Construction of Equi-sum\((V, F)\)

1. Collect values of \(A\) and sort tuples increasingly:

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

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

3. Determine \(\beta + 1\) positions in \(\text{list}_A\), 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 \([v_{\text{min}}^i, v_{\text{max}}^i]\) for every bucket \(b_i\) of the values of the \(\text{list}_A\), meaning \(v_{\text{max}}^i = \text{list}_A[i]\) and \(v_{\text{min}}^i = \text{list}_A[i - 1]\)

5. Determine frequency \(f_i\) for every bucket \(b_i\) as number of values between \(v_{\text{min}}^i\) and \(v_{\text{max}}^i\); if for multiple buckets \(v_{\text{max}}^i\) 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 $I(Q, C_i^j)$;

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

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

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

\[ K[(x-X_i)/h] \text{ area} = 1 \]

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:
    \[ K_G(\vec{x}) = (2\pi)^{-\frac{d}{2}} \exp\left(-\frac{1}{2} \vec{x}^T \vec{x}\right) \]
  - 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 occuring 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 \( Q \)

```plaintext
for i ← 1 to n do
    optPlans \[ r_i \] ← \( r_i \);

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

<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(\text{CUSTOMER}) )</td>
</tr>
<tr>
<td>{ PRODUCT }</td>
<td>5.000</td>
<td>( R(\text{PRODUCT}) )</td>
</tr>
<tr>
<td>{ SUPPLIER }</td>
<td>100</td>
<td>( R(\text{SUPPLIER}) )</td>
</tr>
<tr>
<td>{ ORDER }</td>
<td>20.000</td>
<td>( R(\text{ORDER}) )</td>
</tr>
</tbody>
</table>

### 1. Step

- \( R(\text{CUSTOMER}) \)
- \( R(\text{PRODUCT}) \)
- \( R(\text{SUPPLIER}) \)
- \( R(\text{ORDER}) \)

### 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(\text{K}) \Join R(\text{P}) )</td>
</tr>
<tr>
<td>{ C, S }</td>
<td>100.000</td>
<td>( R(\text{K}) \Join R(\text{L}) )</td>
</tr>
<tr>
<td>{ C, O }</td>
<td>20.000</td>
<td>( R(\text{K}) \Join R(\text{B}) )</td>
</tr>
<tr>
<td>{ P, S }</td>
<td>5.000</td>
<td>( R(\text{P}) \Join R(\text{L}) )</td>
</tr>
<tr>
<td>{ P, O }</td>
<td>20.000</td>
<td>( R(\text{P}) \Join R(\text{B}) )</td>
</tr>
<tr>
<td>{ S, O }</td>
<td>2.000.000</td>
<td>( R(\text{L}) \Join R(\text{B}) )</td>
</tr>
</tbody>
</table>
DP: Example /2

- 3. Step

<table>
<thead>
<tr>
<th>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) ⊙ R(S)) ⊙ R(C)</td>
<td>5.000</td>
</tr>
<tr>
<td>{C, P, O}</td>
<td>20.000</td>
<td>(R(C) ⊙ R(O)) ⊙ R(P)</td>
<td>20.000</td>
</tr>
<tr>
<td>{C, S, O}</td>
<td>2.000.000</td>
<td>(R(C) ⊙ R(O)) ⊙ R(S)</td>
<td>20.000</td>
</tr>
<tr>
<td>{P, S, O}</td>
<td>20.000</td>
<td>(R(P) ⊙ R(S)) ⊙ 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

\(((R(S) \bowtie R(P)) \bowtie R(O)) \bowtie R(C)\)
Input: join query $Q$ over the relations $r_1, \ldots, r_n$
Output: Query plan for $Q$

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

\[
\text{for } i \leftarrow 2 \text{ to } n \text{ do}
\]
\[
\quad \text{foreach } s \subseteq \{r_1, \ldots, r_n\} \text{ with } |s| = i \text{ do}
\]
\[
\quad \quad \text{optPlans } [s] \leftarrow \{\};
\]
\[
\quad \quad \text{foreach } r_k \in s \text{ do}
\]
\[
\quad \quad \quad \text{optPlans } [s] \leftarrow \text{optPlans } [s] \cup \text{join-plans(optPlans } [s - \{r_k\}], r_k) ;
\]
\[
\quad \quad \quad \text{prune-plans(optPlans } [s]);
\]

return 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

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

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

**Plan Relation for G**

<table>
<thead>
<tr>
<th>QS</th>
<th>PlanList</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>QEP₁, QEP₂</td>
</tr>
<tr>
<td>q₂</td>
<td>QEP₃</td>
</tr>
<tr>
<td>q₃</td>
<td>QEP₄, QEP₈</td>
</tr>
<tr>
<td>q₄</td>
<td>QEP₉, QEP₁₀</td>
</tr>
<tr>
<td>q₁q₂</td>
<td>QEP₁₁</td>
</tr>
<tr>
<td>q₁q₃</td>
<td>QEP₁₇</td>
</tr>
<tr>
<td>q₁q₄</td>
<td>QEP₂₁</td>
</tr>
<tr>
<td>q₁q₂q₃</td>
<td>QEP₂₇, QEP₃₀</td>
</tr>
<tr>
<td>q₁q₂q₄</td>
<td>QEP₃₁</td>
</tr>
<tr>
<td>q₁q₃q₄</td>
<td>QEP₃₂, QEP₃₃</td>
</tr>
<tr>
<td>q₁q₂q₃q₄</td>
<td>QEP₃₅</td>
</tr>
</tbody>
</table>

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

- **QS:** Qualifier set | **PlanList:** Optimal query execution plan
- **q₁, . . . , q₄:** Qualifier for relations | **P₁, . . . , P₄:** QS with 1, . . . 4 relation
Parallelized DP approach

