OPTIMIZING GROUP-BY AND AGGREGATION USING GPU-CPU CO-PROCESSING

Diego Tomé
CWI
Amsterdam, The Netherlands
diego.tome@cwi.nl

Tim Gubner
CWI
Amsterdam, The Netherlands
tim.gubner@cwi.nl

Mark Raasveldt*
CWI
Amsterdam, The Netherlands
m.raasveldt@cwi.nl

Eyal Rozenberg
CWI
Amsterdam, The Netherlands
e.rozenberg@cwi.nl

Peter Boncz
CWI
Amsterdam, The Netherlands
peter.boncz@cwi.nl
OPTIMIZING GROUP-BY AND AGGREGATION USING GPU-CPU CO-PROCESSING
GPU AS ACCELERATOR FOR DATABASE SYSTEMS

- Promising: Order-of-magnitude performance gains
  - But only if the input data is on GPU Global Memory
- Unclear how to integrate into full system
- Entire dataset does not fit into GPU global memory
CHALLENGES IN THE GPU ARCHITECTURE

if (expression) do something else do something else
GPU-CPU CO-PROCESSING SYSTEM

- GPU only system could be worse in many cases
- Solution: GPU-CPU co-processing system
  - Ideally never slower than CPU-only systems
  - Automatically decide between CPU and GPU

WHAT SHOULD GPU-CPU CO-PROCESSING SYSTEM LOOK LIKE?
GPU-CPU CO-PROCESSING SYSTEM

- We take a small step into this design space
- Focus on GROUP BY and AGGREGATION in this system
- In the context of the familiar OLAP scenario: TPC-H Q1
- Investigate the design space and look into trade-offs
  - Without assuming data is in GPU memory!
SELECT
  l_returnflag,
  l_linestatus,
  sum(l_quantity) as sum_qty,
  sum(l_extendedprice) as sum_base_price,
  sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
  sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
  avg(l_quantity) as avg_qty,
  avg(l_extendedprice) as avg_price,
  avg(l_discount) as avg_disc,
  count(*) as count_order
FROM
  lineitem
WHERE
  l_shipdate <= date '1998-12-01' - interval '90' day
GROUP BY
  l_returnflag,
  l_linestatus
ORDER BY
  l_returnflag,
  l_linestatus;
OPTIMIZING GROUP-BY AND AGGREGATION USING GPU-CPU CO-PROCESSING

DESIGN SPACE
**DESIGN SPACE – COMPRESSION**

<table>
<thead>
<tr>
<th>Column</th>
<th>SQL Type</th>
<th>Uncompressed Size (bits)</th>
<th>Method</th>
<th>Compressed Size (bits)</th>
<th>Optimum Compressed Size (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_quantity</td>
<td>INTEGER</td>
<td>32</td>
<td>Null-Suppression</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>l_extendedprice</td>
<td>DECIMAL(15,2)</td>
<td>64</td>
<td>Null-Suppression</td>
<td>32</td>
<td>21</td>
</tr>
<tr>
<td>l_discount</td>
<td>DECIMAL(15,2)</td>
<td>64</td>
<td>Null-Suppression</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>l_tax</td>
<td>DECIMAL(15,2)</td>
<td>64</td>
<td>Null-Suppression</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>l_shipdate</td>
<td>DATE</td>
<td>32</td>
<td>Frame-of-Reference + NS</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>l_returnflag</td>
<td>VARCHAR(1)</td>
<td>8</td>
<td>Dictionary</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>l_linestatus</td>
<td>VARCHAR(1)</td>
<td>8</td>
<td>Dictionary</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total per Tuple (as bytes)</td>
<td></td>
<td></td>
<td></td>
<td>272</td>
<td>75</td>
</tr>
</tbody>
</table>

Table 1: Compression schemes for the columns in lineitem used in TPC-H Query 1

**SUB-OPTIMALLY IS 27% OF THE ORIGINAL TUPLE**

**FROM SF = 100 WE TRANSFER 5.625 GB OF DATA**

**OPTIMUM IS 18% OF THE ORIGINAL TUPLE**

---

Database Compression on Graphics Processors
Fang, Wenbin and He, Bingsheng and Luo, Qiong
Proceedings of the VLDB Endowment, 2010

Faster Across the PCIe Bus: A GPU Library
Rozenberg, Eyal and Boncz, Peter
Damon@SIGMOD, 2017
DESIGN SPACE - HASHING VS. SORTING

- Hash Function on group keys
- Sorting data, by order of the group keys
- Previous work shows that hash-based performs better
- Sorting is better only with a very high amount of groups
DESIGN SPACE – HASH TABLE DESIGN

- Collision resolution
  - Following a chain to find a position
  - Continued probing of subsequent table cells
  - Re-applying the hash function
- Collision Avoidance
  - Use of an injective function as hash

Efficient Hash Tables on the Gpu
Alcantara, Dan Anthony Feliciano
PhD thesis University of California at Davis, 2011

Hash, displace, and compress
Belazzougui, D. et. al.
Springer, 2009
DESIGN SPACE – HASH TABLE PLACEMENT

- Caches are not coherent
- Registers are bigger than Shared memory
- Shared memory is organized into 32 interleaved banks
- Excessive use of Registers is translated to Global memory access

EXPLORE THE WHOLE HIERARCHY
How Data parallel Co-processing Works?
How Data parallel Co-processing Works?

