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

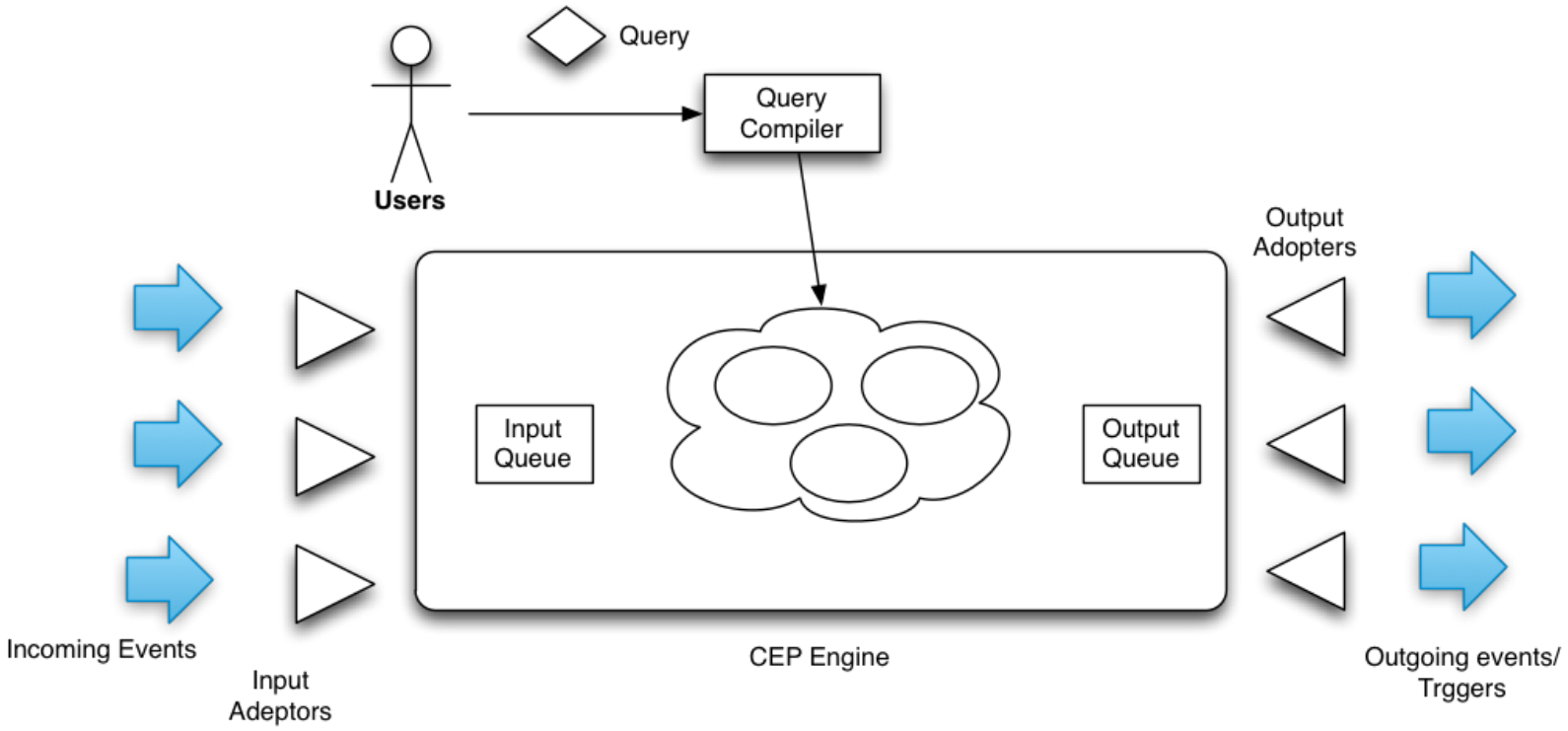
HiPC 2015

Outline

- Introduction and Motivation
- Proposed GPU based CEP Framework
- Evaluation and Results
- Conclusion and future work

Complex Event Processing (CEP)

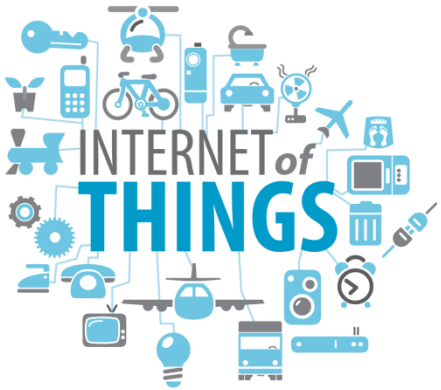
- Processing multiple event streams to identify meaningful patterns, using complex conditions & temporal windows



CEP Performance?

