HoneyComb
A Parallel Worst-Case Optimal Join on Multicores

Jiacheng Wu, Dan Suciu

June 2025

SIGMOD Conference

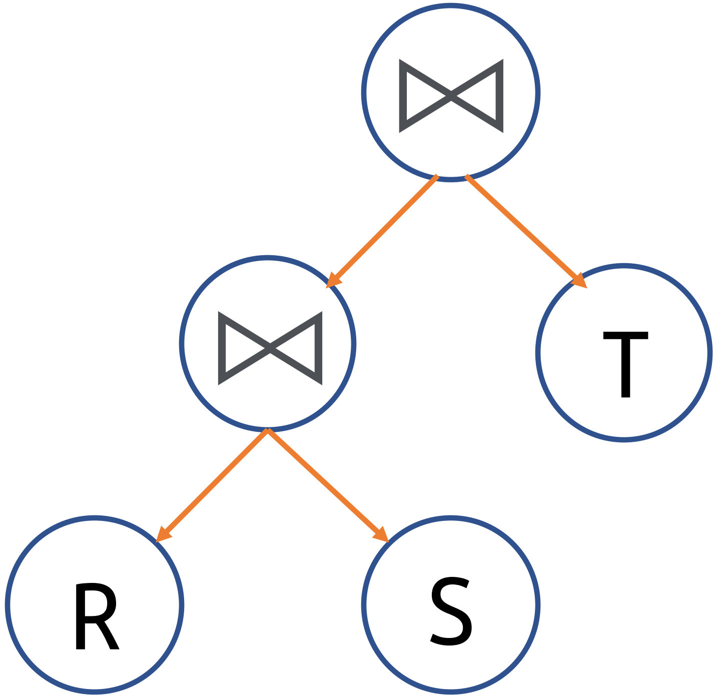
Introduction

What is Worst-Case Optimal Join

- Triangle Query: Q(X, Y, Z) <- R(X, Y), S(Y, Z), T(X, Z)

- Binary Join Plan
  - Large Intermediate Results

- Worst Case Optimal Joins
  - NO large intermediate results
  - Theoretical Guarantees



Q(X, Y, Z) <- R(X, Y), S(Y, Z), T(X, Z)

WCOJ – Generic Join Implementation

- Triangle Query:
  - $|R|=|S|=|T|=N$

```
For x ∈ R.X ∩ T.X
  For y ∈ R[x].Y ∩ S.Y
    For z ∈ S[y].Z ∩ T[x].Z
      Q += (x, y, z)
```

- Generic Join (WCOJ)
  - Complexity: $O(N^{1.5})$
  - = worst size of results

- But how can GJ be parallelized?

$Q(X, Y, Z) \leftarrow R(X, Y), S(Y, Z), T(X, Z)$

Traditional Parallelization

- Partition on Top Level
  - $X = X_1 \cup X_2 \ldots$
  - $R_i = \{(x, y) \in R \mid x \in X_i\}$
  - $T_i = \{(x, z) \in T \mid x \in X_i\}$
- Umbra, LogicBlox
  - Lazy Hash Trie Building
    - Build subtries when needed
    - Skip not used subtries
  - Work Stealing Framework

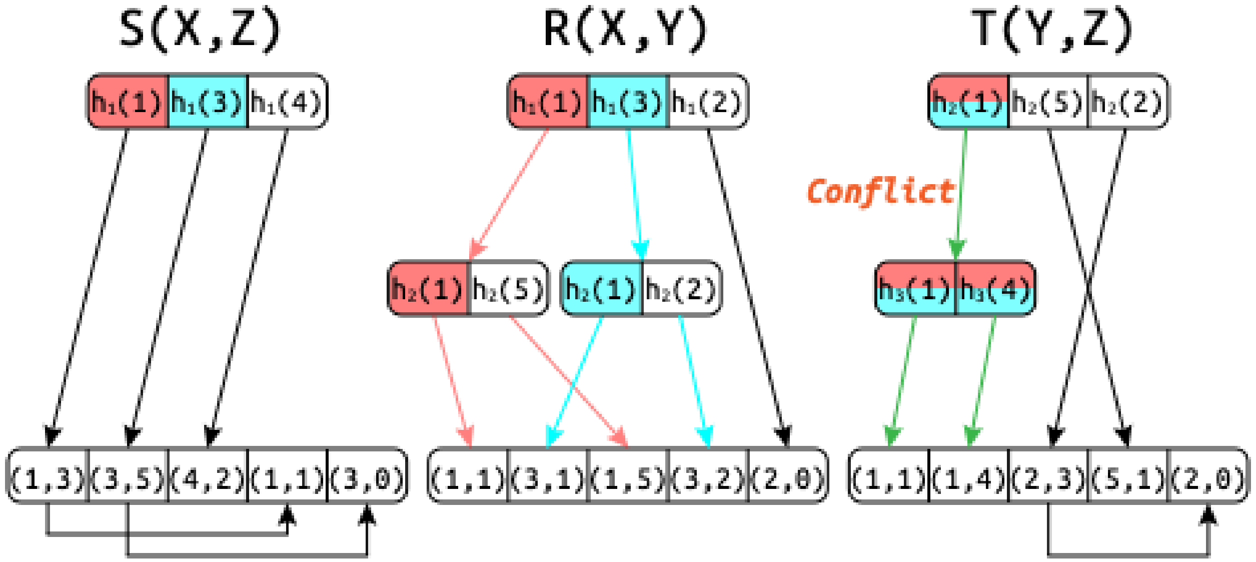
```
// Parititon R.X and T.X on X
// In parallel, Thread i:
For x ∈ Ri.X ∩ Ti.X
  For y ∈ Ri[x].Y ∩ S.Y
    For z ∈ S[y].Z ∩ Ti[x].Z
      Q += (x, y, z)
```

[Freitag'2020]

Q(X, Y, Z) <- R(X, Y), S(Y, Z), T(X, Z)

Traditional Parallelization: Issues

- On Sparse Data
  - Sensitive to Skew...
  - ...in Input and Output
- On Dense Data
  - Lazy index: increased cost
  - Read or Write Conflicts
  - Not Hardware Friendly



