Toward GPU Accelerated Data Stream Processing

Marcus Pinnecke, David Broneske and Gunter Saake
University of Magdeburg, Germany

May 27, 2015
Background and Motivation

Fundamentals, Windowing, GPU Acceleration in DBMS/SPS
Examples

- **System Monitoring and Fraud Prevention** — Log files about load, network activity, storage
- **Social Media** — Identify topics of interest online, such as *top-k* hash tags on Twitter
- ...

Requirements

- Real-time response
- Continuous processing and analysis
- High-volume data, potentially infinite
- High-velocity data (many changes)
Infinite streams of data, but...

- **Limited** main memory and
- Only **sequential access**

**Solutions**

- Reduction of data amount (e.g., sampling) or
- Buffering (**windowing**)
Data Stream Processing
Processing Model and Windowing

- **Windows**
  - Count-Based
  - More common for real applications
  - Variable number of events per window
  - Problematic due to limited GPU memory
  - Time-Based

**stream of events**

**finite**

**stream of windows**

**finite**

**finite**
Data Stream Processing
Bottleneck — Example Join Algorithm

- Number of join candidates depends on number of events inside window
Data Stream Processing

Bottleneck — Example Join Algorithm

- Number of join candidates depends on number of events inside window
- Many events in the same instant for time-based windows
  - Decrease of throughput
Data Stream Processing

Bottleneck — Back Pressure

Data flow systems (e.g., stream processing) suffer of back pressure

**Back pressure**

- Upwards-propagated decrease of throughput
- To the level of the slowest component

*Results is need for load shedding.*
Data Stream Processing

Bottleneck

throughput

slowest component
Data Stream Processing

Bottleneck

throughput

slowest component
Data Stream Processing

Bottleneck
Data Stream Processing

Bottleneck

throughput

slowest component
Data Stream Processing

Bottleneck

the slowest component throughputs
Data Stream Processing

Bottleneck

... … \( \sigma \) …

throughput

... … \( \sigma \) …

slowest component
Data Stream Processing
Bottleneck — Solutions

- Parallelization of operators

- Distributed computation

Site 1

Site 2

more computation resources
Toward GPU Accelerated Data Stream Processing

In DBMS?
GPUs in DMBS

- ... Efficient co-processor
- ... Might outperform CPUs for certain operations
- ... Computations are highly parallel (SIMD)
- ... Huge corpus on research results

Some conclusions

- Data transfer costs to and from graphic card are critical
- Operation should match GPU architecture (e.g., branch free)
- Operation must be expensive enough to amortize transfer costs
- Column-oriented architectures save transfer costs
GPU Acceleration for Data Stream Processing

Challenges

- Limited memory on graphic cards
- (time-based) windows can be huge
- event representation (tuple) does not match the GPU architecture
GPU-ready Stream Processing

Our 1\textsuperscript{st} contribution: Handle graphic card memory limitation for very large windows via \textit{bucketing}
We suggest

Portioning streams of variable-length window of tuples

into a stream of “Buckets”

**Bucket**: fixed-size window portions with column-oriented event representation
GPU-ready Stream Processing

Bucketing (2)

Bucket-at-a-Time
Let’s say bucket size 3

Let’s say bucket size 5
Bucket-at-a-Time
GPU-ready Stream Processing
Bucketing (2)

Bucket-at-a-Time
Let’s say bucket size 3

Let’s say bucket size 5
Bucket-at-a-Time
GPU-ready Stream Processing
Bucketing (2)

3 events, column-oriented

Bucket-at-a-Time
Let’s say bucket size 3

Let’s say bucket size 5
Bucket-at-a-Time
GPU-ready Stream Processing
Bucketing (2)

Let's say bucket size 3
Let's say bucket size 5

5 events, column-oriented
GPU-ready Stream Processing

Bucketing (2)

Let's say bucket size 3

Let's say bucket size 5

Bucket-at-a-Time

Bucket-at-a-Time
GPU-ready Stream Processing
Bucketing (2)

Let's say bucket size 3

Let's say bucket size 5
GPU-ready Stream Processing
Bucketing (2)

Let’s say bucket size 3

Bucket-at-a-Time

Let’s say bucket size 5

Bucket-at-a-Time
GPU-ready Stream Processing
Bucketing (2)

Bucket-at-a-Time
Let’s say bucket size 3

Bucket-at-a-Time
Let’s say bucket size 5
GPU-ready Stream Processing

Benefits through Bucketing

- **Each operator** requests its own bucket size $k$

- The bucket size is **independent of** the actual **window length**
  - Memory allocation on graphic card has an upper bound for input

- Bucketing **flips event representation**
  - Processing entire columns

- **Window length > bucket size**, the **window is split** into portions

- **Single bucketing-operator** can be **subscribed by many** operators
# Toward GPU Accelerated Data Stream Processing

## Buckets versus Windows

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Windowing</th>
<th>Bucketing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bounding infinite stream</strong></td>
<td>[Bouncing infinite stream]</td>
<td>[Portioning windows]</td>
</tr>
<tr>
<td><strong>Consumes</strong></td>
<td><strong>Stream of events</strong></td>
<td><strong>Stream of windows</strong></td>
</tr>
<tr>
<td><strong>Produce</strong></td>
<td><strong>Stream of windows</strong></td>
<td><strong>Stream of buckets</strong></td>
</tr>
<tr>
<td><strong>#Events</strong></td>
<td><strong>Might be huge</strong></td>
<td><strong>Has upper bound</strong></td>
</tr>
<tr>
<td><strong>Events Representation</strong></td>
<td><strong>Tuples</strong></td>
<td><strong>Column-wise</strong></td>
</tr>
</tbody>
</table>
GPU-ready Stream Processing

Achieve bucketing

Actual View

Ring Buffer 1

Ring Buffer 2

... 

