Accelerated Event Processing with GPUs

Prabodha Srimal, University of Moratuwa, Sri Lanka
Dilum Bandara, University of Moratuwa, Sri Lanka
Srinath Perera, WSO2 Inc., Sri Lanka

HiPC 2015
Outline

• Introduction and Motivation
• Proposed GPU based CEP Framework
• Evaluation and Results
• Conclusion and future work
Complex Event Processing (CEP)

- Processing multiple event streams to identify meaningful patterns, using complex conditions & temporal windows

![Diagram of CEP process]

- Users submit queries, which are compiled into a query compiler.
- The CEP engine processes the queries against incoming events.
- Output is generated based on the processed events and the query conditions.
- Output adaptors manage the generated output, while input adaptors handle incoming events.
CEP Performance?

• What is CEP performance?
  o Event handling Rate
  o Latency (Processing/Information)
  o Number of standing queries
  o Complexity of queries and events
  o Resource Consumption

• Why CEP performance matters?
  o Data avalanche
Growth of Digital Universe

10x
Data growth from 2013 to 2020 from 4.4 trillion GB to 44 trillion GB

Data that is Useful if Tagged & Analyzed
22% 37%

Source: EMC Digital Universe with Research & Analysis by IDC. 2014

Actually analyzed in 2013 less than 5%

Data from Internet of Things 2% to 10% from 2013 to 2020
Where CEP is Used?

- Monitoring Applications
- Manufacturing Applications
- Data Warehousing Application
- Operational Analytics Applications
- Web Analytics Applications
- Financial Trading Applications
- Relational Database Applications

Latency:
- Months
- Days
- hours
- Minutes
- Seconds
- 100 ms
- < 1ms

Aggregate Data Rate (Events/seconds)

Source – WSO2 CEP 3.1.0 product release webinar
CEP Performance

• How to improve?
  o Distributed event processing

• But....
  o Some event operators cannot distribute
  o Shared state – Event Window, etc.
  o Network communication overhead
  o Some queries are very complex

• Can GPUs help?
Related Work

• Few research on GPU for Event Processing
• Published work for GPU on CEP
• Commercial CEP vendors advertise use of GPUs
  o No publicly available implementation details
  o Only for some specific use cases
• Related research is not enough...
Objectives

Investigate how and when GPUs can be used to improve the query processing performance of Complex Event Processing engines

Siddhi CEP engine used as CEP implementation
Outline

• Motivation
  o CEP Performance
  o Growth of Digital Universe
  o Improving CEP performance

• **Proposed Solution**
  o GPU Event Processing Library
  o Siddhi CEP Changes

• Evaluation and Results

• Conclusions and Future work
Proposed Solution

- **GPU Event Processing Library**
  - Event Processors implemented in CUDA
  - Low-level GPU device handling

- **Integration with Siddhi CEP**
  - Extended QueryRuntime called “GpuQueryRuntime”
GPU Event Processing Library

• A general purpose event processing library
  o Decouple low-level GPU device handling with CEP implementation
  o Test and debug device specific implementation separately
  o Better control over GPU devices
  o Direct access to CUDA Runtime API and Driver API
GPU Event Processors

• Implementation of event processing operators using CUDA Kernel(s)
• Each operator consist of one or more CUDA kernel(s)

• Currently implemented
  o Filter event processor
  o Window event processor
  o Stream Join event processor
@plan:name('QueryPlan1') @plan:parallel
define stream StockStream (  
symbol string,  
price float,  
volume int);

@info(name='query1')
from StockStream [price > 100 AND  
symbol == "GOOG"]#window.time(5 min)
select price, volume, count(price) stockCount
insert into TopStockStream;
Event Stream Join

```java
define stream BidStream (symbol string, bidPrice float, bidQty int);

define stream AskStream (symbol string, askPrice float, askQty int);

from BidStream#window(1 sec) as B
join AskStream#window(1 sec) as A
on (B.symbol == A.symbol)
select B.symbol, B.bidPrice, A.askPrice
insert into StockStream;
```
```sql
from BidStream#window.length(1000) as B
join AskStream#window.length(2000) as A
on (B.symbol == A.symbol)
select B.symbol, B.bidPrice, A.askPrice
insert into StockStream;
```

- Sequential processing does not scale with event window size
- Parallel processing can improve performance
- But needs synchronized access to event windows
Outline

• **Motivation**
  - CEP Performance
  - Growth of Digital Universe
  - Improving CEP performance

• **Proposed Solution**
  - GPU Event Processing Library
  - *Siddhi CEP Changes*

