Advanced Query Optimization
Advanced Topics in Databases

Andreas Meister
Otto-von-Guericke University Magdeburg
Summer Semester 2014
Agenda I

1. Motivation
2. Query Processing Phases
3. Logical Query Optimization
4. Physical Query Optimization
5. Outlook
Part I

Motivation
Database Optimization Problems

- Several optimization problems in databases:
  - Query optimization
  - Physical database design
  - Maintenance tasks

- In this lecture: Focus on query optimization
Physical Database Design

- Optimization of data storage and access
- Provides options for query optimization
- Consists of several subproblems:
  - Replication strategies
  - Materialized view selection
  - Index selection
  - Partitioning schema
Maintenance tasks

- Guarantee the efficiency during the runtime
- Several other optimization problems:

  - Data compression
  - Update merging
  - Buffer management
  - Self-Tuning
Basic principles - Query Optimization

• Basis language
  • SQL
  • Relational calculus
  • Here: Relational algebra

• Optimization goals
  • Fast query processing
    ⇒ Consider as few page accesses/rows as possible during query processing
    ⇒ Consider as few page accesses/rows as possible during all operations
Example

```
SELECT CUSTOMER.CNo, Last_Name
FROM CUSTOMER, ORDER
WHERE CUSTOMER.CNo = ORDER.CNo
    AND Date = DATE '22-NOV-13'
```

- **Table CUSTOMER**: 100 rows; one page: 5 rows
- **Table ORDER**: 10,000 rows; one page: 10 rows
- 50 orders per day
- 50 rows of (CNo, Last_Name) on one page
- 3 rows of $R(CUSTOMER) \times R(ORDER)$ on one page
- Puffer for every relation size: 1, no clamping sets
Direct evaluation

1. $R_1 := R(\text{CUSTOMER}) \times R(\text{ORDER})$
   
   Page accesses:
   - $r : (100/5 \cdot 10.000/10) = 20.000$
   - $w : (100 \cdot 10.000)/3 = 333.000 \text{ (ca.)}$

2. $R_2 := \sigma_{\text{SEL}}(R_1)$
   
   - $r : 333.000 \text{ (ca.)}$
   - $w : 50/3 = 17 \text{ (ca.)}$

3. $R_{\text{erg}} := \pi_{\text{PROJ}}(R_2)$
   
   - $r : 17$
   - $w : 1$

Overall ca. 687.000 page accesses and ca. 333.000 pages for intermediate storage
Optimized evaluation

1. \( R_1 := \sigma_{\text{Date}=’22.11.13’}(R(\text{ORDER})) \)
   - \( r : 10.000/10 = 1.000 \)
   - \( w : 50/10 = 5 \)

2. \( R_2 := R(\text{CUSTOMER}) \bowtie_{\text{CNo}=\text{CNo}} R_1 \)
   - \( r : 100/5 \cdot 5 = 100 \)
   - \( w : 50/3 = 17 \)

3. \( R_{\text{erg}} := \pi_{\text{PROJ}}(R_2) \)
   - \( r : 17 \)
   - \( w : 1 \)

Ca. 1.140 page accesses (improved by factor about 600)
Evaluation with index usage

Indices $I(\text{ORDER(Date)})$ and $I(\text{CUSTOMER(CNo)})$

1. $R_1 := \sigma_{\text{Date}=22.11.13}(R(\text{ORDER}))$ over $I(\text{ORDER(Date)})$
   - $r$: minimal 5, maximal 50; $w: 50/10 = 5$

2. $R_2 := \text{sort } R_1$ over $\text{CNo}$
   - $r + w: 5 \cdot \log 5 = 15$ (ca.)

3. $R_3 := \text{CUSTOMER } \bowtie_{\text{CNo} = \text{CNo}} R_2$
   - $r: 100/5 + 5 = 25$; $w: 50/3 = 17$

4. $R_{\text{erg}} := \pi_{\text{PROJ}}(R_3)$
   - $r: 17$; $w: 1$

Maximal ca. 130 and minimal ca. 85 page accesses
Comparison of different variants

<table>
<thead>
<tr>
<th>Execution variants</th>
<th>Read and write operations</th>
<th>pages for intermediate results</th>
</tr>
</thead>
<tbody>
<tr>
<td>direct evaluation</td>
<td>ca. 687.000</td>
<td>ca. 333.000</td>
</tr>
<tr>
<td>optimized evaluation</td>
<td>ca. 1.140</td>
<td>17</td>
</tr>
<tr>
<td>evaluation with indices</td>
<td>min. 85 max. 130</td>
<td>17</td>
</tr>
<tr>
<td>with pipelining</td>
<td>51 to 96</td>
<td>5 (plus sorting)</td>
</tr>
</tbody>
</table>
Part II

Phases
Phases of Query processing

SQL-Query

Translation & View resolving

Algebra

Standardization & Simplification

Optimization

Access plan

Execution

Code-Generation

Plan parameterization

Translation time

Runtime

Logical optimization

Physical optimization

Cost-based Selection
Phases of query processing

1. Translation and view resolving
   - Simplification of arithmetic expressions
   - Resolve subqueries
   - Insertion of view definitions

2. Logical or algebraic optimization
   - Transformation of queries independent of storage information
3. **Physical or internal Optimization**
   - Consider storage information (Indices, Clusters)
   - Choose algorithms
   - Consider several alternative internal plans

4. **Cost-based selection**
   - Use statistical information (table size, selectivity of attributes) for the selection of one internal plan

5. **Plan parameterization**
   - In pre-compiled queries (e.g., n embedded-SQL): Replace placeholders with values

6. **Code-Generation**
   - Transformation of the access plan in executable code
Phases of Query processing /3

- Representation of queries during the processing
  - Algebraic expression $\rightarrow$ **Operator tree**
    - Operators as nodes
    - Edges represent data flow
  - Later $\rightarrow$ **Query execution plan (QEP)**
    - Concrete algorithms as operator nodes
Part III

Logical Optimization
Logical Optimization

- **Heuristic methods**
  - E.g., algebraic optimization (Rewriting)
  - For relational algebra + grouping, ...