Our Methods

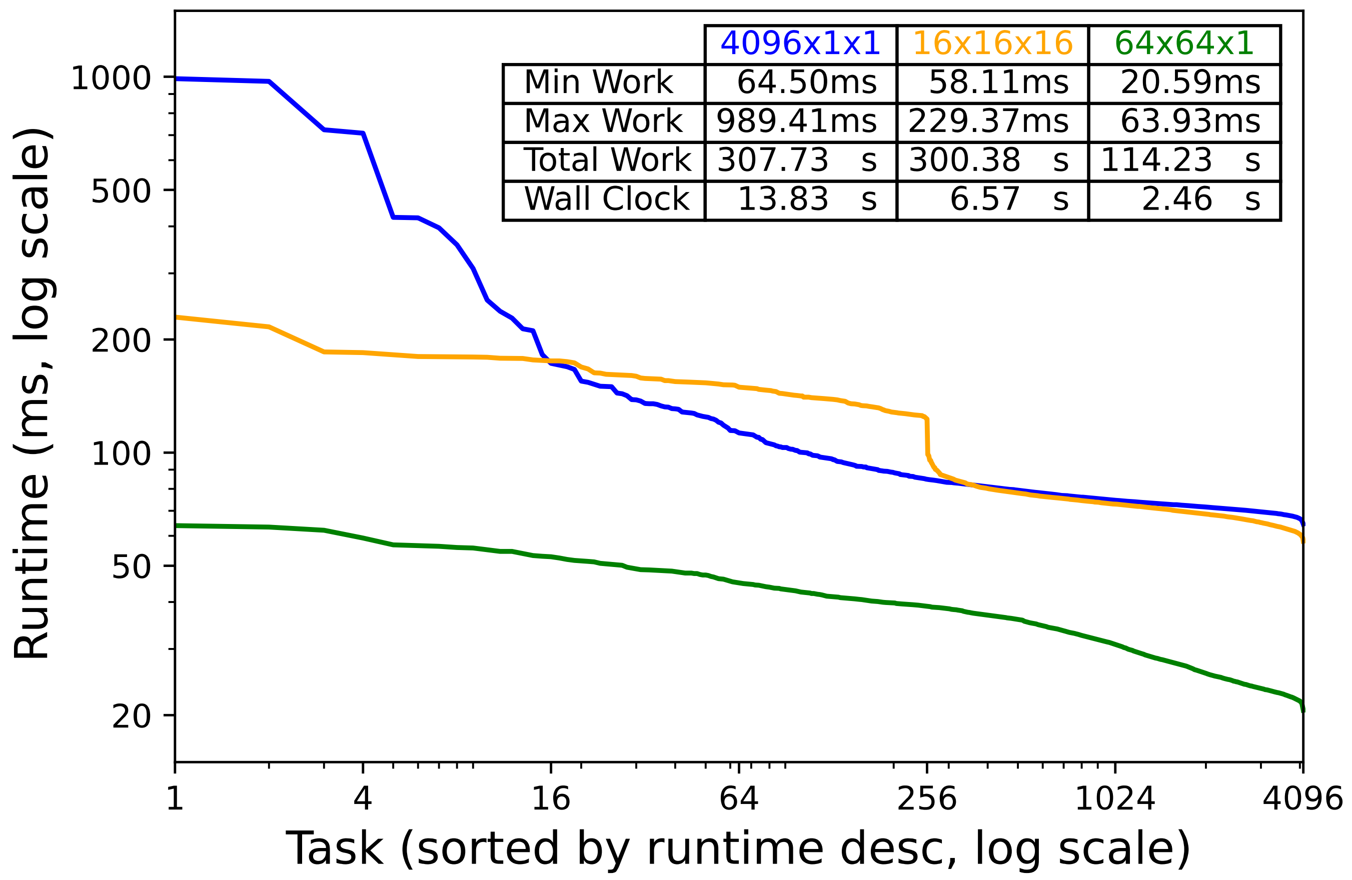
- Partition the Domains
  - $X = X_1 \cup X_2 \cup \ldots \cup X_I$
  - $Y = Y_1 \cup Y_2 \cup \ldots \cup Y_J$
  - $Z = Z_1 \cup Z_2 \cup \ldots \cup Z_K$
- Partition the Relations
  - $R_{ij} = R \cap (X_i \times Y_j)$
    $= \{(x,y) \in R \mid a \in X_i \ \& \ b \in Y_j\}$
  - $S_{jk} = S \cap (Y_j \times Z_k)$
  - $T_{ik} = T \cap (X_i \times Z_k)$
- Task (i, j, k)
  - Access split relations (RO)
  - Join Independently

```
// Partition R, S and T (see text)
// In parallel, Thread (i, j, k)
For x ∈ Rij.X ∩ Tik.X
  For y ∈ Rij[x].Y ∩ Sjk.Y
    For z ∈ Sjk[y].Z ∩ Tik[x].Z
      Q += (x, y, z)
```

Q(X, Y, Z) <- R(X, Y), S(Y, Z), T(X, Z)

Skew of the Wall-clock Time per Task

- Triangle on Orkut
- 4096x1x1
  - 4096 parts on X
  - Traditional Para WCOJ
- 16x16x16
  - 16 parts on X, Y, Z
  - Hypercube Optimal
- 64x64x1
  - 64 parts on X, Y
  - Our WCOJ

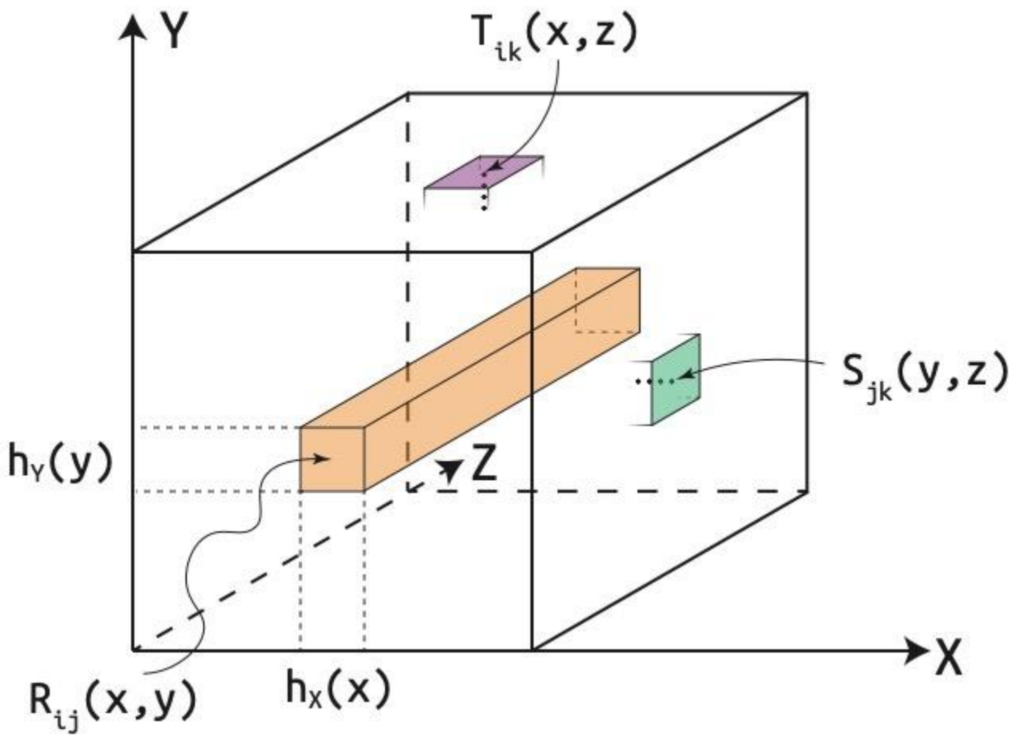


Background

Q(X, Y, Z) <- R(X, Y), S(Y, Z), T(X, Z)

