Demonstrating Transfer-Efficient Sample Maintenance on Graphics Cards
Martin Kiefer, Max Heimel, Volker Markl

Setting: GPU-assisted Selectivity Estimation

- Keep statistical model of the data on the GPU.
- Compute selectivity estimates based on this model.

- Enables larger models & more complex methods.
- Produces better estimates, resulting in better query plans!
- Indirect method of using GPUs to accelerate database systems that does not suffer from data transfer / data placement / etc...

- We use Kernel Density Estimation (KDE):
  - State-of-the-art estimation method from statistics literature.
  - Converges faster to true distribution than MD histograms.

- Computes selectivities based on a data sample.

- Step 1: Draw a uniform sample \( S = (z_1, ..., z_N) \) from \( R \).

- Step 2: Center Kernel functions \( K_r \) around sample points.

- Kernel functions are basically symmetric probability density functions.

- Step 3: Estimate the density at a point by averaging the Kernels.

- \( \hat{p}_H(x) = \frac{1}{N} \sum_{z \in S} K_r(x-z) \)

- Produces high-quality estimates, but is expensive!
  - Luckily, it can be efficiently parallelized and mapped to a GPU!
  - Using a GPU enables a superior selectivity estimation method!

Problem: KDE Model Maintenance

- What happens when the database changes?
  - KDE estimates are computed from a data sample.

- If the user changes data, the sample becomes stale, resulting in incorrect estimates & degraded query plan quality.

- We need to maintain the sample wrt database updates!

- Sample maintenance is a well-known problem.
  - However: Our sample is located on a graphics card!

- Maintenance has to be applied over the (limited) PCI Express bus.

- We have to make sure that we do not saturate the bus!

- The maintenance algorithm must be transfer-efficient!

- Good sample quality, while being economical with data transfers.

Correlated Acceptance/Rejection (CAR)

- State-of-the-art sample maintenance algorithm, dealing with insertions, updates and deletions.

- Is CAR a transfer-efficient algorithm?

The Karma Score: Error-driven sample maintenance

- We know the query’s selectivity \( p(x) \) after execution.

- In KDE, every sample point contributes probability mass independent of the others.

- We can quantify the impact on the estimation error for every sample point after the query!

- Compute the "adjusted estimate" for every point by removing the point’s contribution:

- \( \hat{p}^{(i)}_H(x) = \hat{p}_H(x) - \hat{p}^{(i)}_H(x) \)

- For "bad" points, this adjusted estimate should be closer to the true selectivity. Vice versa for "good" points.

- Gives an indication whether points are "helpful".

- We aggregate the Karma scores over subsequent queries.

- Gives an indication how "valuable" a sample point is overall.

- Cumulative Karma is limited to \( K_{max} \) to accelerate sample convergence.

Periodic Karma Replacement (PKR)

- Every n queries, replace point with worst Karma.

- GPU implementation:
  - A parallel reduction is run to compute the minimum.
  - Position of worst point is transferred to the host.
  - A newly sampled point is transferred back to replace it.

Triggered Karma Replacement (TKR)

- Replace points whose Karma is worse than \#.

- GPU implementation:
  - Offloading sample points is computed in parallel.
  - Bitmap identifying outdated points is transferred to host.
  - Newly sampled points are transferred to device.

Demo results

- Cumulative transfer time:

- Conclusion:

- GPU-assisted selectivity estimation requires sample maintenance!

- Karma-Based Sample Maintenance can dramatically reduce data transfers while offering quality comparable to CAR! 

- Generally, TKR is preferable over PKR (better quality, more reactive).

- However: Depending on workload properties, CAR is still preferable.

- Source code available at: http://goog.de/oGQNaD

Demo workload: Moving clusters

- Clusters are created along a line.
  - Moving query target, changing data:
    - We start with three clusters.
    - Clusters are queried while the next cluster is inserted.
    - Periodically, the oldest cluster is deleted.
    - Query focus shifts to the most recent cluster.
  - Ten clusters are created and deleted at runtime.

- Workload options:
  - Points per cluster
  - Cluster spread
  - Algorithm hyperparameters

- Models evolving database with an archive.