- **Exact methods**
  - Reducing number of joins
  - For particular relational algebra queries
Algebraic Optimization

- Replacement of terms of the relational algebra based on algebraic equivalence

- Equivalence used as replacement rules

- Heuristic methods:
  - Move operations to reduce intermediate results
  - Identify redundancies
Removing of redundant operations

- Necessary in queries with views

\[ R(\text{ACT\_PRODUCTS}) = R(\text{PRODUCTS}) \bowtie \pi_{\text{ProdNo, Label}}(\ldots \sigma_{\text{Date} = \text{current\_date}}(R(\text{ORDER}))) \]

- Query on view:

\[ \pi_{\text{Label, Price}}(R(\text{PRODUCT}) \bowtie R(\text{ACT\_PRODUCTS})) \]

- View expansion:

\[ \pi_{\text{Label, Price}}(R(\text{PRODUCTS}) \bowtie R(\text{PRODUCTS}) \bowtie \pi_{\text{ProdNo, Label}}(\ldots \sigma_{\text{DATE} = \text{current\_date}}(R(\text{ORDER}))) \]) \]

- Rules: idempotence

\[ R = R \bowtie R \] meaning \( \bowtie \) is idempotent
Moving of selections

$$\sigma_{\text{Price}>100}(R(\text{ORDER}) \bowtie R(\text{PRODUCT}))$$

more efficient:

$$R(\text{ORDER}) \bowtie (\sigma_{\text{Price}>100}(R(\text{PRODUCT})))$$

Rules:

$$\sigma \text{ and } \bowtie \text{ commutate}$$

Only if the attributes of the selections predicate supporting this
Join order

- Statistical information of the catalog necessary

\[ (R(\text{CUSTOMER}) \bowtie R(\text{PRODUCT})) \bowtie R(\text{ORDER}) \]

- First join: Cartesian product, therefore:

\[ R(\text{CUSTOMER}) \bowtie (R(\text{PRODUCT}) \bowtie R(\text{ORDER})) \]

- Rules:

  \( \bowtie \) is associative and commutative

- No unique preferable application of this rule (therefore, internal optimization, cost-based)
Algebraic rules - Example

- **CommJoin**: Operator $\Join$ is commutative:
  \[ r_1 \Join r_2 \iff r_2 \Join r_1 \]

- **AssocJoin**: Operator $\Join$ is associative:
  \[ (r_1 \Join r_2) \Join r_3 \iff r_1 \Join (r_2 \Join r_3) \]

- **ProjProj**: For operator $\pi$ the outer dominates the inner parameter in the combination:
  \[ \pi_X(\pi_Y(r_1)) \iff \pi_X(r_1) \]

- Plenty more rules available [Saake et al., 2012]
Further rules

• Idempotences

\[
\text{IdemUnion: } r_1 \cup r_1 \iff r_1 \\
\text{IdemIntersect: } r_1 \cap r_1 \iff r_1 \\
\text{IdemJoin: } r_1 \Join r_1 \iff r_1 \\
\text{IdemDiff: } r_1 - r_1 \iff \{} 
\]

• Association with empty relation:

\[
\text{EmptyUnion: } r_1 \cup \{\} \iff r_1 \\
\text{EmptyIntersect: } r_1 \cap \{\} \iff \{\} \\
\text{EmptyJoin: } r_1 \Join \{\} \iff \{\} \\
\text{EmptyDiffRight: } r_1 - \{\} \iff r_1 \\
\text{EmptyDiffLeft: } \{\} - r_1 \iff \{\}
\]

• for $\Join$, $\cup$ and $\cap$: \textit{commutative} and \textit{associative law}
Algebraic optimization: Algorithm

- Simple optimization algorithm
  1. Resolve complex selection predicate, if applicable resolving of \( \neg \) and \( \lor \)
  2. Remove redundant operators
  3. Pushing down selections as near as possible to the leaf
  4. Resolve cross joins
  5. Pushing projections in leaf direction

- Single steps will be executed in the stated order until no replacement can be performed
Algebraic optimization: example

\[ \pi_{\text{OrderNo}, \text{CNo}} \left( \sigma_{\text{Date}<'18.2.14'} \land \text{Label}='\text{Arabica Black}' \left( \pi_{\text{OrderNo}, \text{CNo}, \text{Date}, \text{Label}} \left( R(\text{PRODUCT}) \bowtie R(\text{ORDER}) \bowtie R(\text{CUSTOMER}) \right) \right) \right) \]
Unoptimized query plan

\[ \pi \text{OrderNo, CNo} \]
\[ \sigma \text{Date} > '18.02.14' \land \text{Label} = 'Arabica Black} \]
\[ \pi \text{OrderNo, CNo, ProdNo, Label, Date} \]
\[ \bowtie \text{CUSTOMER} \]
\[ \bowtie \text{PRODUCTS} \]
\[ \bowtie \text{ORDER} \]
Query plan

- Remove redundant operations

\[ \pi \text{OrderNo, CNo} \]
\[ \sigma \text{Date > '18.02.14'} \land \text{Label = 'Arabica Black'} \]

CUSTOMER

PRODUCTS

ORDER
Query plan /2

- Move of the selections

```
π
σ
ORDERPRODUCT
CUSTOMER
Label = 'Arabica Black'
Date > '18.02.14'
OrderNo, CNo
⊲ ⊳
⊲ ⊳
```

```
σ
Label = 'Arabica Black'
σ
Date > '18.02.14'
PRODUCT
ORDER
```
Query plan /3

- with additional projections

\[ \pi_{\text{OrderNo, CNo}} \]
\[ \sigma_{\text{Label} = 'Arabica Black'} \]
\[ \sigma_{\text{Date} > '18.02.14'} \]
Algebraic optimization: example from CoGaDB

```
SELECT  d_year, s_nation, p_category,
       SUM(lo_revenue - lo_supplycost) AS profit
FROM  lineorder, dates, customer, supplier, part
WHERE  lo_orderdate = d_datekey AND
    lo_custkey = c_custkey AND
    lo_suppkey = s_suppkey AND
    lo_partkey = p_partkey AND
    c_region = 'AMERICA' AND s_region = 'AMERICA'
    AND (d_year = 1997 OR d_year = 1998) AND
    (p_mfgr = 'MFGR# 1' OR p_mfgr = 'MFGR# 2')
GROUP BY  d_year, s_nation, p_category
ORDER BY  d_year, s_nation, p_category
```
Algebraic optimization: example from CoGaDB

- Transform sql query in relational algebra/ operator tree
Algebraic optimization: example from CoGaDB

- Resolve complex selection predicate
Algebraic optimization: example from CoGaDB

• Push down selections

\[
\gamma \sum(LO_{REVENUE} - LO_{SUPPLYCOST});D_{YEAR},S_{NATION},P_{CATEGORY}
\]

\[
\tau \ D_{YEAR},S_{NATION},P_{CATEGORY}
\]

\[
\sigma_{LO_{PARTKEY}=P_{PARTKEY}}
\]

\[
\sigma_{P_{MFGF}=MFGR1 OR P_{MFGF}=MFGR2}
\]

Part

\[
\sigma_{S_{REGION}=AMERICA}
\]

SUPPLIER

\[
\sigma_{C_{REGION}=AMERICA}
\]

CUSTOMER

\[
\sigma_{LO_{ORDERDATE}=D_{DATEKEY}}
\]

LINEORDER

\[
\sigma_{D_{YEAR}=1997 OR D_{YEAR}=1998}
\]

DATES
Algebraic optimization: example from CoGaDB

- Resolve cross joins
Possibilities for logical query optimization on modern hardware

