Database Compression on Graphics Processors
Seminar: Techniques for implementing main memory database systems

Maximilian Springer

26.11.2018
Why query processing on GPUs?
Why query processing on GPUs?

GPUs are very fast for **simple, highly-parallelizable, branchless workflows.**
Why query processing on GPUs? - Raw specifications

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>18</td>
<td>4352</td>
</tr>
</tbody>
</table>
Why query processing on GPUs? - Raw specifications

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>18</td>
<td>4352</td>
</tr>
<tr>
<td>Max Single Core Speed</td>
<td>4.20 GHz</td>
<td>1.635 GHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Why query processing on GPUs? - Raw specifications

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cores</strong></td>
<td>18</td>
<td>4352</td>
</tr>
<tr>
<td><strong>Max Single Core Speed</strong></td>
<td>4.20 GHz</td>
<td>1.635 GHz</td>
</tr>
<tr>
<td><strong>Max Mem Bandwidth</strong></td>
<td>85.2 GB/s</td>
<td>616 GB/s</td>
</tr>
</tbody>
</table>

Maximilian Springer

Database Compression on Graphics Processors

26.11.2018 4 / 34
Why query processing on GPUs? - Raw specifications

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>18</td>
<td>4352</td>
</tr>
<tr>
<td>Max Single Core Speed</td>
<td>4.20 GHz</td>
<td>1.635 GHz</td>
</tr>
<tr>
<td>Max Mem Bandwidth</td>
<td>85.2 GB/s</td>
<td>616 GB/s</td>
</tr>
</tbody>
</table>
Why query processing on GPUs? - Raw specifications

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cores</strong></td>
<td>18</td>
<td>4352</td>
</tr>
<tr>
<td><strong>Max Single Core Speed</strong></td>
<td>4.20 GHz</td>
<td>1.635 GHz</td>
</tr>
<tr>
<td><strong>Max Mem Bandwidth</strong></td>
<td>85.2 GB/s</td>
<td>616 GB/s</td>
</tr>
<tr>
<td><strong>Max Mem Size</strong></td>
<td>128 GB</td>
<td>11 GB</td>
</tr>
</tbody>
</table>
Why query processing on GPUs? - Drawbacks

- CPU ↔ GPU connection is slow
Why query processing on GPUs? - Drawbacks

- Drawbacks:

CPU

RAM

85.2 GB/s

GPU

GPU RAM

PCIe 3.0 x16 (16GB/s)

616 GB/s
Why query processing on GPUs? - Drawbacks

- CPU ↔ GPU connection is slow
- Single core speed is slower
Why query processing on GPUs? - Drawbacks

- CPU ⇔ GPU connection is slow
- Single core speed is slower
- GPU code should be *branchless, simple* and *non-diverging*
Why query processing on GPUs? - Solutions to drawbacks

- Reduce copy cost necessary for query coprocessing
Why query processing on GPUs? - Solutions to drawbacks

- Reduce copy cost necessary for query coprocessing
- Idea: *calculation speed* ↓ *copy speed* ↑
Why query processing on GPUs? - Solutions to drawbacks

- Reduce copy cost necessary for query coprocessing
- Idea: **calculation speed** ↓ **copy speed** ↑
  - Transfer compressed data onto the GPU
Compression on GPU

- Focus on easy compression techniques
Compression on GPU

- Focus on easy compression techniques
  - Prevent branching
Compression on GPU

- Focus on easy compression techniques
  - Prevent branching
  - Simple code
Compression on GPU

- Focus on easy compression techniques
  - Prevent branching
  - Simple code
  - Less diverging between threads
Compression on GPU

- Focus on easy compression techniques
  - Prevent branching
  - Simple code
  - Less diverging between threads
- Chain them together to achieve better ratios
## Compression techniques [5]

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS</td>
<td>Null Suppression with Fixed Length</td>
</tr>
<tr>
<td>NSV</td>
<td>Null Suppression with Variable Length</td>
</tr>
<tr>
<td>DICT</td>
<td>Dictionary</td>
</tr>
<tr>
<td>BITMAP</td>
<td>Bitmap</td>
</tr>
<tr>
<td>RLE</td>
<td>Run-Length-Encoding</td>
</tr>
<tr>
<td>DELTA</td>
<td>Delta</td>
</tr>
</tbody>
</table>
Cascaded compression

RLE, \[[[\text{DELTA}, \text{NS}] \mid \text{NS}]]
Cascaded compression

RLE, [[[DELTA, NS] | NS]