Ring Buffer n

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3

$n$ Stream Schema Length
GPU-ready Stream Processing
Achieve bucketing

(a₁, b₁, c₁)

Actual View

Ring Buffer 1

Ring Buffer 2

... 

Ring Buffer n

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3

n Stream Schema Length
GPU-ready Stream Processing

Achieve bucketing

Actual View

Ring Buffer 1

Ring Buffer 2

Ring Buffer $n$

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3
$n$ Stream Schema Length
GPU-ready Stream Processing
Achieve bucketing

(a_2 b_2 c_2)

Actual View
Ring Buffer 1
Ring Buffer 2
... 
Ring Buffer n

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3
n Stream Schema Length
GPU-ready Stream Processing
Achieve bucketing
GPU-ready Stream Processing
Achieve bucketing

Actual View
Ring Buffer 1
Ring Buffer 2
Ring Buffer n

(a_3 \ b_3 \ c_3)
GPU-ready Stream Processing
Achieve bucketing

Actual View
Ring Buffer 1
Ring Buffer 2
Ring Buffer n

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3

n Stream Schema Length

a_1, a_2, a_3
b_1, b_2, b_3
c_1, c_2, c_3
GPU-ready Stream Processing

Achieve bucketing

(a_4 \ b_4 \ c_4)

Actual View

Ring Buffer 1

Ring Buffer 2

Ring Buffer n

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3

n Stream Schema Length
GPU-ready Stream Processing
Achieve bucketing

Actual View

Ring Buffer 1

Ring Buffer 2

Ring Buffer n

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3

Stream Schema Length

n

\[
\begin{array}{cccc}
 a_1 & a_2 & a_3 & a_4 \\
 b_1 & b_2 & b_3 & b_4 \\
 \ldots & & & \\
 C_1 & C_2 & C_3 & C_4 \\
\end{array}
\]
GPU-ready Stream Processing
Achieve bucketing

Actual View

Ring Buffer 1

Ring Buffer 2

Ring Buffer n

(a_5 \ b_5 \ c_5)
GPU-ready Stream Processing
Achieve bucketing

Actual View

Ring Buffer 1

Ring Buffer 2

Ring Buffer n

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3

Stream Schema Length

n

a_1 a_2 a_3 a_4 a_5

b_1 b_2 b_3 b_4 b_5

C_1 C_2 C_3 C_4 C_5
GPU-ready Stream Processing
Achieve bucketing

(a_6 \ b_6 \ c_6)

Actual View

Ring Buffer 1

Ring Buffer 2

Ring Buffer n

Ring Buffer

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3
Stream Schema Length n
GPU-ready Stream Processing
Achieve bucketing

Actual View

Ring Buffer 1

Ring Buffer 2

Ring Buffer n

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3
n Stream Schema Length
GPU-ready Stream Processing
Achieve bucketing

(a_7 \ b_7 \ c_7)

Actual View
Ring Buffer 1
Ring Buffer 2
Ring Buffer n

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3
n Stream Schema Length

Achieve bucketing
Achieve bucketing

Actual View

Ring Buffer 1

Ring Buffer 2

Ring Buffer $n$

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3
$n$ Stream Schema Length
GPU-ready Stream Processing
Achieve bucketing

(a_8 b_8 c_8)

Ring Buffer 1

Ring Buffer 2

Ring Buffer n

Actual View

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3

n Stream Schema Length
GPU-ready Stream Processing
Achieve bucketing

Actual View
Ring Buffer 1
Ring Buffer 2
Ring Buffer n

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3

Stream Schema Length

n

Toward GPU Accelerated Data Stream Processing
GPU-ready Stream Processing
Achieve bucketing

(a₉ b₉ c₉)

Actual View
Ring Buffer 1
Ring Buffer 2
Ring Buffer n

Slice subscriber 1
Slice subscriber 2
Slice subscriber 3
n Stream Schema Length

Toward GPU Accelerated Data Stream Processing
Open Research Challenges

Our 2nd contribution: Identification of research challenges related to co-processing for Data Stream Processing
Open Research Challenges
Modern hardware and scheduling in Stream Processing

- **Other** specialized **co-processors** might be possible
  - Intel Xeon Phi or FPGAs for instance

- Optimized **algorithm** and executions **models for** the certain **co-processor**

- More than CPU-only Data Stream Processing:
  - Large physical **query execution plan space**

- **Find** best performance for a
  - **Logic plan** and
  - **Load sharing** between devices

*Further research should be investigated to find limitations and benefits for applying modern hardware here.*
Conclusion
Conclusion

Bucketing windows enables GPU-ready Data Stream Processing for very large windows

- Memory allocation has upper bound for input (fixed-size)
- Reduces transfer costs (column-selection)

We present an approach to achieve bucketing

- Separate operator, independent of SPS’s tuple-at-a-time or batch-at-a-time support
- Ring buffer per attribute plus per-subscriber slice
- Enables processing of large-scale windows on limited graphic card memory
  - No fallback to CPU required

We identify research challenges for further co-processing in this context.

- Other co-processors with specialized algorithm — limitations and benefits
- Large search space for query plans (logical operator — devices — concrete algorithm) — optimizer