- Application of different rules in parallel not possible
  - Deterministic application of rules
  - One optimal result

- Evaluate several plans in parallel

- Determine redundant operations faster
Part IV

Physical Optimization
From logical to physical optimization

- What to do next?

```
π
π
σ
σ
ORDER PRODUCT
CUSTOMER
OrderNo, CNo CNo
OrderNo, CNo
Label = 'Arabica Black' Date > '18.02.14'
⊲ ⊳
⊲ ⊳
π
π
σ σ
ORDER PRODUCT CUSTOMER
OrderNo, CNo CNo
OrderNo, CNo
Label = 'Arabica Black' Date > '18.02.14'
⊲ ⊳
⊲ ⊳
MERGE
⊲ ⊳
MERGE
LABEL = 'Arabica Black' DATE > '18.02.14'
⊲ ⊳
PRODUCT ORDER
```

Andreas Meister
Advanced Query Optimization
Last Change: June 16, 2014
From logical to physical optimization

- What to do next?
Algorithm selection

- Selection
  - $\sigma_{\text{REL}}^{\varphi}$: Selection through relation-scan
  - $\sigma_{\text{IND}}^{\varphi}$: Selection over index

- Join
  - $\bowtie^{\text{LOOP}}$: Nested-loop join
  - $\bowtie^{\text{MERGE}}$: Merge join (Requirement: Input relations are sorted on joint attribute(s))
  - $\bowtie^{\text{HASH}}$: Hash-Join
Algorithm selection /2

- **Projection**
  - $\pi_{\text{AttList}}^{\text{REL/with}}$: Projection through relation-scan; **with** duplicate removal
  - $\pi_{\text{AttList}}^{\text{REL/without}}$: Projection through relation-scan; **without** duplicate removal
  - $\pi_{\text{AttList}}^{\text{SORT/with}}$: Projection through scan over relation sorted by $\text{AttList}$ **with** duplicate removal
  - $\pi_{\text{AttList}}^{\text{SORT/without}}$: Projection through scan over relation sorted by $\text{AttList}$ **without** duplicate removal
Algorithm selection /3

• Grouping
  • $\gamma^\text{SORT}_{F;\text{AttList}}$ Grouping on $\text{AttList}$ and application of the aggregation $F$ through sorting
  • $\gamma^\text{HASH}_{F;\text{AttList}}$ Grouping on $\text{AttList}$ and application of the aggregation $F$ through hashing
Algorithm selection /4

- Index access on predicate $A \Theta a$

$$\sigma_{A \Theta a}^{IND}(l(R(A))) \rightarrow \text{list(tid)}$$

- Special case

$$\sigma_{true}^{IND}(l(R(A)))$$

- Combination of projection

$$\pi_{\text{AttList}}^{IND/\text{with}} \text{ or } \pi_{\text{AttList}}^{IND/\text{without}}$$

- Without access of base relation

$$\pi_{A}^{IND}(l(R(A)))$$
New operators

- For TID-lists: ""Realization""-operator $\rho$

$$\rho(\langle \text{TID-List for } R\text{-Tuple} \rangle, R(R))$$

- Set operators $\cup$, $\cap$ and $-$ on TID-Lists

- **Sorting** of tuples $\omega$

$$\omega_{\text{AttList}}(\langle \text{Tupel-order} \rangle)$$
Example for execution plans

\[
\text{SELECT} \quad * \\
\text{FROM} \quad \text{ORDER} \\
\text{WHERE} \quad \text{ProdNo} = 42 \quad \text{AND} \\
\quad (\text{SName} = 'Kaffeebude' \quad \text{OR} \\
\quad \text{SName} = 'CoffeeShop') \quad \text{AND} \\
\quad \text{Amount} < 10
\]
Example for execution plans /2

\[ \sigma_F \rightarrow \sigma_{\text{IND}} \]  
\[ \sigma_{\text{IND}} \text{Amount} < 10 \]  
\[ \left( \sigma_{\text{IND}} \text{OrderNo} = 42 \right) \]  
\[ I(\text{ORDER(ProdNo))} \]  
\[ \sigma_{\text{IND}} \text{SName} = 'Kaffeebude' \]  
\[ I(\text{ORDER(SName))} \]  
\[ \sigma_{\text{IND}} \text{SName} = 'Coffeeshop' \]  
\[ I(\text{ORDER(SName))} \]
Algebraic optimization: example from CoGaDB

```
ɣ SUM(LO.REVENUE - LO_SUPPLYCOST); D.YEAR, S.NATION, PCATEGORY
σ P_MFGR = MFGRr1 OR P_MFGR = MFGRr2
σ D.YEAR = 1997 OR D.YEAR = 1998
σ S_REGION = AMERICA
σ C_REGION = AMERICA
σ D.ORDERDATE = D_DATEKEY
```

Andreas Meister
Advanced Query Optimization
Last Change: June 16, 2014
46/114
Cost-based Optimization

- query
- span search space
- equivalent plans
- search strategy
- "best" plan
- transformation rules
- cost model
Spanning of the search space

- Search space: Set of all equivalent query plans
- Spanning through \textit{transformation rules} (algebraic rules)
- Focus: \textit{join trees}
- for \( n \) relations:
  - Alone \( n! \) different left- or right-deep trees (permutation)
  - \( n = 10 \): 3.628.800 left-deep and 17.643.225.600 trees overall!
- Therefore: Pruning of the search space through
  - Heuristics (algebraic optimization)
  - Predefined tree "form"
Join trees

- Linear order of joins: left- or right-deep trees
- Bushy trees
Spanning of the search space /2

- Number of bushy join tree between \( n \) relations

\[
S(1) = 1 \\
S(n) = \sum_{i=1}^{n-1} S(i)S(n-i)
\]

- For number \( i \) from 1 \ldots n – 1 leafs of on part tree: \( S(i) \) different tree forms
- For the remaining \( n – i \) leafs: also \( S(n – i) \) forms
- For every form: assign \( n \) relations in \( n! \) variants as leaf
- Overall: for \( n \) relations \( S(n) \cdot n! \) variants
Spanning of the search space /3

\[ R(\text{ORDER}) \Join R(\text{PRODUCT}) \Join R(\text{CUSTOMER}) \Join R(\text{SUPPLIER}) \]
Search strategies

- Walking through the search space
- Selection of the minimum-cost plan
- Basis: cost model

Search strategy determines
- which plans will be considered (complete / partial search)
- in which order the alternatives evaluated

- Variants: deterministic, randomized
Cost model: Components

- **Cost function**: To estimate the execution cost of operations or queries

- **Statistics**: Over size of the relations (cardinality, tuple size), value ranges and distributions

- **Formulas**: To calculate the size of (intermediate) results based on the statistics
Cost function

