with \( R_3(z, x') \)
- Output all triangles \( R_1(x, y), R_2(y, z), R_3(z, x) \)

Load: \( O(n/p) \) tuples (i.e. \( \epsilon=0 \))

Number of rounds: \( r = 2 \)
Example in one step

Find all triangles
\[ C_3(x,y,z) = R_1(x,y), R_2(y,z), R_3(z,x) \]

Algorithm 2:
Servers form a cube: \[ [p] \equiv [p^{1/3}] \times [p^{1/3}] \times [p^{1/3}] \]

For each server \( 1 \leq u \leq p \):

Step 1: Choose random hash functions \( h_1, h_2, h_3 \)
send \( R_1(x,y) \) to servers \( (h_1(x) \mod p^{1/3}, h_2(y) \mod p^{1/3}, *) \)
send \( R_2(y,z) \) to servers \( (*, h_2(y) \mod p^{1/3}, h_3(z) \mod p^{1/3}) \)
send \( R_3(z,x) \) to servers \( (h_1(x) \mod p^{1/3}, *, h_3(z) \mod p^{1/3}) \)

Output all triangles \( R_1(x,y), R_2(y,z), R_3(z,x) \)

Load: \( O(n/p \times p^{1/3}) \) tuples \( (\epsilon = 1/3) \)
Number of rounds: \( r = 1 \)
We Show

Find all triangles
\[ C_3(x,y,z) = R_1(x,y), \ R_2(y,z), \ R_3(z,x) \]

Load: \( O(n/p \times p^{1/3}) \) tuples \( (\varepsilon = 1/3) \)
Number of rounds: \( r = 1 \)

Above algorithm is optimal for any randomized 1 round MPC algorithm for the triangle query

Based on general characterization of queries based on the fractional cover number of their associated hypergraph
Fractional Cover Number $\tau^*$

**Vertex Cover LP:**

$\tau^* = \min \sum v_i$

Subject to:

$\sum_{x_i \in \text{vars}(R_j)} v_i \geq 1 \ \forall j$

$v_i \geq 0 \ \forall i$

**Edge Packing LP:**

$\tau^* = \max \sum u_j$

Subject to:

$\sum_{x_i \in \text{vars}(R_j)} u_j \leq 1 \ \forall i$

$u_j \geq 0 \ \forall j$

$\tau^*(C_k) = k/2$

$\tau^*(L_k) = \left\lfloor k/2 \right\rfloor$
1-Round No Skew

Theorem: Any 1-round randomized MPC algorithm with $p = \omega(1)$ and load $o(N/p^{1/\tau^*(Q)})$ will fail to compute connected query $Q$ on some matching database input with probability $\Omega(1)$.

$\tau^*(C_3) = 3/2$ so need $\Omega(N/p^{2/3})$ load, i.e. $\varepsilon \geq p^{1/3}$ for $C_3$

... previous 1-round algorithm is optimal

Can get a matching upper bound this for all databases without skew by setting parameters in randomized algorithm generalizing the triangle case

• exponentially small failure probability
HyperCube/Shares Algorithm

**Vertex Cover LP:**

\[ \tau^* = \min \sum_i v_i \]

Subject to:

\[ \sum_{x_i \in \text{vars}(R_j)} v_i \geq 1 \quad \forall j \]

\[ v_i \geq 0 \quad \forall i \]

**Algorithm:** Decompose \( p = (p_{v1}/\tau^*,..., p_{vm}/\tau^*) \) and hash each tuple as in triangle case
1-Round No Skew

**Theorem:** Any 1-round randomized MPC algorithm with $p = \omega(1)$ and load $o(N/p^{1/\tau^*(Q)})$ will fail to compute connected query $Q$ on some matching database input with probability $\Omega(1)$.

Follows from...

**Lemma:** For any deterministic 1-round MPC algorithm, any processor that receives $O(N/p^\delta)$ bits about each input relation learns only $O(E[|Q(I)|]/p^{\tau^*(Q) \delta})$ correct answers to connected query $Q$ on average for a random matching database input $I$. 
Communication Complexity

Consequence

Whenever $\tau^*(Q) > 1$, solving $Q$ with $p$ processors in one round requires $\omega(N/p)$ bits received per processor where $N = \# \text{ input bits}$.