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

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

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

• **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]
Allocation Schemata /2

- **Equi-Depth Allocation:**
  Equally distribute each \((|P_{smallSZ}| \times |P_{S-smallSZ}|)\) over all threads.

\[
\begin{align*}
P_1 \bowtie_{\theta} P_3 & \left\{ \begin{array}{c}
(q_1, q_1, q_2, q_3) & (q_1, q_1, q_2, q_4) & (q_1, q_1, q_3, q_4) \\
(q_2, q_1, q_2, q_3) & (q_2, q_1, q_2, q_4) & (q_2, q_1, q_3, q_4) \\
(q_3, q_1, q_2, q_3) & (q_3, q_1, q_2, q_4) & (q_3, q_1, q_3, q_4) \\
(q_4, q_1, q_2, q_3) & (q_4, q_1, q_2, q_4) & (q_4, q_1, q_3, q_4)
\end{array} \right. \\
\end{align*}
\]

\[
\begin{align*}
P_2 \bowtie_{\theta} P_2 & \left\{ \begin{array}{c}
(q_1, q_2, q_1, q_2) & (q_1, q_2, q_1, q_3) & (q_1, q_2, q_1, q_4) \\
(q_1, q_3, q_1, q_2) & (q_1, q_3, q_1, q_3) & (q_1, q_3, q_1, q_4) \\
(q_1, q_4, q_1, q_2) & (q_1, q_4, q_1, q_3) & (q_1, q_4, q_1, q_4)
\end{array} \right. \\
\end{align*}
\]

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

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

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

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

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

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

\[
\begin{align*}
P_1 \bowtie_{\theta} P_3 &= \{ \\
\begin{array}{c}
(q_1,q_1q_2q_3) \\
(q_2,q_1q_2q_3) \\
(q_3,q_1q_2q_3) \\
(q_4,q_1q_2q_3) \\
(q_1,q_1q_2q_4) \\
(q_2,q_1q_2q_4) \\
(q_3,q_1q_2q_4) \\
(q_4,q_1q_2q_4) \\
\end{array} \\
\end{align*}
\]

\[
\begin{align*}
P_2 \bowtie_{\theta} P_2 &= \{ \\
\begin{array}{c}
(q_1q_2,q_1q_2) \\
(q_1q_3,q_1q_2) \\
(q_1q_4,q_1q_2) \\
(q_1q_2,q_1q_3) \\
(q_1q_3,q_1q_3) \\
(q_1q_4,q_1q_3) \\
(q_1q_2,q_1q_4) \\
(q_1q_3,q_1q_4) \\
(q_1q_4,q_1q_4) \\
\end{array} \\
\end{align*}
\]

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

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

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

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

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

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

for $S=2$ to $N$ do
    SSDVs = AllocateSearchSpace($S,m$);
    for $i=1$ to $m$ do
        threadPool.SubmitJob(MultiplePlanJoin(SSDV$s[i], S))
        threadPool.sync();
    MergeAndPrunePlanPartitions($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 = GetOuterRange(ssdvElmt, blkIdx);
  for t_o = P_{smallSZ}[stOutIdx] to P_{smallSZ}[endOutIdx] step by ssdvElmt.outInc do
    \langle stBlkOff, endBlkOff \rangle =
    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

- 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]
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>P_1</th>
<th>QS</th>
<th>PlanList</th>
<th>SV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>q_1</td>
<td>...</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>q_2</td>
<td>...</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>q_3</td>
<td>...</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>q_4</td>
<td>...</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>q_5</td>
<td>...</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>q_6</td>
<td>...</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>q_7</td>
<td>...</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>q_8</td>
<td>...</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P_3</th>
<th>QS</th>
<th>PlanList</th>
<th>SV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>q_1</td>
<td>q_2  q_3</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>q_1</td>
<td>q_2  q_4</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>q_1</td>
<td>q_2  q_5</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>q_1</td>
<td>q_2  q_6</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>q_1</td>
<td>q_3  q_4</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>q_1</td>
<td>q_4  q_7</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>q_1</td>
<td>q_4  q_8</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>q_2</td>
<td>q_5  q_6</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>q_4</td>
<td>q_7  q_8</td>
<td>10</td>
</tr>
</tbody>
</table>

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

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 exists
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