- Cost types:
  - I/O-costs: Caused by reading and writing of blocks from or to the external storage
  - CPU-costs: for internal calculations, comparison etc.,
  - Communication costs: in case of distributed database systems

- Usually:

\[ \text{cost} = \text{cost}_{IO} + W \cdot \text{cost}_{CPU} \]

- Factor \( W \) to calibrate regarding hardware
Cost formulas

• Idea:
  • Estimate total expense through the cardinality of intermediate results
  • Cardinality over **Selectivity** of the operators

• Selectivity $sel$:

\[
sel = \frac{\text{Expected size of the result}}{\text{Cardinality of the input relation}}
\]

• Assumption: equal distribution, independence of the attribute
Cost formulas: Selection - Example

\[ |\sigma_F(R(R))| = sel(F, R) \cdot |r| \]

- Estimation (for interpolateable, arithmetic values):

\[
\begin{align*}
sel(A = v, R) &= \frac{1}{val_{A,r}} \\
sel(A < v, R) &= \frac{v - A_{min}}{A_{max} - A_{min}} \\
sel(A > v, R) &= \frac{A_{max} - v}{A_{max} - A_{min}} \\
sel(A \text{ between } v_1 \text{ and } v_2, R) &= \frac{v_2 - v_1}{A_{max} - A_{min}}
\end{align*}
\]

- Further cost formulas presented in [Saake et al., 2012]
Improvement of the estimation

- **Parametrised functions**: Function parameter for approximation of the data distribution (e.g., normal or Zipf distribution)

- **Sample**: Estimate selectivity based on a random selected sample

- **Histograms**: Approximation of the real distribution
Histograms: Principle

- frequency
- attribute values
- real distribution
- approximate distribution
Equi-width-Histograms

- Equi-sum: sum of the source values of the bucket is equal; corresponds to the $\beta$ part of the sum of all source values

- Particularly: **Equi-sum($V$, $S$)**
  - Sorting parameter: Attribute values
  - Source parameter: Span

- Aggregate of connected ranges of attribute values in one equi-width-bucket meaning for a bucket with $[v_{\text{min}}, v_{\text{max}}]$: 

$$|v_{\text{max}} - v_{\text{min}}| \approx \frac{1}{\beta} (\max(V) - \min(V))$$
Equi-depth-Histograms

- Particularly: **Equi-sum(V, F)**
  - Sorting parameter: Attribute values
  - Source parameter: frequency

- Equal frequency ("height") in all buckets through adaptation of the width or amount of accounted buckets (range)
- If $f_i$ of $v_i$ is greater as the maximal frequency: distribute over multiple buckets
Equi-width vs. Equi-depth-Histogramm

<table>
<thead>
<tr>
<th>Price</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>9</th>
<th>30</th>
<th>50</th>
<th>99</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount</td>
<td>100</td>
<td>75</td>
<td>30</td>
<td>20</td>
<td>15</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Range of price:

- [0, 10), [10, 20), [20, 30), [30, 40), [40, 50), [50, 60), [60, 70), [70, 80), [80, 90), [90, 99]
- [0, 1][1, 1][1, 1][1, 1][1, 2][2, 2][2, 2][2, 3][3, 5][5, 99]
Construction of Equi-sum(V, F)

1. Collect values of A and sort tuples increasingly:

   \[ list_A := \pi_A^{\text{SORT/without}}(R(R)) \]

2. Choose the amount \( \beta \) of buckets

3. Determine \( \beta + 1 \) positions in \( list_A \), so that the equal number of tuples between two following positions resides;
   Positions: \( 1, 1 + \beta, \ldots, 1 + (n - 1)\beta, 1 + n\beta \) with \( n = \frac{\max(V)}{\beta} \)

4. Determine value range \( [v_{\min}^i, v_{\max}^i] \) for every bucket \( b_i \) of the values of the \( list_A \), meaning \( v_{\max}^i = list_A[i] \) and \( v_{\min}^i = list_A[i - 1] \)

5. Determine frequency \( f_i \) for every bucket \( b_i \) as number of values between \( v_{\min}^i \) and \( v_{\max}^i \); if for multiple buckets \( v_{\max}^i \) is equal, divide \( f_i \) through the amount of these buckets
Problems with Histograms

- Huge space-requirements for multi-dimensional histograms (range queries)
- Benefits of histograms depends on query requirements and histogram parameters
- Reconstruction needed, if
  - Parameters of histogram changes
  - Data changes
- Update cost can dominate the benefits of histograms
  \[\rightarrow\] Flexible selectivity estimators needed
Advanced selectivity estimators

- **Flexible selectivity estimators:**
  - **Parametric:**
    - Calculate or estimate a probability density function for the data distribution
    - Calculate the selectivity estimation by integrating over the probability density function
    - E.g., k-means clustering
  - **Sample-based:**
    - Select a sample of the available data set
    - Calculate distribution information out of the data
    - Use distribution information to estimate selectivity
    - E.g, kernel density estimator
Selectivity Estimation via Clustering

• Requirement of efficient usage of histograms in higher dimensions:

  *Data independence*

• However, real world data are normally correlated or clustered

• **Idea:** Use clustering to determine data distribution
Selectivity Estimation via Clustering

- Parametric approach [Böhm et al., 2005]

**Input**: Data set $D$, Range-Query $Q$

**Output**: Selectivity estimation

Compute model $M$ of $D$ by $\text{EM}(D)$;

    foreach cluster $C_i \in M$ do
        foreach dimension $j$ of $D$ do
            Compute $I(Q, C^j_i)$;

        Compute $I(Q, C_i) = \prod_{i=1}^{d} I(Q, C^j_i)$;

    return $\sum_{i=1}^{k} w_{C_i} \cdot I(Q, C_i)$;

- $\text{EM}(D)$ = k-means clustering
- $I(Q, C^j_i) = \text{Integral of i-th Cluster in Dimension j over Q}$
- $w_{C_i} = \text{cluster weights}$
k-means clustering - Example

1.

2.