HyperCube: Distributed

- Partition attributes
  - Split X|Y|Z into I|J|K slices
- **Node** is assigned label
  - (i,j,k) i∈[I], j∈[J], k∈[K]
- R(X, Y) is split to I*J parts
  - (x,y) is **sent** to $R_{ij}(X, Y)$ when $h_X(x)\%I=i$ and $h_Y(y)\%J=j$
  - $R_{ij}(X, Y)$ is **broadcast** to nodes (i,j,*) for any *∈[K]
- S(Y, Z), T(X, Z): similarly

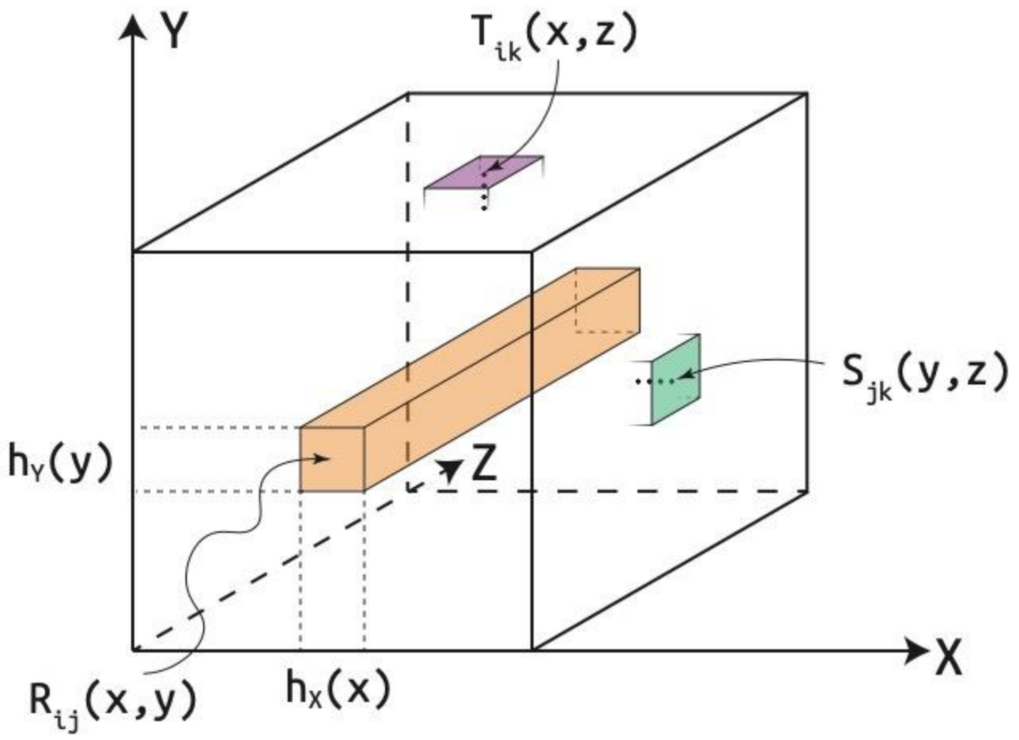


[N] on its own means set {1,2,…,N}

Q(X, Y, Z) <- R(X, Y), S(Y, Z), T(X, Z)

HyperCube: Shared memory (Ours)

- Partition attributed
  - Split X|Y|Z into I|J|K slices
- **Task** is assigned label
  - (i,j,k) i∈[I], j∈[J], k∈[K]
- Partition R into I*J parts
  - (x,y) **belongs to** $R_{ij}(X, Y)$ when $h_X(x)\%I=i$ and $h_Y(y)\%J=j$
  - $R_{ij}(X, Y)$ is **accessed** by tasks (i,j,*) with any *∈[K]
- S(Y, Z), T(X, Z): similarly



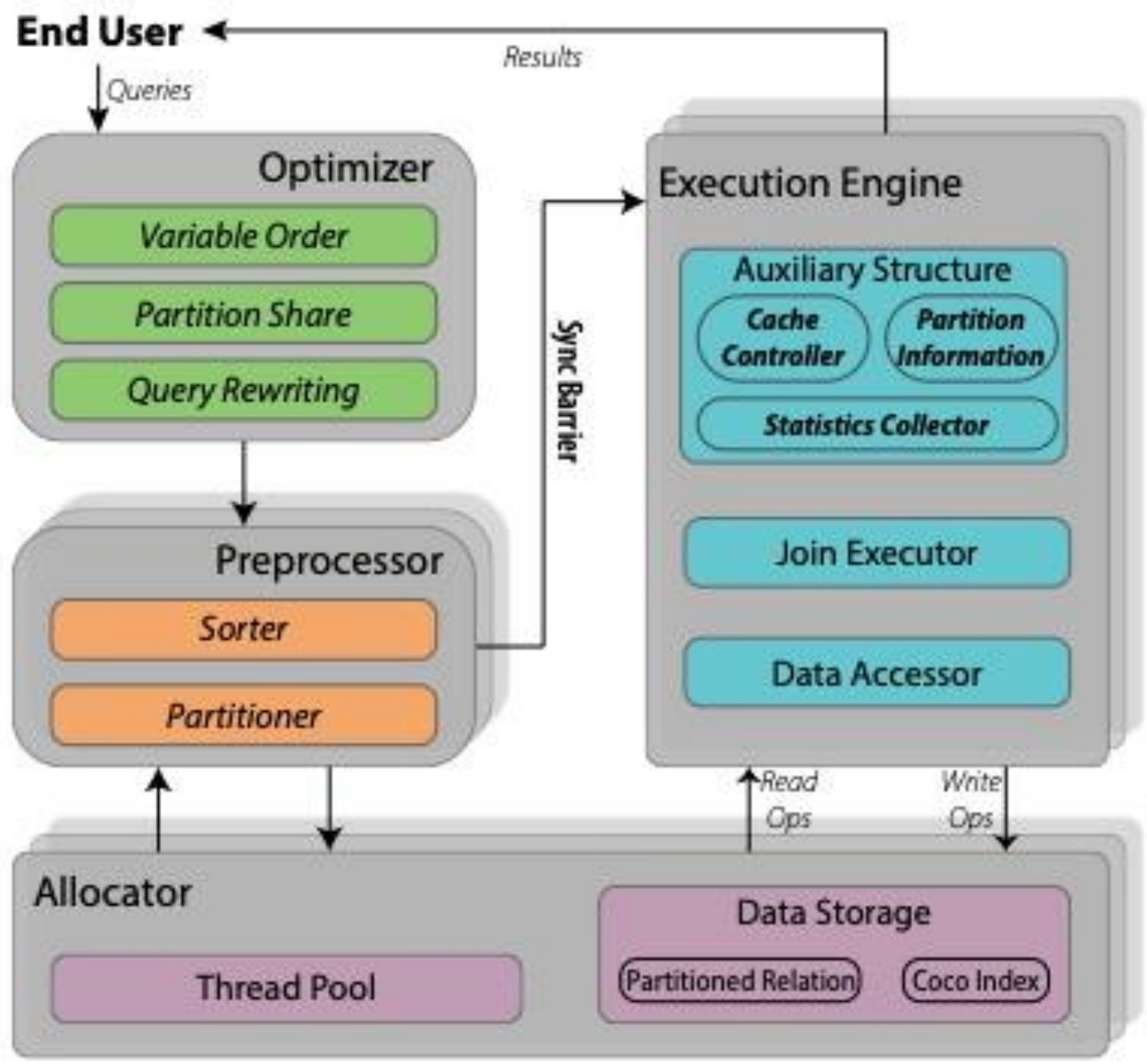
[N] on its own means set {1,2,…,N}

HoneyComb

Architecture

- Four Components
  - Allocator
  - Optimizer
  - Preprocessor
  - Executor

- Two Stages
  - Preprocessing Stage
  - Join Stage



[Axtmann'2017]

