Adaptive Query Processing on Vectorized Hardware

2nd TAB-Talk, 30.10.2020, Johannes Pietrzyk
Research Hypothesis

Roles enrich the process of designing and developing adaptive software systems.
General Overview (State-of-the-Art)

Optimization Techniques

- Share Data:
  - Share base data through specialized operators
  - Joint usage of intermediate data across multiple queries through plan-adjustments

Scalar (SISD)

Query-at-a-Time (Pipelining, Batch at a Time, …)

Scan-Sharing

Multi-Query-Optimization

Data Sharing

Base data

Intermediate data

Single Query

Multiple Query


From Scalar to Vectorized Processing

Example:

\[
\begin{array}{c}
\text{A} + \text{B} = \text{C}
\end{array}
\]

**Single Instruction Single Data (SISD)**

“Scalar”

**Single Instruction Multiple Data (SIMD)**

“Vectorized”
Optimization Techniques

- Share Data:
  - Share base data through specialized operators
  - Joint usage of intermediate data across queries

- Vectorization:
  - One instruction on multiple data at once
  - Significant speedups for single queries

Data-Level Parallelism

Vectorization (SIMD)

Scalar (SISD)

Shared Vectors

Use vector registers as HW resource for sharing data.
Use SIMD primitives for sharing computations.
Use roles to adapt both aspects at runtime.

Data Sharing

Single Query

Multiple Query

Base data

Intermediate data

None

Scan-Sharing

Multi-Query-Optimization

Query-at-a-Time (Pipelining, Batch at a Time, ...)

Base data

Intermediate data
Operators in DBMS

Database Operators

- $\sigma$: SELECT
- $\pi$: PROJECT
- $\gamma$: GROUP
- $\Sigma$: SUM
- $+$: CALC
- $\cup/\cap$: UNION/INTERSECT
- $\bowtie$: JOIN

Research Prototype – MorphStore

- Vectorization and Compression as first class citizen

<table>
<thead>
<tr>
<th>Operator</th>
<th>Load-Phase</th>
<th>Compute-Phase</th>
<th>Store-Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma$</td>
<td>load</td>
<td>compare</td>
<td>store</td>
</tr>
<tr>
<td>$\pi$</td>
<td>load, gather</td>
<td>/</td>
<td>store</td>
</tr>
<tr>
<td>$+$</td>
<td>load</td>
<td>add, mul, div</td>
<td>store</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>load</td>
<td>+</td>
<td>store</td>
</tr>
<tr>
<td>$\gamma/\bowtie$</td>
<td>load, gather</td>
<td>$+$, $\sigma$</td>
<td>store, scatter</td>
</tr>
</tbody>
</table>

Manageable number of operators

Operators have primitives in common
An Example

Inspired by SSB Query 1.1

- Varying predicates for filters
- Different aggregation functions
- Base data can be shared
- No sharing of intermediates possible
- BUT: Sharing on primitive level possible

```
SELECT MAX(lo.discount)  
FROM lineorder as lo, date as d  
WHERE lo.orderdate = d.datekey  
AND d.year = 1993

SELECT MIN(lo.discount)  
FROM lineorder as lo, date as d  
WHERE lo.orderdate = d.datekey  
AND d.year = 1994
```
Sharing on primitive level – using vectorization

**Filter-Operator**
- **Primitive**-sharing (compare) and **Data**-sharing possible
- Query-dependent (predicate) initialization phase
- Query-dependent (operators) load and store phase
  - Load-Phase: 1 Column
  - Store-Phase: 2 Columns

**Join-Operator**
- **Primitive**-sharing but no **Data**-sharing possible
- Query-dependent load and store phase
  - Load-Phase: 2 Columns
  - Store-Phase: 2 Columns
- Even the compare primitive can be shared across operators

**Sharing potential**
- Depending on
  - Hardware
  - Incoming queries:
    - Accessed columns
    - Common operators
    - Reusable intermediates
    - Selectivity of operators / data properties

Sharing has to be adapted at workload and query runtime
First Evaluation of Shared Vectors*

Sharing potential depending on Query and Column count

- Number of avail. vector lanes depends on vector size and value size

Setup

- Simple query consisting of two shared operators
- Systematic evaluation of impact of vector lane, query and column count

Platform

- Intel Xeon Gold 6126
- 92 GB DDR4
- 3.7 Ghz max. core frequency
- 64-bit CentOs (7.7.1908)

*published at DaMoN’20
First Evaluation of Shared Vectors

Key Findings:
- If work can be shared, Shared Vectors are beneficial
- Bigger vector registers lead to better speedup
- Value Sizes have significant impact on speedup
- The more queries, the higher the speedup
- The more columns, the lower the speedup

Multiple tuning knobs available

High demand on a flexible, adaptive solution
Mapping System Design to CROM Model

Adaption to

- Hardware
  - Available vector sizes
- Processed data
  - Data size
- Workload-Structure

Roles really enrich the process of designing our adaptive system.
Summary

Conclusion

▪ Presented a novel approach to share data and computations using vectorized hardware (using TVL*)
▪ First evaluations published at DaMoN’20
  - Systematically evaluated impact of contexts
  - Focused on input preprocessing
  - First results seem very promising

▪ Proposed concept rely on frequent adaption
▪ Seems to perfectly fit for Roles as a technique for runtime adaptivity

Future Work

▪ Open question:
  - How to realize and limit possible role transitions for IPP and OPP at runtime

▪ Implementation of:
  - Generalized operator model
  - Roles and compartments to enable changes of input and output processing during operator runtime
  - Consistent role transitions

▪ Evaluation of:
  - Impact of more complex queries
  - Role transition costs at runtime

▪ Design of a cost-based optimizer to determine which roles to apply
  - Using an Event-Condition-Action Loop

Adaptive Query Processing on Vectorized Hardware

2\textsuperscript{nd} TAB-Talk, 30.10.2020, Johannes Pietrzyk
List of relevant publications