- What is CEP performance?
  - Event handling Rate
  - Latency (Processing/Information)
  - Number of standing queries
  - Complexity of queries and events
  - Resource Consumption
- Why CEP performance matters?
  - Data avalanche

Growth of Digital Universe



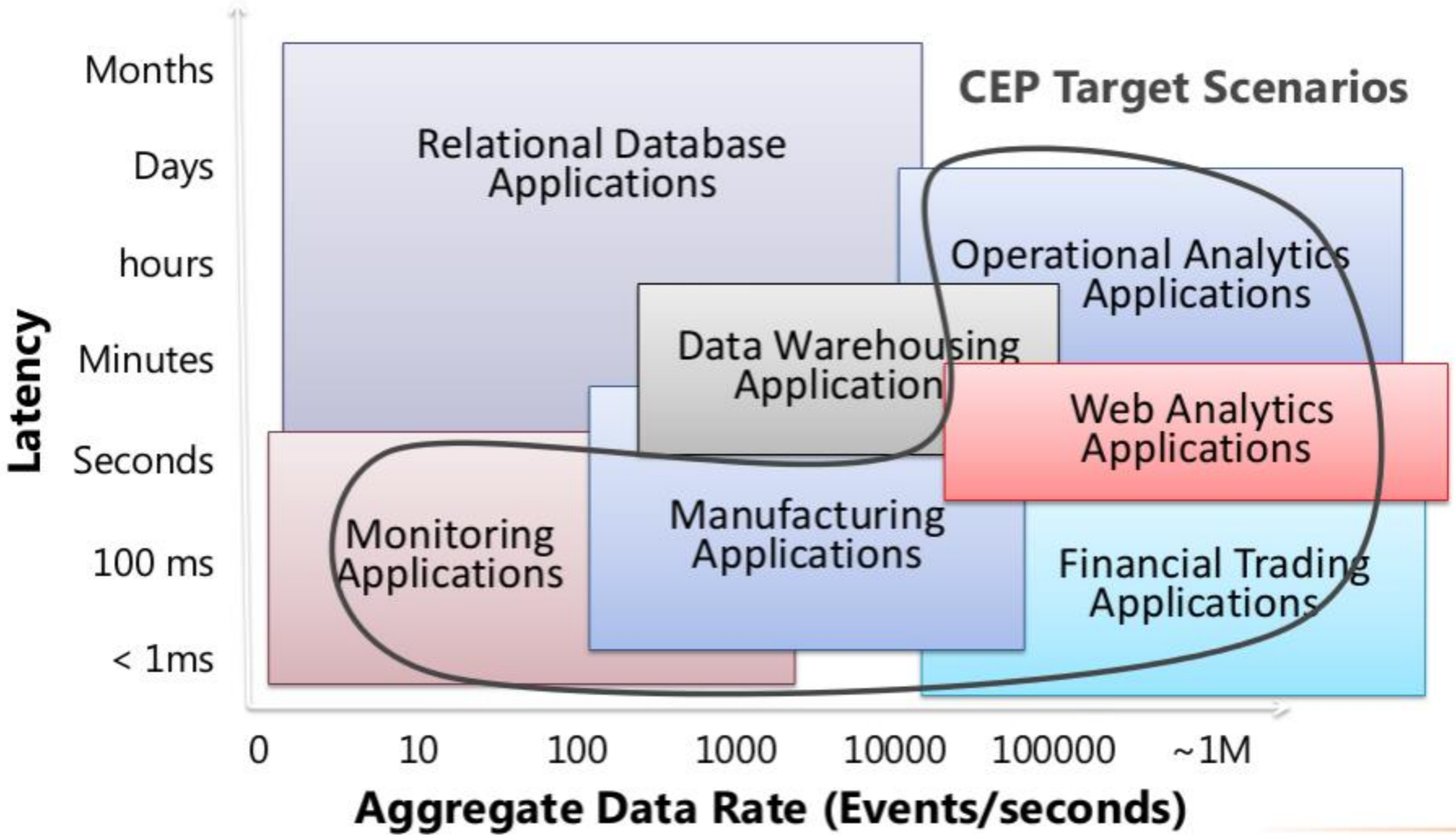
10x
Data growth from 2013 to 2020
from 4.4 trillion GB to 44 trillion GB

Actually analyzed in 2013
less than 5%

Data from Internet of Things
2% to 10%
from 2013 to 2020

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

Where CEP is Used?