Preprocessor: Partition

- $R_{ij}(X, Y) = \{(x, y) \in R \mid h_X(x) = i,\ h_Y(y) = j\}$
- $S_{jk}(Y, Z) = \{(y, z) \in S \mid h_Y(y) = j,\ h_Z(z) = k\}$
- $T_{ik}(X, Z) = \{(x, z) \in T \mid h_X(x) = i,\ h_Z(z) = k\}$

- Partitioned Relation
  - Multi-Dim Array
    - $R_{ij}(X, Y)$; $S_{jk}(Y, Z)$; $T_{ik}(X, Z)$
  - Partition is sorted
- Sorting Cheaper
  - # part is small
    - Ips4o Sort - Parallel
    - Partition - Sequential
  - # part is large
    - Std Sort - Sequential
    - Partition - Parallel

R

X	Y	Z
0	1	2
0	1	2
0	1	3
0	4	1
2	4	1
2	4	1
2	5	7
5	3	9

A

h_X	h_Y	h_Z
1	1	0
1	1	0
1	1	0
1	0	0
0	0	0
0	0	0
0	0	0
1	0	0

$\tilde{R}$

X	Y	Z
2	4	1
2	4	1
2	5	7
0	4	1
5	3	9
0	1	2
0	1	2
0	1	3

PartID			B	B^{pref}
0	0	0	3	0
0	1	0	2	3
1	0	0	0	5
1	1	0	3	5

[Kjolstad'2017]

Preprocessor: Coco Index

- $R_{ij}(X, Y) = \{(x, y) \in R \mid h_X(x) = i, h_Y(y) = j\}$
- $S_{jk}(Y, Z) = \{(y, z) \in S \mid h_Y(y) = j, h_Z(z) = k\}$
- $T_{ik}(X, Z) = \{(x, z) \in T \mid h_X(x) = i, h_Z(z) = k\}$

- Compressed Column
  - Trie-like
  - Flatten sorted array
  - For each partition
  - Parallel construction

- Benefits
  - Reduce construction
  - Allow compute eagerly
  - Avoid contention

R

X	Y	Z
0	1	2
0	1	2
0	1	3
0	4	1
2	4	1
2	4	1
2	5	7
5	3	9

$\xrightarrow[\textit{Build}]{\textit{Scan}}$

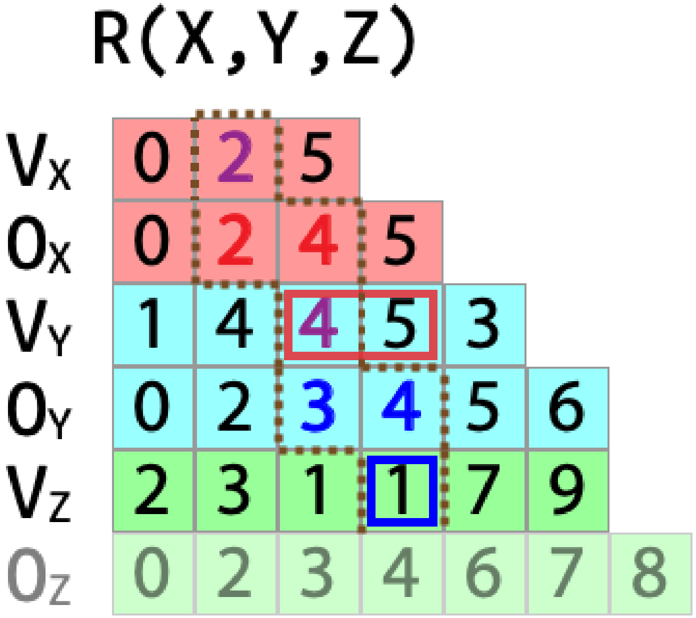
CoCo Index

V_X	0	2	5				
O_X	0	2	4	5			
V_Y	1	4	4	5	3		
O_Y	0	2	3	4	5	6	
V_Z	2	3	1	1	7	9	
O_Z	0	2	3	4	6	7	8

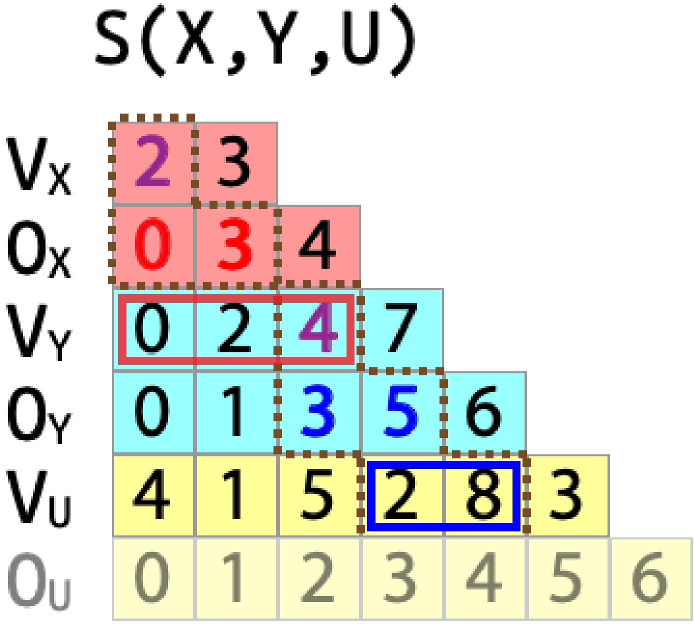
Executor: Join over CoCo

- $R_{ij}(X, Y) = \{(x, y) \in R \mid h_X(x) = i, h_Y(y) = j\}$
- $S_{jk}(Y, Z) = \{(y, z) \in S \mid h_Y(y) = j, h_Z(z) = k\}$
- $T_{ik}(X, Z) = \{(x, z) \in T \mid h_X(x) = i, h_Z(z) = k\}$

- Compute GJ in parallel
  - Assign partition to thread with id mapping
  - Thread (i,j,k) access R_{ij}, S_{jk}, T_{ik} partition
- Intersect over Coco
  - Merge Intersection
    - Search next larger values iteratively
    - Exp, Quad, Lin
  - With Restriction
    - In next level (trie)
    - By offset vector