Lower bound implies failure even when total communication is $\omega(N)$
Slightly Stronger Lower Bound Model: MPC with Relation Servers

**Input:** each relation $R_1, R_2, \ldots, R_k$ is stored on a separate input server.

**Step 1:** each input server $i$ distributes $R_i$ to the $p$ processors

**Steps 2, 3, ...:** the $p$ processors perform the computation as before
Information for a Single Processor

Prop: If $|\text{msg}|$ is $O(N/p^\delta)$ then $E_1[|\mathbf{K}_{\text{msg}(R_j(l))}(R_j)|]$ is $O(n/p^\delta)$

$\mathbf{K}_{\text{msg}}(Q)$ = set of query answers known by this processor

$\mathbf{K}_{\text{msg}}(Q) = \mathbf{K}_{\text{msg}_1}(R_1) \bowtie \ldots \bowtie \mathbf{K}_{\text{msg}_k}(R_k)$
Finishing off 1-Round Lower Bound

**Lemma:** In 1 round, for a processor that receives $O(N/p^\delta)$ bits about each input relation,

$$E_I[|K_{msg(I)}(Q)|] \text{ is } O(E[|Q(I)|]/p \tau^*(Q) \delta).$$

Proof Ideas:

- Apply an inequality due to Friedgut to bound $E_I[|K_{msg(I)}(Q)|]$ in terms of $E_I[|K_{msg(R_j(I))}(R_j)|]$.

- Friedgut’s inequality uses a *fractional edge cover* of $Q$.
  - Relate the fractional edge cover and the fractional edge packing number $\tau^*(Q)$ to obtain the bound.
Friedgut’s Inequality

Given:
(QUERY) hypergraph $Q$ with hyper-edges $S_j \subseteq [k]$ for all $j \in [\ell]$
For all $a \in [n]^k$ write $a_j$ for the projection of $a$ on coordinates of $S_j$
Variables $w_j(a_j)$ for each $a_j \in [n]^{S_j}$

Then for any fractional edge cover $u = (u_1, u_2, ..., u_\ell)$ of $Q$

$$\sum_{a \in [n]^k} \prod_{j=1}^{\ell} w_j(a_j) \leq \prod_{j=1}^{\ell} \left( \sum_{a_j \in [n]^k} w_j(a_j)^{\frac{1}{u_j}} \right)^{u_j}$$

Apply with $w_j(a_j) = \text{Probability (over the input distribution) that processor learns that } a_j \in R_j$
LHS = Expected number of answer tuples processor learns
RHS = Product of independent quantities based on what a processor learns about each relation
Dealing with Skew
(Irregular Hypergraphs)

- Suppose that in computing a join $R_1(x,y)R_2(y,z)$ there is value $v$ of attribute $y$ that has very high degree in relation $R_1$

- Then information about $R_1$ tuples containing $v$ must be spread out to multiple processors or there will be a hot spot of heavy load.

- However, without side information, the server for relation $R_2$ will not know about this plan and will not know to replicate information about its tuples containing $v$. 
Dealing with Skew
(Irregular Hypergraphs)

[BKoutrisSuciu 2014]

• Can quantify losses due to these hot spots and lack of coordination

• An alternative: Augment the 1-round MPC
  – Servers share identities and approximate degrees of “heavy hitter” vertices in any relation
    • those of degree \( \geq \frac{m_j}{p} \) where relation \( j \) has \( m_j \) tuples
    • \( O(p) \) such vertices in total
    • may be found by random sampling
  
  – Use this information to split up the processors into blocks and apply separate HyperCube algorithms on each block
Dealing with Skew
(Irregular Hypergraphs)

• With “heavy hitter” info can achieve, e.g.
  – For simple join with relation sizes $m_1, m_2$
    • Load $L = \max \left( \frac{m_1}{p}, \frac{m_2}{p}, \sqrt{\frac{m_1 m_2}{p}} \right)$
  – For triangle join with relation sizes $m_1, m_2, m_3$
    Load $L = \max \left( \frac{m_1}{p}, \frac{m_2}{p}, \frac{m_3}{p}, \sqrt{\frac{m_1 m_2}{p}}, \sqrt{\frac{m_2 m_3}{p}}, \sqrt{\frac{m_1 m_3}{p}}, \frac{(m_1 m_2 m_3)^{\frac{1}{3}}}{p^{\frac{2}{3}}} \right)$

• Algorithms can be slowed down to work as sequential external memory algorithms and match known bounds
• Many cases remain open
Lower Bounds for Multiple Round MPC: A Circuit Complexity Barrier

In each round, each processor receives $L$ bits of input and produces $L$ bits of output so...

MPC in $r$ rounds can simulate any circuit of depth $r$ with $p$ gates per level each of which computes an arbitrary function $g: \{0, 1\}^L \rightarrow \{0, 1\}^L$.

Just restricting the power of the processors doesn’t do much to help avoid a similar circuit complexity barrier.
Lower Bounds for Multiple Round MPC: Structured Model Required

• Problem: Messages are arbitrary bits that can give complex partial information

• Possible solution: restrict messages to database tuples

• Not enough: Set of tuples sent from A to B becomes their common knowledge after 1st round. Can be re-sent in later rounds to code arbitrary bits.