3.
k-means clustering

- Different approaches available
- Assign data points to cluster

Simple approach:

**Input**: Data set $D$, number of initial cluster $k$

**Output**: Clustered data set

Select $k$ initial cluster centroids;

repeat

Assign each data point to one centroid;
Recompute the position of centroids;

until *centroids do not change*;
k-means-clustering: problems

- How to choose the number of clusters?
- Initial placement of centroids influence the clustering
- Assign a point to **one** cluster not always possible
k-means clustering - Gaussian Distribution Cluster

Figure: Taken from [Böhm et al., 2005]
k-means clustering - Gaussian Distribution Cluster

- Assigning data points to one cluster not always possible
  → Assign data point to multiple clusters with certain probability

- Assign a data point $x$ to a cluster $C$
  \[ P(x|C) = \frac{1}{\sqrt{(2\pi)^d |\sum_C|}} e^{\frac{1}{2}(x-\mu_C)^T (\sum_C)^{-1} (x-\mu_C)} \]

- Combined probability for k cluster
  \[ P(x) = \sum_{i=1}^{k} w_{C_i} P(X|C_i) \]

- Probability that a data point belongs to a cluster
  \[ P(C|x) = w_C \frac{P(x|C)}{P(x)} \]
k-means clustering - Gaussian Distribution Cluster

- Parameters
  - $\mu_C$: mean value of all data points in $C$
    \[ \mu_C = \frac{\sum_{x \in D} x \cdot P(C|x)}{\sum_{x \in D} P(C|x)} \]
  - $\Sigma_C$: covariance matrix
    \[ \Sigma_C = \frac{\sum_{x \in D} P(C|x)(x - \mu_C)(x - \mu_C)^T}{\sum_{x \in D} P(C|x)} \]
  - $w_C$: weight of cluster $C$
    \[ w_C = \frac{1}{|D|} \sum_{x \in D} P(C|x) \]
Integral estimation

- Approximation needed
- Several approaches available
- Example: Monte Carlo Integration
  - \( I = \int dx f(x) \)
  - where \( x = (u_1, \cdots, u_d) \)
  - Monte Carlo estimation: \( E = \frac{1}{N} \sum_{n=1}^{N} f(x_n) \)
  - \( \lim_{n \to \infty} \frac{1}{N} \sum_{n=1}^{N} f(x_n) = I \)

- Advantage
  - Error independent of dimension

- Disadvantage
  - Compute-intensive
Kernel density estimator

- Sample-based Approach:
  - Sample points distributes some probability mass to its neighborhood
    - 2-dimensional: area
    - 3-dimensional: volume
    - ...
  - Distribution of probability mass depending on used kernel function, e.g.,
    - Normal / Gaussian
    - Epanechnikov
    - ...
  - Centering local probability distributions around the sample points
  - Estimated distribution: normalized sum of all local of distribution
Different kernel functions

Figure: Taken from Wikipedia
Overlapping of different sample points

Figure: Taken from [Heimel and Markl, 2012]
Impact of different bandwidth parameter

Figure: Taken from [Heimel and Markl, 2012]
Kernel density estimator

\[ K[(x-X_i)/h] \]

area = 1

Figure: Adapted from [Blohsfeld et al., 1999]
Kernel density estimator [Blohsfeld et al., 1999]

**Input**: Number of samples $n$, Sample set $X$, query range $[a, b]$, bandwidth $h$, kernel function $K$

**Output**: Selectivity estimation

$s = 0.0$

for $i = 1$ to $n$ do

  if $(X[i] \in [a + h, b - h])$ then
    $s+ = 1.0$;
  else if $(X[i] \in [a - h, a + h])$ and $X[i] \notin [b - h; b + h]$ then
    $s+ = 0.5 - K \left( \frac{a - X[i]}{h} \right)$;
  else if $(X[i] \in [b - h, b + h]$ and $X[i] \notin [a - h; a + h]$) then
    $s+ = K \left( \frac{b - X[i]}{h} \right) - 0.5$;
  else if $(X[i] \in \{ [b - h, b + h] \cup [a - h; a + h] \})$ then
    $s+ = K \left( \frac{b - X[i]}{h} \right) - K \left( \frac{a - X[i]}{h} \right)$;

return $\frac{s}{n}$;
Estimating the density at a point

- Basic formula: \( \hat{p}(\vec{x}) = \frac{1}{s} \sum_{i=1}^{s} K_H(\vec{x} - \vec{x}(i)) \)

- Probability density function: \( K_H(\vec{x}) = \frac{1}{|H|} K(H^{-1} \vec{x}) \)

- \( H \): bandwidth matrix;

- \( K(\vec{x}) \): kernel function
  - Gaussian kernel function: \( K_G(\vec{x}) = (2\pi)^{-\frac{d}{2}} \exp(-\frac{1}{2} \vec{x}^T \vec{x}) \)
  - Epanechnikov kernel function: \( K_E(\vec{x}) = \left(\frac{3}{4}\right)^d \cdot \prod_{i=1}^{d} (1 - x_i^2) \cdot 1_{|x_i| \leq 1} \)
Kernel density estimator

• Advantages
  • Fast convergence against underlying distribution
  • Good support for multidimensional data
  • Easy maintenance

• Disadvantages
  • Compute-intensive
Summary - Selectivity estimation

- Histograms
  - Easy, understandable approach
  - Low computational overhead during runtime
  - High maintenance cost
  - High storage consumption in higher dimension

- Parametric & Sample-based
  - Reduces maintenance costs
  - Compute-intensive
  - Flexible estimation of selectivities
  - Trade-off between accuracy and computational cost
Outline - GPU-accelerated selectivity estimators

- Presented advanced selectivity estimators adapted on GPUs

  - Advantages:
    - Parallel processing of sample points
    - Reduced execution time
    - More accurate estimations in the same time compared to CPU-algorithms
Cost-based plan selection

- Search strategy for cost-optimal plan
- Costs: Total cost based on the cost model
- Goal: Avoid exhaustive search
- Deterministic vs. randomized approaches
Dynamic programming

- Idea: Optimal solution only contains optimal partial solutions
  - Partition the problem in depended subproblems
  - Solve subproblems optimal
  - Solve subproblmes occurring several times only once

- For query optimization:
  - Optimize join order between $n$ relations through optimization of partial joins between 2, 3, $n - 1$ relations
Dynamic programming /2

- Approach: cost table with
  - $k$-membered subset $\mathcal{R} \subseteq \{r_1, r_2, \ldots, r_n\}$
  - optimal solution (join order)
  - cost
  - needed information for cost calculation (e.g., size of intermediate results)
DP: Basis algorithm for join order

**Input**: Join query $Q$ on relations $r_1, \ldots, r_n$

**Output**: Query plan for $Q$

```
for $i \leftarrow 1$ to $n$ do
  optPlans[$r_i$] $\leftarrow$ $r_i$;
for $i \leftarrow 2$ to $n$ do
  foreach $s \subseteq \{r_1, \ldots, r_n\}$ with $|s| = i$ do
    optPlans[$s$] $\leftarrow$ $\{\}$;
    foreach $r_k \in s$ do
      optPlans[$s$] $\leftarrow$ optPlans[$s$] $\cup$ join-plans(optPlans[$s - \{r_k\}$], $r_k$);
return optPlans[$\{r_1, \ldots, r_n\}$];
```
DP: Example

1. Step

<table>
<thead>
<tr>
<th>R</th>
<th>result size</th>
<th>optimal plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>{CUSTOMER}</td>
<td>1.000</td>
<td>R(CUSTOMER)</td>
</tr>
<tr>
<td>{PRODUCT}</td>
<td>5.000</td>
<td>R(PRODUCT)</td>
</tr>
<tr>
<td>{SUPPLIER}</td>
<td>100</td>
<td>R(SUPPLIER)</td>
</tr>
<tr>
<td>{ORDER}</td>
<td>20.000</td>
<td>R(ORDER)</td>
</tr>
</tbody>
</table>