R.X = 2
index(2, V_X) = 1
$O_X[1]=2$ $O_X[2]=4$
$V_Y[2,4) = \{4,5\}$



S.X = 2
index(2, V_X) = 0
$O_X[0]=0$ $O_X[1]=3$
$V_Y[0,3) = \{0,2,4\}$

Optimizer: Cost Model

- Robust to Datasets
  - Dense or Sparse
  - Skew or Uniform
- Key Ideas
  - measure the complexity of intersections based on the size of relations
  - Sum up the complexity with specific constants
  - Estimate the summation by calculating worst case projected join size

Table 1. Basic Notations for Cost Model

Notations	Definition
$\boldsymbol{x}$ / $\boldsymbol{X}$	tuple of constants / variables
$\boldsymbol{x}_\sigma$ / $\boldsymbol{X}_\sigma$	permuted tuple of constants / variables
$\boldsymbol{x}_{i:j}$ / $\boldsymbol{X}_{i:j}$	projected tuple of constants / variables[4]
$\boldsymbol{x} \oplus \boldsymbol{X}$	concatenation of constants or variables
$\mathcal{S}$	set of relations, e.g. $\{R_{j_1}, \cdots, R_{j_k}\}$
$\lvert\mathcal{S}\rvert$	the size of $\mathcal{S}$, e.g. k above
$\mathcal{N}$	cardinalities of relations in $\mathcal{S}$, e.g. $\{\lvert R_{j_1}\rvert, \ldots, \lvert R_{j_k}\rvert\}$

$$C[\boldsymbol{x}_{\sigma(1:i-1)} \oplus X_{\sigma(i)}] = |\mathcal{S}[X_{\sigma(i)}]| \cdot \min \mathcal{N}[\boldsymbol{x}_{\sigma(1:i-1)} \oplus X_{\sigma(i)}] \cdot \log_2\left(1 + \frac{\max \mathcal{N}[\boldsymbol{x}_{\sigma(1:i-1)} \oplus X_{\sigma(i)}]}{\min \mathcal{N}[\boldsymbol{x}_{\sigma(1:i-1)} \oplus X_{\sigma(i)}]}\right)$$

$$C[X_{\sigma(1:i)}] = \sum_{\boldsymbol{x}_{\sigma(1:i-1)} \in Q(X_{\sigma(1:i-1)})} C[\boldsymbol{x}_{\sigma(1:i-1)} \oplus X_{\sigma(i)}]$$

$$C[X_{\sigma(1:i)}] \leq |\mathcal{S}(X_{\sigma(i)})| \cdot (\sum \min \mathcal{N}) \cdot \log_2\left(1 + \frac{\sum \max \mathcal{N}}{\sum \min \mathcal{N}}\right)$$

Optimizer: Ordering Partitioning

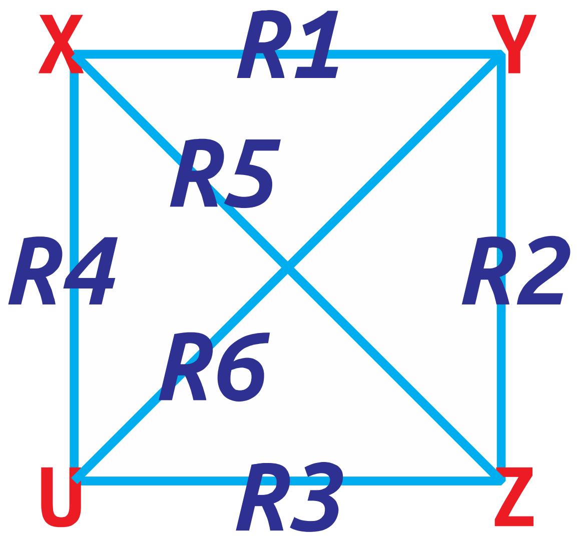
- Ordering - Based on Cost Model
  - Choose a variable order with min cost
  - Use DP $O(2^{n-1})$ for small #attr and Greedy O(N*N) for large #attr
- Partitioning – Based on viable variable ordering
  - Notice value will be discovered redundantly

$$\mathbb{C}[X_\sigma, P_\sigma] = \sum_{i \in [n]} \Big((\prod_{j>i} P_{\sigma(j)}) C[X_{\sigma(1:i)}] \Big)$$

  - Prune Heuristically by avoiding large costs and small parts
  - Measure Evenness to favor evenly distributed partition numbers
    - But with a bias having more shares on last variables in the ordering

Duplicated Intersection

- Illustration using the 4-clique query
- Colored Intersection is duplicated
  - Independent to previous attributes
  - Redundant to compute inside the loop

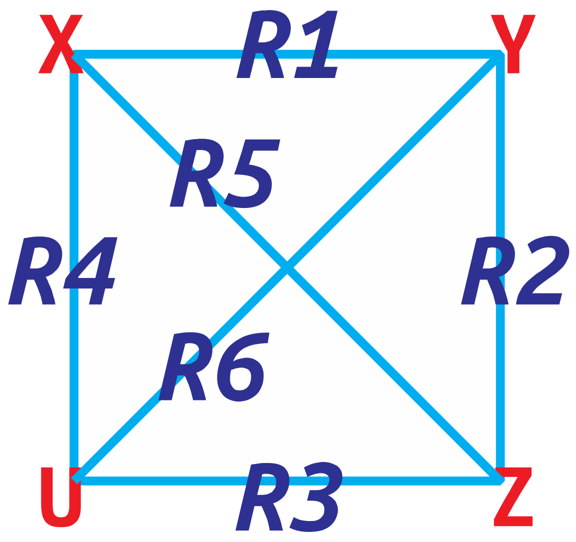



```
Q(X,Y,Z,U) := R1(X,Y),R2(X,Z),R3(X,U),
              R4(Y,Z),R5(Y,U),R6(Z,U).
```

```
# BEFORE REWRITING
For x ∈ R1.X ∩ R2.X ∩ R3.X
  For y ∈ R1[x].Y ∩ R4.Y ∩ R5.Y
    For z ∈ R2[x].Z ∩ R4[y].Z ∩ R6.Z
      For u ∈ R3[x].U ∩ R5[y].U ∩ R6[z].U
        Q += (x, y, z, u)
```

Query Rewriting

- Procedure:
    - Identify and Lift up duplicated intersections
    - Treat as independent computational entities
    - Compute them only once
    - Cache/Reuse the results
- Keeping WCOJ
    - May be not optimal
    - Use cost model to avoid




```
# AFTER REWRITING
tmp_Y = R4.Y ∩ R5.Y # Lift Up Y
For x ∈ R1.X ∩ R2.X ∩ R3.X
  tmp_Z = R2[x].Z ∩ R6.Z # Lift Up Z
  For y ∈ R1[x].Y ∩ tmp_Y
  tmp_U = R3[x].U ∩ R5[y].U # Lift Up U
    For z ∈ R4[y].Z ∩ tmp_Z
      For u ∈ R6[z].U ∩ tmp_U
        Q += (x, y, z, u)
```