- Source – WSO2 CEP 3.1.0 product release webinar

CEP Performance

- How to improve?
  - Distributed event processing

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

- Can GPUs help?

Related Work

- Few research on GPU for Event Processing
- Published work for GPU on CEP
  - G. Cugola and A. Margara, “Low latency complex event processing on parallel hard- ware,” *Journal of Parallel and Distributed Computing, 2012*
  - S. Mallick and M. Emani, “Method and system for performing event-matching with a graphical processing unit,” 2012, US Patent
- Commercial CEP vendors advertise use of GPUs
  - No publically available implementation details
  - Only for some specific use cases
- Related research is not enough…

Objectives

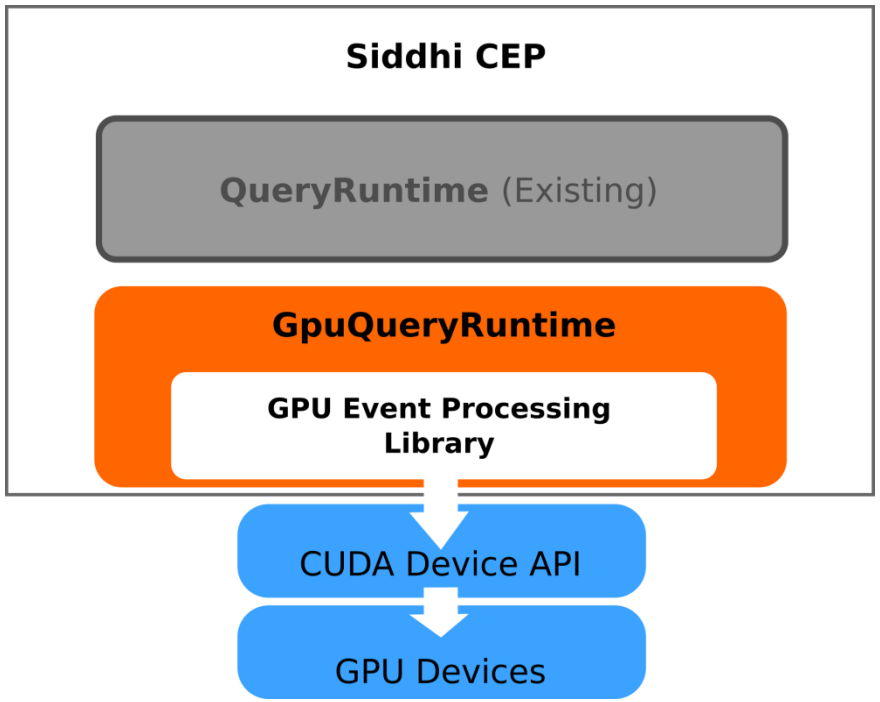
Investigate how and when GPUs can be used to improve the query processing performance of Complex Event Processing engines

Siddhi CEP engine used as CEP implementation

Outline

- Motivation
  - 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

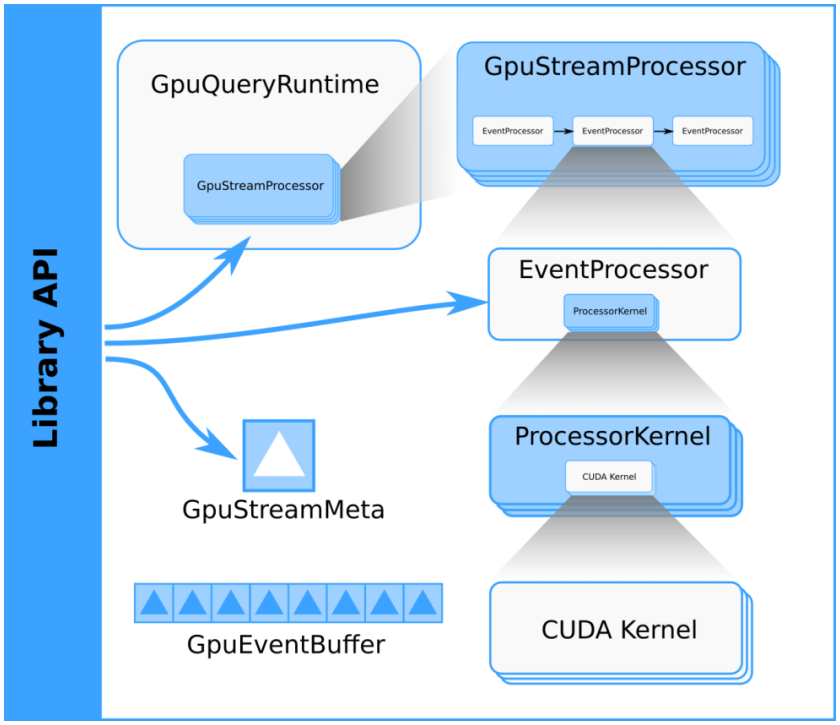
Proposed Solution



- 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
  - Decouple low-level GPU device handling with CEP implementation
  - Test and debug device specific implementation separately
  - Better control over GPU devices
  - Direct access to CUDA Runtime API and Driver API