2. Step

<table>
<thead>
<tr>
<th>R</th>
<th>result size</th>
<th>optimal plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>{C, P}</td>
<td>5.000.000</td>
<td>R(K) $\bowtie$ R(P)</td>
</tr>
<tr>
<td>{C, S}</td>
<td>100.000</td>
<td>R(K) $\bowtie$ R(L)</td>
</tr>
<tr>
<td>{C, O}</td>
<td>20.000</td>
<td>R(K) $\bowtie$ R(B)</td>
</tr>
<tr>
<td>{P, S}</td>
<td>5.000</td>
<td>R(P) $\bowtie$ R(L)</td>
</tr>
<tr>
<td>{P, O}</td>
<td>20.000</td>
<td>R(P) $\bowtie$ R(B)</td>
</tr>
<tr>
<td>{S, O}</td>
<td>2.000.000</td>
<td>R(L) $\bowtie$ R(B)</td>
</tr>
</tbody>
</table>
### DP: Example /2

- **3. Step**

<table>
<thead>
<tr>
<th>$\mathcal{R}$</th>
<th>result size</th>
<th>optimal plan</th>
<th>costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ C, P, S }</td>
<td>5.000.000</td>
<td>$R(P) \bowtie R(S) \bowtie R(C)$</td>
<td>5.000</td>
</tr>
<tr>
<td>{ C, P, O }</td>
<td>20.000</td>
<td>$R(C) \bowtie R(O) \bowtie R(P)$</td>
<td>20.000</td>
</tr>
<tr>
<td>{ C, S, O }</td>
<td>2.000.000</td>
<td>$R(C) \bowtie R(O) \bowtie R(S)$</td>
<td>20.000</td>
</tr>
<tr>
<td>{ P, S, O }</td>
<td>20.000</td>
<td>$R(P) \bowtie R(S) \bowtie R(O)$</td>
<td>5.000</td>
</tr>
</tbody>
</table>

- **Costs:** Sum of the largest intermediate results
DP: Example /3

- **4. Step**

<table>
<thead>
<tr>
<th>plan</th>
<th>costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(((R(P) \bowtie R(S)) \bowtie R(C)) \bowtie R(O))</td>
<td>5.005.000</td>
</tr>
<tr>
<td>(((R(C) \bowtie R(O)) \bowtie R(P)) \bowtie R(S))</td>
<td>40.000</td>
</tr>
<tr>
<td>(((R(C) \bowtie R(O)) \bowtie R(S)) \bowtie R(P))</td>
<td>2.020.000</td>
</tr>
<tr>
<td>(((R(P) \bowtie R(S)) \bowtie R(O)) \bowtie R(C))</td>
<td>25.000</td>
</tr>
</tbody>
</table>

- **optimal join order**

\(((R(S) \bowtie R(P)) \bowtie R(O)) \bowtie R(C)\)
DP: Pruning

**Input**: join query $Q$ over the relations $r_1, \ldots, r_n$

**Output**: Query plan for $Q$

for $i \leftarrow 1$ to $n$ do
  
  $\text{optPlans}[r_i] \leftarrow \text{access-plans}(r_i)$;

  $\text{prune-plans}(\text{optPlans}[r_i])$;

for $i \leftarrow 2$ to $n$ do
  
  foreach $s \subseteq \{r_1, \ldots, r_n\}$ with $|s| = i$ do
    
    $\text{optPlans}[s] \leftarrow \{\}$;

    foreach $r_k \in s$ do
      
      $\text{optPlans}[s] \leftarrow \text{optPlans}[s] \cup \text{join-plans}(\text{optPlans}[s - \{r_k\}], r_k)$;

    $\text{prune-plans}(\text{optPlans}[s])$;

return $\text{optPlans}[\{r_1, \ldots, r_n\}]$;
Problems with traditional DP Approach

- Serial execution of calculations
- Independent calculations available

\[ |S| = 2: R1-R2; R1-R3; R1-R4 \]

- Current multi-core CPUs offer parallel execution
- GPUs offers more parallelism than CPUs

→ DP-approach must be parallelized to benefit from new hardware trends
Parallelized DP approach

**Idea:** Partition independent calculations [Han et al., 2008]

- **QS**: Qualifier set  
  - **PlanList**: Optimal query execution plan
- $q_1, \ldots, q_4$: Qualifier for relations  
  - $P_1, \ldots, P_4$: QS with 1, \ldots, 4 relation

**Figure:** Taken from [Han et al., 2008]
Parallelized DP approach

**Idea:** Distribute over different threads [Han et al., 2008]

<table>
<thead>
<tr>
<th>P&lt;sub&gt;1&lt;/sub&gt; @ P&lt;sub&gt;3&lt;/sub&gt;</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;2&lt;/sub&gt;,q&lt;sub&gt;3&lt;/sub&gt;)</td>
<td>(q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;2&lt;/sub&gt;,q&lt;sub&gt;4&lt;/sub&gt;)</td>
</tr>
<tr>
<td>(q&lt;sub&gt;2&lt;/sub&gt;,q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;2&lt;/sub&gt;,q&lt;sub&gt;3&lt;/sub&gt;)</td>
<td>(q&lt;sub&gt;2&lt;/sub&gt;,q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;2&lt;/sub&gt;,q&lt;sub&gt;4&lt;/sub&gt;)</td>
</tr>
<tr>
<td>(q&lt;sub&gt;3&lt;/sub&gt;,q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;2&lt;/sub&gt;,q&lt;sub&gt;3&lt;/sub&gt;)</td>
<td>(q&lt;sub&gt;3&lt;/sub&gt;,q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;2&lt;/sub&gt;,q&lt;sub&gt;4&lt;/sub&gt;)</td>
</tr>
<tr>
<td>(q&lt;sub&gt;4&lt;/sub&gt;,q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;2&lt;/sub&gt;,q&lt;sub&gt;3&lt;/sub&gt;)</td>
<td>(q&lt;sub&gt;4&lt;/sub&gt;,q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;2&lt;/sub&gt;,q&lt;sub&gt;4&lt;/sub&gt;)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P&lt;sub&gt;2&lt;/sub&gt; @ P&lt;sub&gt;2&lt;/sub&gt;</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;2&lt;/sub&gt;,q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;2&lt;/sub&gt;)</td>
<td>(q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;2&lt;/sub&gt;,q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;3&lt;/sub&gt;)</td>
</tr>
<tr>
<td>(q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;3&lt;/sub&gt;,q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;2&lt;/sub&gt;)</td>
<td>(q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;3&lt;/sub&gt;,q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;3&lt;/sub&gt;)</td>
</tr>
<tr>
<td>(q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;4&lt;/sub&gt;,q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;2&lt;/sub&gt;)</td>
<td>(q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;4&lt;/sub&gt;,q&lt;sub&gt;1&lt;/sub&gt;,q&lt;sub&gt;3&lt;/sub&gt;)</td>
</tr>
</tbody>
</table>