Experiments

Setup

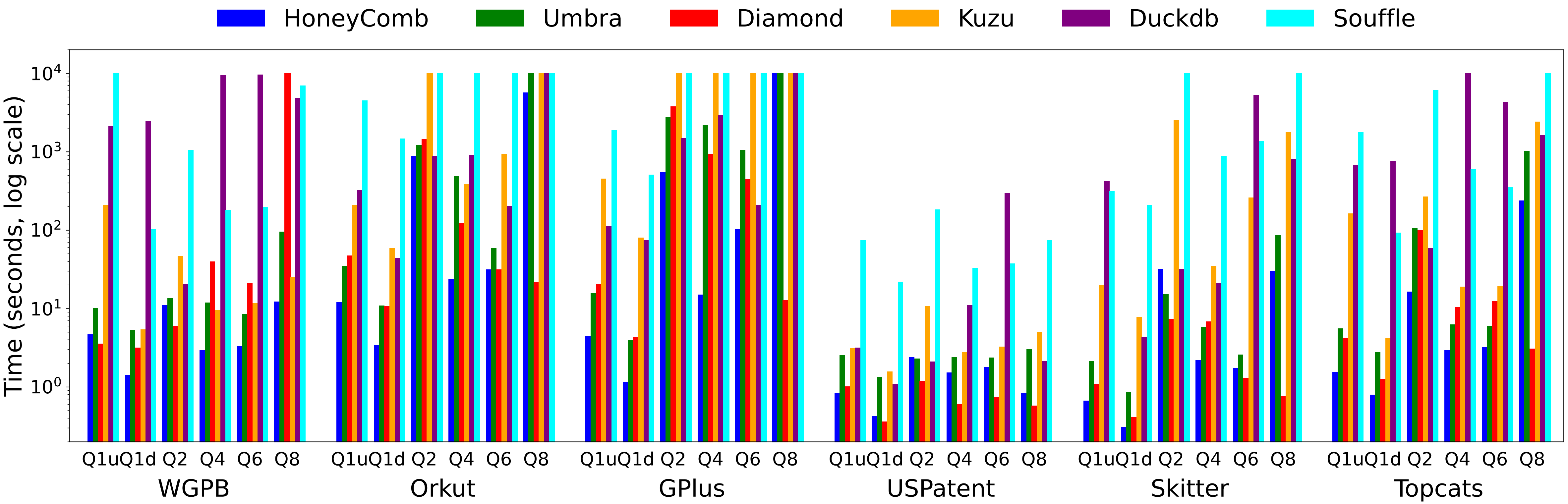
- Baselines
  - Umbra (DB with WCOJ)
  - Diamond (Umbra Variants)
  - Kuzu (Graph with WCOJ)
  - DuckDB (DB no WCOJ)
  - Souffle (Datalog no WCOJ)

Name	# Node	# Edge	Feature
WGPB [1, 28]	54.0M	81.4M	sparse, skew
Orkut [41]	3.07*M*	117*M*	partial dense, uniform
GPlus [39]	107K	13.6M	dense, skew
USPatent [38]	3.77M	16.5M	sparse, uniform
Skitter [38]	1.69M	11.1M	sparse, partial skew
Topcats [50]	1.79M	28.5M	partial dense, skew

Name	**Queries**
Q1 (Triangle)	`Q(X, Y, Z) := R(X, Y), S(Y, Z), T(X, Z).`
Q2 (4-Loop)	`Q(X, Y, Z, U) := R1(X, Y), R2(X, Z), R3(Y, U), R4(Z, U).`
Q4 (4-Diamond)	`Q(X, Y, Z, U) := R1(X, Y), R2(X, Z), R3(Y, U), R4(Z, U), R5(Y, Z).`
Q6 (4-Clique)	`Q(X, Y, Z, U) := R1(X, Y), R2(X, Z), R3(Y, U), R4(Z, U), R5(Y, Z), R6(X, U).`
Q8 (2-Triangle)	`Q(X, Y, Z, U, V) := R1(X, Y), R2(X, Z), R3(Y, Z), R4(Z, U), R5(Z, V), R6(U, V).`
LW (Loomis-Whitney)	`Q(X, Y, Z, U) := R1(X, Y, Z), R2(X, Y, U), R3(X, Z, U), R4(Y, Z, U).`
CT (Clover-Triangle)	`Q(U, X, Y, Z) := R5(U, X, Y), R6(U, X, Z), R7(U, Y, Z).`

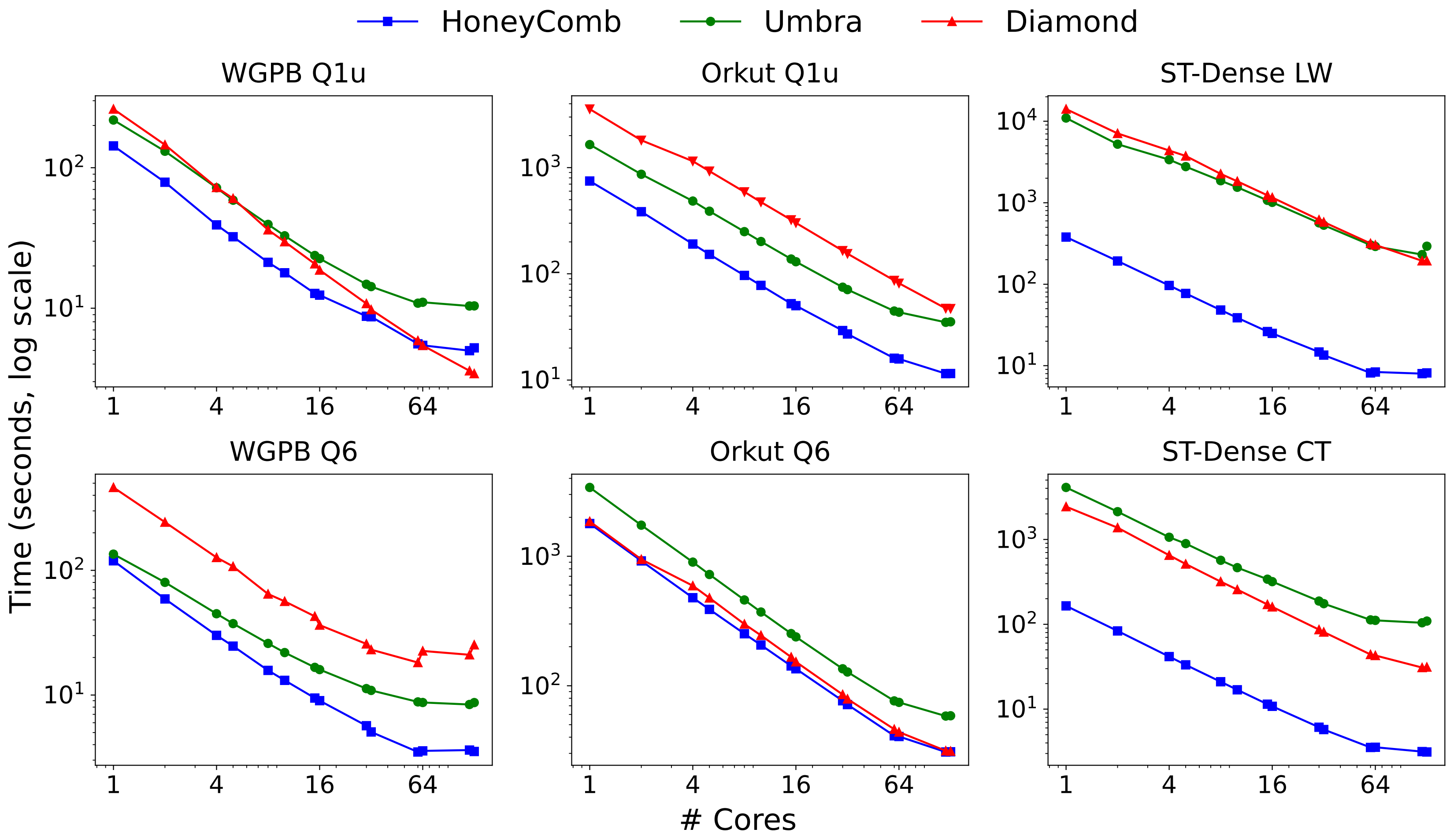
Performance (Graph)