↓
Cascaded compression

\[ \text{RLE}, \left[ \left[ \text{DELTA}, \text{NS} \right], \text{NS} \right] \]

\[ \downarrow \]

\[ \text{RLE} \]
Cascaded compression

\[
\text{RLE, } [[\text{DELTA, NS}] \mid \text{NS}]\]

\[
\Downarrow
\]

\[
\text{RLE} \quad \text{→} \quad \text{(values, column)}
\]
Cascaded compression

RLE \rightarrow (\text{values, column})

NS \leftarrow \text{DELTA}
Cascaded compression

\[
\text{RLE} \rightarrow \text{(values, column)}
\]

\[
\text{NS} \quad \text{DELTA} \quad \text{NS}
\]
Architecture
In CUDA the thread model is a grid.
GPU Architecture[6]

 gridSize.x = 16
In CUDA the thread model is a grid of blocks
GPU Architecture[6]

gridDim.x = 16

blockDim.x = 16

blockIdx.x = 2
In CUDA the thread model is a grid of blocks of threads.
GPU Architecture[6]

blockIdx.x = 2

threadIdx.x = 5
GPU Architecture[6]

- Threads of a block organized in *warps* of 32 threads
GPU Architecture[6]

- Threads of a block organized in *warps* of 32 threads
- *Warps* executed on GPU processors
GPU Architecture[6]

- Threads of a block organized in *warps* of 32 threads
- *Warps* executed on GPU processors
- *Warp* code similar to *SIMD*
CUDA usage[6]

- Standard way: Compile CUDA code that can be invoked from host

- JIT way: Use LLVM to compile CUDA on the fly

- Allows optimization of compression schema code

- Simplify combination of compression code

- Allow custom optimization
CUDA usage[6]

- Standard way: Compile CUDA code that can be invoked from host
- JIT way: Use LLVM to compile CUDA on the fly[7][8]
CUDA usage[6]

- Standard way: Compile CUDA code that can be invoked from host
- JIT way: Use LLVM to compile CUDA on the fly[7][8]
  - Allows optimization of compression schema code
CUDA usage[6]

- Standard way: Compile CUDA code that can be invoked from host
- JIT way: Use LLVM to compile CUDA on the fly[7][8]
  - Allows optimization of compression schema code
  - Simplify combination of compression code
CUDA usage [6]

- Standard way: Compile CUDA code that can be invoked from host
- JIT way: Use LLVM to compile CUDA on the fly [7][8]
  - Allows optimization of compression schema code
  - Simplify combination of compression code
  - Allow custom optimization
Primitives
Primitives

- Map
  - One thread per element in the array executing the map function

- Scatter
  - One thread per write position

- Prefix Sum
  - Divide and conquer workload utilizing warp instructions and blocks

Maximilian Springer
Database Compression on Graphics Processors
26.11.2018
Primitives

- Map
  - One thread per element in the array executing the map function
Primitives

- Map
  - One thread per element in the array executing the map function
- Scatter
Primitives

- **Map**
  - One thread per element in the array executing the map function
- **Scatter**
  - One thread per write position
Primitives

- Map
  - One thread per element in the array executing the map function
- Scatter
  - One thread per write position
- Prefix Sum[9]
Primitives

- **Map**
  - One thread per element in the array executing the map function

- **Scatter**
  - One thread per write position

- **Prefix Sum[9]**
  - Divide and conquer workload utilizing warp instructions and blocks
Dynamic Parallelism[6]
Dynamic Parallelism[6]

- Allows the launch of kernels of different size from another kernel
Dynamic Parallelism[6]

- Allows the launch of kernels of different size from another kernel
- Allows modelling of primitives via kernels
Dynamic Parallelism[6]

- Allows the launch of kernels of different size from another kernel
- Allows modelling of primitives via kernels
- Minimize wasted GPU processor time
Optimizations
Optimizations

- Merge kernels
Optimizations

- Merge kernels
- Create LLVM Pass[10] to check whether kernel invocation and kernels allow this
Optimizations

- Merge kernels
- Create LLVM Pass[10] to check whether kernel invocation and kernels allow this
- Combine frequents combined primitives together
l_partkey(RLE(6.6%))

l_partkey column from TPC-H (scale = 10, 60M rows)
l_shipmode(DICT(50%))

l_shipmode column from TPC-H (scale = 10, 60M rows)
l_quantity(NS(75%))

l_quantity column from TPC-H (scale = 10, 60M rows)

![Bar chart showing throughput in GB/s for different GPU models.]

- Fang et al. (GTX 295)
- Me (GTX 970)


References II


Thank You!
Questions?