Figure: Taken from [Han et al., 2008]
Allocation Schemata

• Search space: \[ \left\lfloor \frac{s}{2} \right\rfloor \sum_{smallSZ=1} \left( |P_{smallSZ}| \times |P_{SsmallSZ}| \right) \]

• Total Sum Allocation:
Divide the search space in \( m \) (number of threads) smaller parts and distribute them equally over the \( m \) threads

Figure: Taken from [Han et al., 2008]
Equi-Depth Allocation:
Equally distribute each \((|P_{\text{smallSZ}}| \times |P_{\text{S-smallSZ}}|)\) over all threads.

\[ P_1 \otimes_{\theta} P_3 \]
\[ \begin{align*}
(q_1,q_1q_2q_3) & \quad (q_1,q_1q_2q_4) & \quad (q_1,q_1q_3q_4) \\
(q_2,q_1q_2q_3) & \quad (q_2,q_1q_2q_4) & \quad (q_2,q_1q_3q_4) \\
(q_3,q_1q_2q_3) & \quad (q_3,q_1q_2q_4) & \quad (q_3,q_1q_3q_4) \\
(q_4,q_1q_2q_3) & \quad (q_4,q_1q_2q_4) & \quad (q_4,q_1q_3q_4) \\
\end{align*} \]

thread 1

\[ P_2 \otimes_{\theta} P_2 \]
\[ \begin{align*}
(q_1q_2,q_1q_2) & \quad (q_1q_2,q_1q_3) & \quad (q_1q_2,q_1q_4) \\
(q_1q_3,q_1q_2) & \quad (q_1q_3,q_1q_3) & \quad (q_1q_3,q_1q_4) \\
(q_1q_4,q_1q_2) & \quad (q_1q_4,q_1q_3) & \quad (q_1q_4,q_1q_4) \\
\end{align*} \]

thread 2

Figure: Adopted from [Han et al., 2008]
### Allocation Schemata /3

- **Round-Robin Outer Allocation:**
  Randomly distribute join pairs \((t_i, t'_j)\) to thread \((i \mod m)\)

<table>
<thead>
<tr>
<th>P₁ ⊙ θ P₃</th>
<th>P₂ ⊙ θ P₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q₁₂, q₁₂)</td>
<td>(q₁₂, q₁₂)</td>
</tr>
<tr>
<td>(q₂₂, q₂₂)</td>
<td>(q₂₂, q₂₂)</td>
</tr>
<tr>
<td>(q₃₂, q₃₂)</td>
<td>(q₃₂, q₃₂)</td>
</tr>
<tr>
<td>(q₄₂, q₄₂)</td>
<td>(q₄₂, q₄₂)</td>
</tr>
</tbody>
</table>

**Figure:** Adopted from [Han et al., 2008]
Allocation Schemata /4

- **Round-Robin Inner Allocation:**
  Randomly distribute join pairs \((t_i, t'_j)\) to thread \((j \mod m)\)

\[
P_1 \boxtimes P_3 = \begin{cases} 
(q_1,q_1q_2q_3) & (q_1,q_1q_2q_4) & (q_1,q_1q_3q_4) \\
(q_2,q_1q_2q_3) & (q_2,q_1q_2q_4) & (q_2,q_1q_3q_4) \\
(q_3,q_1q_2q_3) & (q_3,q_1q_2q_4) & (q_3,q_1q_3q_4) \\
(q_4,q_1q_2q_3) & (q_4,q_1q_2q_4) & (q_4,q_1q_3q_4) 
\end{cases}
\]

\[
P_2 \boxtimes P_2 = \begin{cases} 
(q_1q_2,q_1q_2) & (q_1q_2,q_1q_3) & (q_1q_2,q_1q_4) \\
(q_1q_3,q_1q_2) & (q_1q_3,q_1q_3) & (q_1q_3,q_1q_4) \\
(q_1q_4,q_1q_2) & (q_1q_4,q_1q_3) & (q_1q_4,q_1q_4) 
\end{cases}
\]

Figure: Adopted from [Han et al., 2008]
Storage of allocation information

- Store distribution information in the search space description vector (SSDV)

SSDV-Entry: \(<smallSZ, [stOutIdx, stBlkIdx, stBlkOff], [endOutIdx, endBlkIdx, endBlkOff], outInc, inInc]\>

- \(smallSZ\): Identifier for join of \((|P_{smallSZ}| \times |P_{S-smallSZ}|)\)
- \(stOutIdx\): Start index of outer tuple
- \(stBlkIdx\): Start block index
- \(stBlkOff\): Offset of inner tuple within block
- \(endOutIdx\): End index of outer tuple
- \(endBlkIdx\): End block index
- \(endBlkOff\): Offset of end inner tuple within block
- \(outInc\): Step size for outer loop
- \(inInc\): Step size for inner loop
Storage of allocation information - example

1 Block:
- Thread 1 - SSDV-Entry:
  \[
  \{ \langle 1, [1, 1, 1], [4, 1, 1], 1, 1 \rangle, \langle 2, [-1, -1, -1], [-1, -1, -1], 1, 1 \rangle \}
  \]
- Thread 2 - SSDV-Entry:
  \[
  \{ \langle 1, [4, 1, 2], [4, 1, 3], 1, 1 \rangle, \langle 2, [1, 1, 1], [3, 1, 3], 1, 1 \rangle \}
  \]

Figure: Taken from [Han et al., 2008]
Parallelized DP approach

**Input**: A connected query graph with quantifiers $q_1, \cdots, q_N$

**Output**: An optimal bushy join tree

```
for $i=1$ to $N$ do
    $Memo[\{q_i\}] = CreateTableAccessPlan(q_i);$ 
    PrunePlans($Memo[\{q_i\}]$);

for $S=2$ to $N$ do
    $SSDV = AllocateSearchSpace(S,m);$ 
    for $i=1$ to $m$ do 
        threadPool.SubmitJob(MultiplePlanJoin($SSDV[i], S$)); 
        threadPool.sync(); 
        MergeAndPrunePlanPartitions(S);

for $i=1$ to $m$ do 
    threadPool.SubmitJob(BuildSkipVectorArray(i)); 
    threadPool.sync(); 

return $Memo[\{q_1, \cdots, q_n\}]$;
```

**Algorithm 1**: ParallelDPEnum [Han et al., 2008]
Parallelized DP approach

Input: $SSDV, S$

for $i=1$ to $\left\lfloor \frac{s}{2} \right\rfloor$ do
  PlanJoin(SSDV[i],S)