- Our approach beats baseline in most queries and datasets
  - On dense data, Umbra is slowed down by the lazy index
  - On sparse data, rewriting and index opt are most useful



Scalability

- Linear Scale
  - <=60 threads
  - physical
- In Detail
  - Perfect Scale on Join Stage
  - Good Scale on Preprocessing Stage

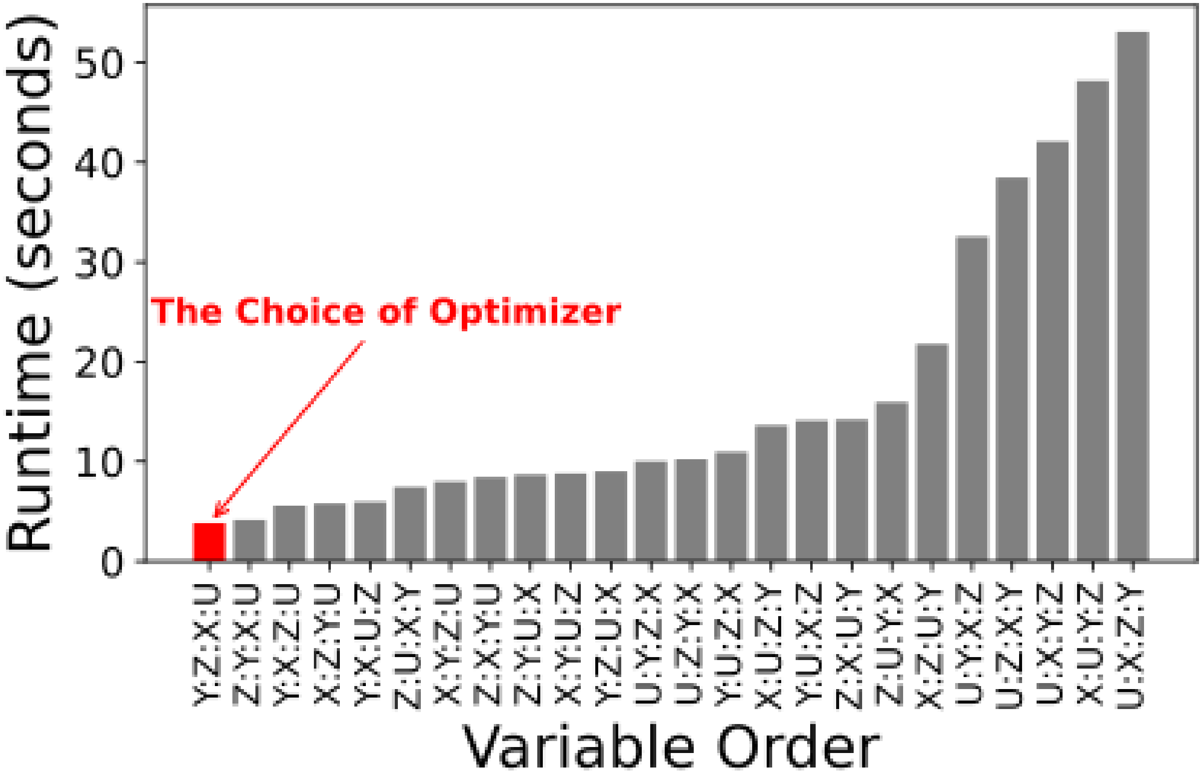


Variable Order

$$Q(X,Y,Z,U) = R1(X,Y) \wedge R2(X,Z) \wedge R3(X,U) \wedge R4(Y,Z) \wedge R5(Y,U) \wedge R6(Z,U)$$

- Runtime (4-clique on WGPB)
- x-label (Variable order)
  - 4!=24 combinations

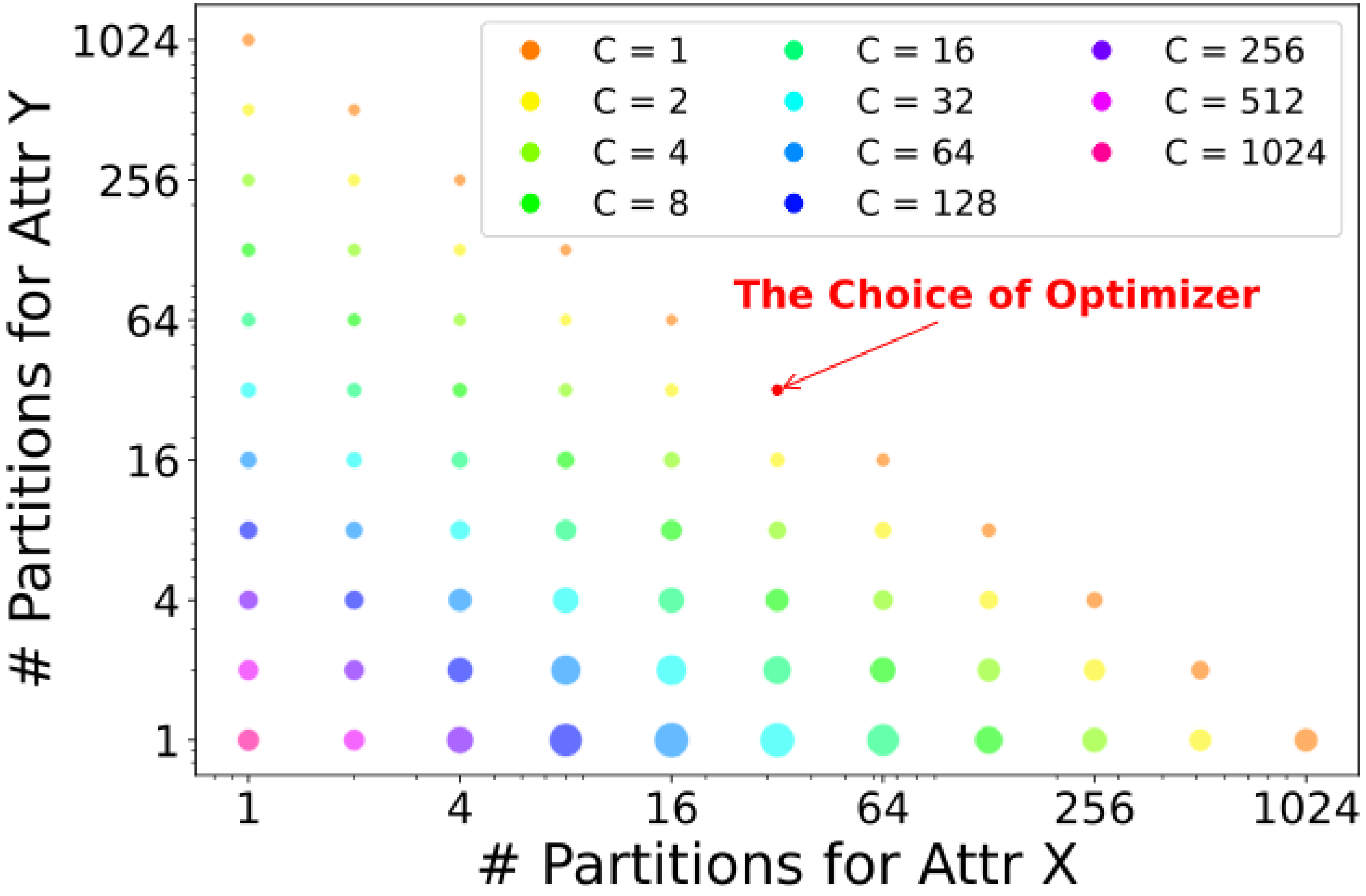
- Variable Order
  - Heavily effect performance
  - Influence # intermediate
  - No need to be the best