• Message routing also needs to be restricted
Lower Bounds for Multiple Rounds: Tuple-based MPC

- Only (join) tuples\(^1\) sent
- Routing based only on round #, tuple\(^2\), & step 1 messages from associated relations

\(^1\) Join tuple: \([R_1(a,b)R_2(b,c)]\)
\(^2\) Or known join tuple containing it e.g. can send \(R_1(a,b)\) using \(R_2(b,c)\)
Algorithm for Multiple Rounds
No Skew

Let \( k_\delta = \lceil 2\delta \rceil \).

**Fact:** \( k_\delta = \max \{ k : \text{path query } L_k \text{ doable in 1 round with at most } N/p^\delta \text{ load} \} \)

**Theorem:** For connected \( Q \) there is a **simple** tuple-based MPC algorithm with \( O(N/p^\delta) \) load on matching DBs using
\[
r = \left\lceil \log \text{radius}(Q) / \log k_\delta \right\rceil + 1 \text{ rounds}
\]

Idea: can effectively shrink paths by a factor \( k_\delta \) per round.

\[\text{radius}(Q) = \min_u \max_v d(u,v) \text{ where } d(u,v) = \# \text{ of edge hops}\]
Lower Bound for Multiple Round Tuple-MPC Computing Tree-like Queries

**Theorem:** For *tree-like* Q any tuple-based MPC algorithm with $O(N/p^\delta)$ load requires $r \geq \log \text{diameter}(Q) / \log k_\delta$ rounds on matching databases.

**Proof ideas:**
Inductively eliminate 2\textsuperscript{nd} round until algorithm reduced to 1 round:
- In 2\textsuperscript{nd} round can only send (join) tuples learned in 1\textsuperscript{st} round
- By 1-round analysis, only a tiny number of join tuples learned in 1\textsuperscript{st} round that are larger than $k_\delta$ - give away full extension of each to an answer for Q (reduces n)
- **Shrink** resulting graph so every join tuple of diameter $k_\delta$ has only one edge remaining. (Eliminates all joins from 1\textsuperscript{st} round)
- All edges sent in 2\textsuperscript{nd} round could have been sent in round 1
Shrinking the query graph
Shrinking the query graph
Shrinking the query graph
Shrinking the query graph
Corollary: Any tuple-based MPC algorithm that finds st-paths in degree 2 undirected graphs with $O(N/p^\delta)$ load for $\delta > 0$ requires $\Omega(\log p)$ rounds.

Idea: reduction from line query $L_k$ with $k = p^\beta$ for some small $\delta > 0$
Open Problems

• Lower bounds for decision or counting problems?
  – All lower bounds known depend on the need to produce multiple outputs

• Handle skew efficiently in multi-round algorithms?
  – Intermediate results may be very large even when few answers

• Lower bounds in the full MPC model for 2 rounds?
  – Some depth 2 circuit lower bounds are known for arbitrary gates

• Multi-round algorithm lower bounds for non-tree examples like multi-round algorithms for examples like $C_5$

• Bounds for other kinds of problems in the MPC model
Thank you!
## Fractional Cover Number $\tau^*$

<table>
<thead>
<tr>
<th>Vertex Cover LP:</th>
<th>Edge Packing LP:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau^* = \min \sum_i v_i$</td>
<td>$\tau^* = \max \sum_j u_j$</td>
</tr>
<tr>
<td>Subject to: $\sum_{x_i \in \text{vars}(R_j)} v_i \geq 1 \ \forall j$</td>
<td>Subject to: $\sum_{x_i \in \text{vars}(R_j)} u_j \leq 1 \ \forall i$</td>
</tr>
<tr>
<td>$v_i \geq 0 \ \forall i$</td>
<td>$u_j \geq 0 \ \forall j$</td>
</tr>
</tbody>
</table>
**Tight Edge Cover LP**

**Edge Packing LP:**

\[
\tau^* = \max \sum_j u_j
\]

Subject to:

\[
\sum_{x_i \in \text{vars}(R_j)} u_j \leq 1 \quad \forall i
\]

\[
u_j \geq 0 \quad \forall j
\]

**Tight Edge Cover:**

Constraints:

\[
\sum_{x_i \in \text{vars}(R_j)} u_j + u_i^* = 1 \quad \forall i
\]

\[
u_j, u_i^* \geq 0 \quad \forall j \forall i
\]

**Lower Bound:** Apply Friedgut’s Inequality using Tight Edge Cover solution. The slack variables \( u_i^* \) correspond to weights on new unary edges that don’t affect probabilities.