• **Evaluation and Results**

• **Conclusions and Future work**
Siddhi Query Model

- **Execution Plan** – Use case
  - Single threaded and multi-threaded
- **Query Runtimes** – Each Queries
  - Event processor pipeline
- **Input event queue and Stream Receivers**
• Integrate GPU processing through query annotations
  
  o \texttt{@gpu} – annotation to GPU parameters

\begin{verbatim}
@plan:name('QueryPlan1') @plan:parallel
define stream cseEventStream (symbol string, price float, 
volume int, change float, pctchange float);

@info(name='query1')
@gpu(cuda.device='0', batch.size='2048', block.size='128')
from cseEventStream[pctchange > 0.1 
  and change < 2.5 
  and volume > 100 
  and price < 70]
select symbol, price, volume, change, pctchange
insert into outputStream;
\end{verbatim}
Outline

• Motivation
  o CEP Performance
  o Growth of Digital Universe
  o Improving CEP Performance

• Proposed Solution
  o GPU Event Processing Library
  o Siddhi Changes

• Evaluation and Results

• Conclusions and Future work
Performance Evaluation

• Used a publically available real-world workload
  o ACM DEBS 2013 – Grand challenge dataset
  o High velocity sensor data of a soccer game

• Measured;
  o Input event consume rate
  o Input event queue publish latency
  o Individual event processing throughput of QueryRuntimes
  o Average event serialization / de-serialization times
  o Average GPU event processing time (memCopy+processing)

• CUDA kernel profiling
  o Individual time measurements for each functions
  o Memory copy times

• Environment
  o GPU devices - Nvidia GeForce GTX 480
  o Host - 64bit Intel Core i7 950 CPU (8 core)
Filter Query Performance

Event consume speedup against query count

- Filter query event consume rate speedup against concurrent query count (event batch size 2048 events).
Filter Query Performance

- GPU processing time breakdown (event batch size 2048 events).

<table>
<thead>
<tr>
<th>GPU Kernel</th>
<th>Time(%)</th>
<th>Average Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>[CUDA memcpy HtoD]</td>
<td>66.73</td>
<td>72.938us</td>
</tr>
<tr>
<td>ProcessEventsFilterKernel(KernelParameters*, int)</td>
<td>29.75</td>
<td>32.542us</td>
</tr>
<tr>
<td>[CUDA memcpyDtoH]</td>
<td>3.52</td>
<td>3.849us</td>
</tr>
</tbody>
</table>
Join Query Performance

- More than 2 times increased throughput in GPU processing than single thread processing
- Less event queue build-up for GPU processing
Join Query Performance

- Processing time breakdown
- Event batch size 2048 events.

<table>
<thead>
<tr>
<th>GPU Kernel</th>
<th>Time(%)</th>
<th>Average Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>[CUDA memcpy DtoH]</td>
<td>84.87</td>
<td>8.5689ms</td>
</tr>
<tr>
<td>ProcessEventsJoinRightTriggerCurrentOn</td>
<td>6.97</td>
<td>2.8136ms</td>
</tr>
<tr>
<td>ProcessEventsJoinLeftTriggerCurrentOn</td>
<td>5.89</td>
<td>2.3815ms</td>
</tr>
<tr>
<td>[CUDA memcpy HtoD]</td>
<td>0.70</td>
<td>139.26us</td>
</tr>
<tr>
<td>JoinSetWindowState</td>
<td>0.67</td>
<td>135.86us</td>
</tr>
<tr>
<td>FilterKernel</td>
<td>0.89</td>
<td>179.513us</td>
</tr>
<tr>
<td>[CUDA memcpy DtoD]</td>
<td>0.01</td>
<td>2.2150us</td>
</tr>
</tbody>
</table>
Query Mix Performance

- Three event queries per use case
- One query on GPU and others on CPU
- More than 10 times increased event processing throughput
- Input event queue build-up is significantly less
Conclusions

• GPUs can improve query processing throughput
  o About 10x for real-world use cases
  o Significantly less event queue build-up
  o Main CPU free for other tasks

• Not all queries preform well with GPUs
  o All queries should not run on GPUs. Run most complex on GPUs
  o A pre-run with GPUs require to identify if there is a performance gain
Future Work

• Implementing GPU Algorithms for Other CEP Operators
• Evaluate other GPU memory types
  o Texture memory for store input events
• Automatic Query Configure to Run on GPUs
• Runtime GPU Kernel Generation
  o Optimized GPU kernels for specific case
  o Improved serialization/de-serialization
Thank You!