Partition

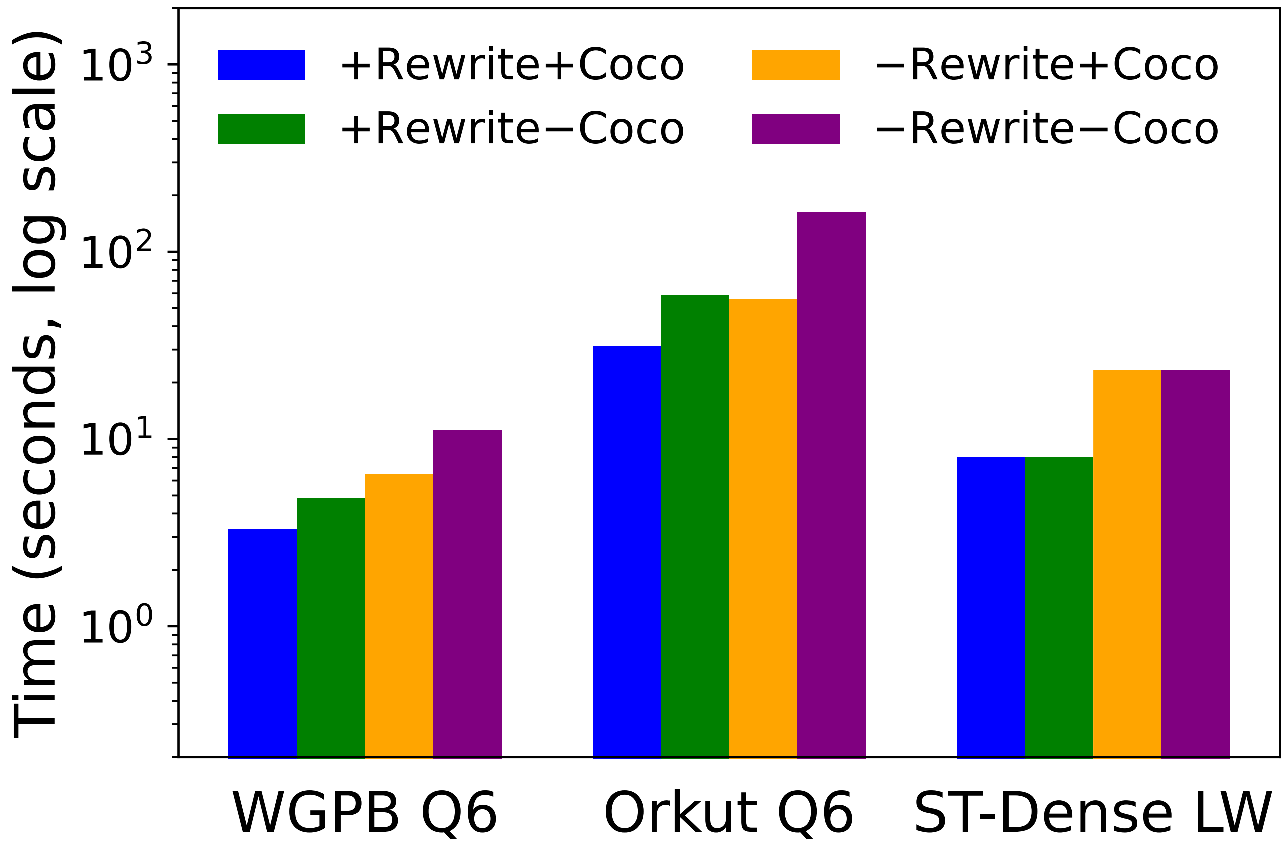
```
// Partition R, S and T (see text)
// In parallel, Thread (i, j, k)
For x ∈ Rij.X ∩ Tik.X
  For y ∈ Rij[x].Y ∩ Sjk.Y
    For z ∈ Sjk[y].Z ∩ Tik[x].Z
      Q += (x, y, z)
```

- Dot size
  - Relative Runtime
  - smaller is better
- More Partitions on C cause performance slow
  - Due to duplicated intersection on B
- Even Partitions on A,B cause performance fast
  - Due to non skewness of output tuples in join



Rewriting and Indexing

- Rewriting
  - Optimize execution plans
  - Efficient query processing
  - No rewrite on Query LW
    - No duplicated intersections
- Coco Index
  - Significant Improvement
  - Index Size: 1GB-5GB
    - Comparable to data sizes
  - Preprocess Time: 0.2s~5s
    - Acceptable Overhead



Conclusion

- Shared Memory ManyCore WCOJ Architecture
  - Sorting based Two Stage WCOJ
  - No Costly Index and No R/W Conflicts
  - Robust to Variety of Dataset
  - Fully Parallelization and Scalable
- With Comprehensive Complexity Cost Model
  - For Variable Ordering and Partitioning
- With Novel Optimization Technique
  - Cache Mechanisms to remove duplication

Future Work

- A Faster Estimation Way
  - reduce to polynomial
  - maybe even linear with precomputed stats
- A Wider Rewriting Mechanism
  - explore/reduce more potential duplication
  - take over the hypertree decomposition
  - integrate with ordering decision

Q & A

Thanks!!!