Algorithm 2: MultiplePlanJoin [Han et al., 2008]
Parallelized DP approach

**Input:** ssdvElmt, S

smallSZ = ssdvElmt.smallSZ; largeSZ = S-smallSZ;

for blkIdx = ssdvElmt.stBlkIdx to ssdvElmt.endBlkIdx do

blk = blkIdx-th block in $P_{largeSZ}$;

$\langle stOutIdx, endOutIdx \rangle = \text{GetOuterRange}(ssdvElmt, blkIdx)$;

for $t_o = P_{smallSZ}[stOutIdx]$ to $P_{smallSZ}[endOutIdx]$ step by ssdvElmt.outInc do

$\langle stBlkOff, endBlkOff \rangle = \text{GetOffsetRangeInBlk}(ssdvElmt, blkIdx, offset\ of\ t_o)$;

for $t_i = blk[stBlkOff]$ to $blk[endBlkOff]$ step by ssdvElmt.inInc do

if $t_o.QS \cap t_i.QS \neq \emptyset$ then

continue;

if not ($t_o.QS$ connected to $t_i.QS$) then

continue;

Resulting plans = CreateJoinPlans($t_o, t_i$);

PrunePlans($P_S, ResultingPlans$);

**Algorithm 3:** PlanJoin [Han et al., 2008]
Storage of allocation information - example

Andreas Meister
Advanced Query Optimization
Last Change: June 16, 2014

Figure: Taken from [Han et al., 2008]

- 1 Block:
  - Thread 1 - SSDV-Entry:
    \[\{\langle 1, [1, 1, 1], [4, 1, 1], 1, 1 \rangle, \langle 2, [-1, -1, -1], [-1, -1, -1], 1, 1 \rangle\}\]  
  - Thread 2 - SSDV-Entry:
    \[\{\langle 1, [4, 1, 2], [4, 1, 3], 1, 1 \rangle, \langle 2, [1, 1, 1], [3, 1, 3], 1, 1 \rangle\}\]
Parallelizing problems

- Only a small amount of combinations of different join sets are valid

(a) \# of disjoint filter calls.  
(b) Selectivities.

Figure: Taken from [Han et al., 2008]
Skip Vector Arrays

**Problem:** High number of invalid combination of qualifier sets

**Idea:**
- Increase performance by skipping unnecessary combinations of join sets
- Store additional skipping information to efficiently determine the next join sets
Skip Vector Arrays

<table>
<thead>
<tr>
<th>P₁</th>
<th>QS</th>
<th>PlanList</th>
<th>SV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>q₁</td>
<td>...</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>q₂</td>
<td>...</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>q₃</td>
<td>...</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>q₄</td>
<td>...</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>q₅</td>
<td>...</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>q₆</td>
<td>...</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>q₇</td>
<td>...</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>q₈</td>
<td>...</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P₃</th>
<th>QS</th>
<th>PlanList</th>
<th>SV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>q₁q₂q₃</td>
<td>...</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>q₁q₂q₄</td>
<td>...</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>q₁q₂q₅</td>
<td>...</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>q₁q₂q₆</td>
<td>...</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>q₁q₃q₄</td>
<td>...</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>q₁q₄q₇</td>
<td>...</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>q₁q₄q₈</td>
<td>...</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>q₂q₅q₆</td>
<td>...</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>q₄q₇q₈</td>
<td>...</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure: Taken from [Han et al., 2008]
Skip Vector Arrays

<table>
<thead>
<tr>
<th>QS</th>
<th>PlanList</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_1q_2q_3</td>
<td>...</td>
</tr>
<tr>
<td>q_1q_2q_4</td>
<td>...</td>
</tr>
<tr>
<td>q_1q_2q_5</td>
<td>...</td>
</tr>
<tr>
<td>q_1q_2q_6</td>
<td>...</td>
</tr>
<tr>
<td>q_1q_3q_4</td>
<td>...</td>
</tr>
<tr>
<td>q_1q_4q_7</td>
<td>...</td>
</tr>
<tr>
<td>q_1q_4q_8</td>
<td>...</td>
</tr>
<tr>
<td>q_2q_5q_6</td>
<td>...</td>
</tr>
<tr>
<td>q_4q_7q_8</td>
<td>...</td>
</tr>
</tbody>
</table>

\[ P_3 \]

Equi-depth partitioning & building SVAs

<table>
<thead>
<tr>
<th>QS</th>
<th>PlanList</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_1q_2q_3</td>
<td>...</td>
</tr>
<tr>
<td>q_1q_2q_4</td>
<td>...</td>
</tr>
<tr>
<td>q_1q_3q_4</td>
<td>...</td>
</tr>
</tbody>
</table>

\[ P_{3,1} \]

<table>
<thead>
<tr>
<th>QS</th>
<th>PlanList</th>
<th>SV</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_1q_3q_4</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>q_1q_4q_7</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

\[ P_{3,3} \]

<table>
<thead>
<tr>
<th>QS</th>
<th>PlanList</th>
<th>SV</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_1q_2q_5</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>q_1q_2q_6</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

\[ P_{3,2} \]

<table>
<thead>
<tr>
<th>QS</th>
<th>PlanList</th>
<th>SV</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_1q_4q_8</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>q_2q_5q_6</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>q_4q_7q_8</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

\[ P_{3,4} \]

Figure: Taken from [Han et al., 2008]
Part V

Outlook
Open Research Questions

• How much can the performance of database optimization be increased by GPU-acceleration?

• How much can the performance of databases be increased by GPU-accelerated database optimization?

• What approach provides a better performance?
  • GPU-accelerated database optimization
  • GPU-accelerated query processing

• How much performance is lost during database optimization by the disadvantages of co-processors?
Outline

- Few approaches for parallelizing database optimization
- Few approaches for co-processor-accelerated database optimization
  - Further research is needed on GPU-accelerated database optimization
    - Join-order-optimization: Only parallelized CPU-approaches exist
Invitation

• Your are invited to join our research on database optimization, e.g., in form of:
  • Bachelor or master thesis
  • Scientific team project
    → ”Data Management on new Hardware” next winter term
  • Scientific individual project

• Contact: Andreas Meister
  (andreas.meister@iti.cs.uni-magdeburg.de)
Conclusion

- Execution of query processing
- Logical query optimization
- Physical query optimization
- Search strategies for cost-based plan selection
Thank you for your attention!
References I

Selectivity Estimation of High Dimensional Window Queries via Clustering.

A comparison of selectivity estimators for range queries on metric attributes.
SIGMOD, pages 239–250. ACM.

Parallelizing query optimization.

A First Step Towards GPU-assisted Query Optimization.
In *ADMS*.

*Datenbanken: Implementierungstechniken*.
mitp-Verlag, Bonn, 3 edition.