Tuples

Result

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

GPU

CPU
How Filter Pre-computation Works?

GPU

Column

Apply the Filter in Parallel

CPU

Bitmap

1 1 0 1 0 1 0 1 0 1 0 0
How Filter Pre-computation Works?

GPU

Transfer the Bitmap

CPU

Column
How Filter Pre-computation Works?

1 0 1 0 1 0 1 0 1 0

Result

1 0 1 0 1 0 1 0 1 0
## DESIGN SPACE - SUMMARY

<table>
<thead>
<tr>
<th>Design Space</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compression</td>
<td>27% of the original tuple = 5.625 GB (SF 100)</td>
</tr>
<tr>
<td>Group-by and aggregation</td>
<td>Hash based implementation</td>
</tr>
<tr>
<td>Hash table design</td>
<td>Collision avoidance with a perfect hash function</td>
</tr>
<tr>
<td>Hash table placement</td>
<td>Explore the whole GPU hierarchy</td>
</tr>
<tr>
<td>Co-processing</td>
<td>Data Parallel and Filter Pre-computation</td>
</tr>
</tbody>
</table>
OPTIMIZING GROUP-BY AND AGGREGATION USING GPU-CPU CO-PROCESSING

EXPERIMENTS
EXPERIMENTS - IMPLEMENTATION FLAVORS

- **Global** - A single hash table in Global GPU memory
- **Local** - A hash table for each working thread in local memory
- **Shared_memory** - A hash table for each working thread in shared memory
- **In_registers** - A separate hash table for each working thread in its registers
- **In_register_per_thread** - A single table cell per working thread in register
EXPERIMENTS - CO-PROCESSING

- CPU implementation adapted from previous work
- We introduce Morsel-driven model of parallelized execution
- NUMA locality (One single socket)
- Co-processing with Filter Pre-computation and Data Parallelism.
RESULTS TPC-H QUERY 01 - SF 100

Overlapping of computation and transfer (Asynchronous)

Global is penalized with atomic operations
RESULTS TPC-H QUERY 01 - SF 100

Slightly speedup performance for local, shared and register

Time to filter on CPU is compensated by bitmap transfer
Data parallel yields a 30% speedup over the best GPU-only implementation.
RESULTS TPC-H QUERY 01 - SF 100

Slows computation down due to the filter/transfer overhead
## RELATED WORK TPC-H QUERY 01

<table>
<thead>
<tr>
<th>System</th>
<th>CPU/GPU</th>
<th>SF 100 (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Fox</td>
<td>GPU</td>
<td>33000</td>
</tr>
<tr>
<td>Ocelot</td>
<td>GPU</td>
<td>3470</td>
</tr>
<tr>
<td>Voodoo</td>
<td>GPU</td>
<td>2940</td>
</tr>
<tr>
<td>AXE/GPU</td>
<td>GPU</td>
<td>2580</td>
</tr>
<tr>
<td>CoGaDB</td>
<td>GPU</td>
<td>3500</td>
</tr>
<tr>
<td>(This work)</td>
<td>GPU</td>
<td>536</td>
</tr>
<tr>
<td>PCI I/O Compressed</td>
<td>GPU</td>
<td>493</td>
</tr>
<tr>
<td>Red Fox</td>
<td>CPU</td>
<td>2.7E+06</td>
</tr>
<tr>
<td>Voodoo</td>
<td>CPU</td>
<td>1620</td>
</tr>
<tr>
<td>Hyper</td>
<td>CPU</td>
<td>1200</td>
</tr>
<tr>
<td>AXE/CPU</td>
<td>CPU</td>
<td>8380</td>
</tr>
<tr>
<td>(This work)</td>
<td>CPU</td>
<td>791</td>
</tr>
<tr>
<td>(This work)</td>
<td>CPU+GPU</td>
<td>385</td>
</tr>
</tbody>
</table>
OPTIMIZING GROUP-BY AND AGGREGATION USING GPU-CPU CO-PROCESSING

CONCLUSION
CONCLUSION

GPU-CPU Co-processing achieved significantly improved performance compared to a CPU-only baseline and GPU-only implementation.

Future work includes: larger number of groups, non-uniform data distribution, JIT compilation and a co-processing framework.

https://github.com/diegomestre2/tpchQ01_GPU