GPU Event Processors

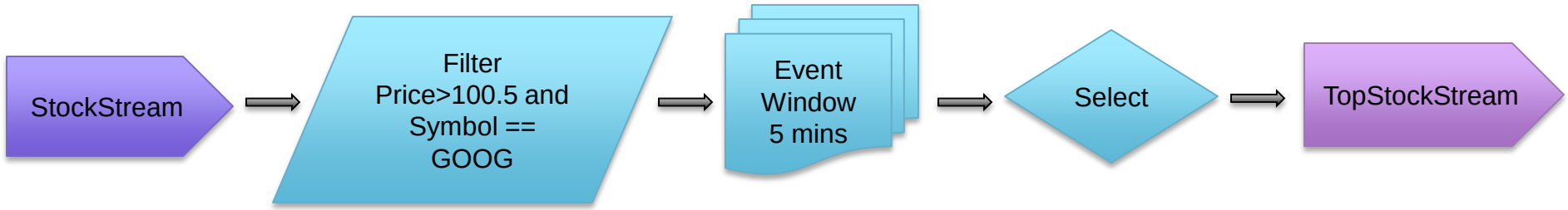
- Implementation of event processing operators using CUDA Kernel(s)
- Each operator consist of one or more CUDA kernel(s)

- Currently implemented
  - Filter event processor
  - Window event processor
  - Stream Join event processor

CEP Query Language – SiddhiQL

```
@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

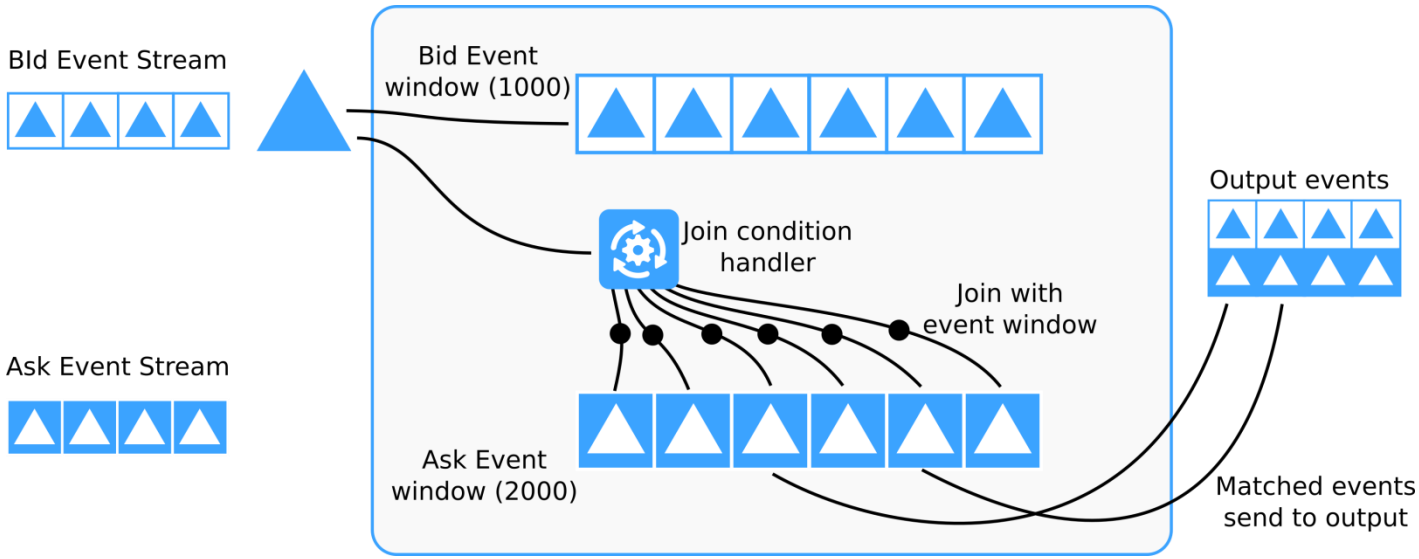
```
define stream BidStream (symbol string, bidPrice float,
                         bidQty int);

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;
```

Event Stream Join Processor

```
        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;
```

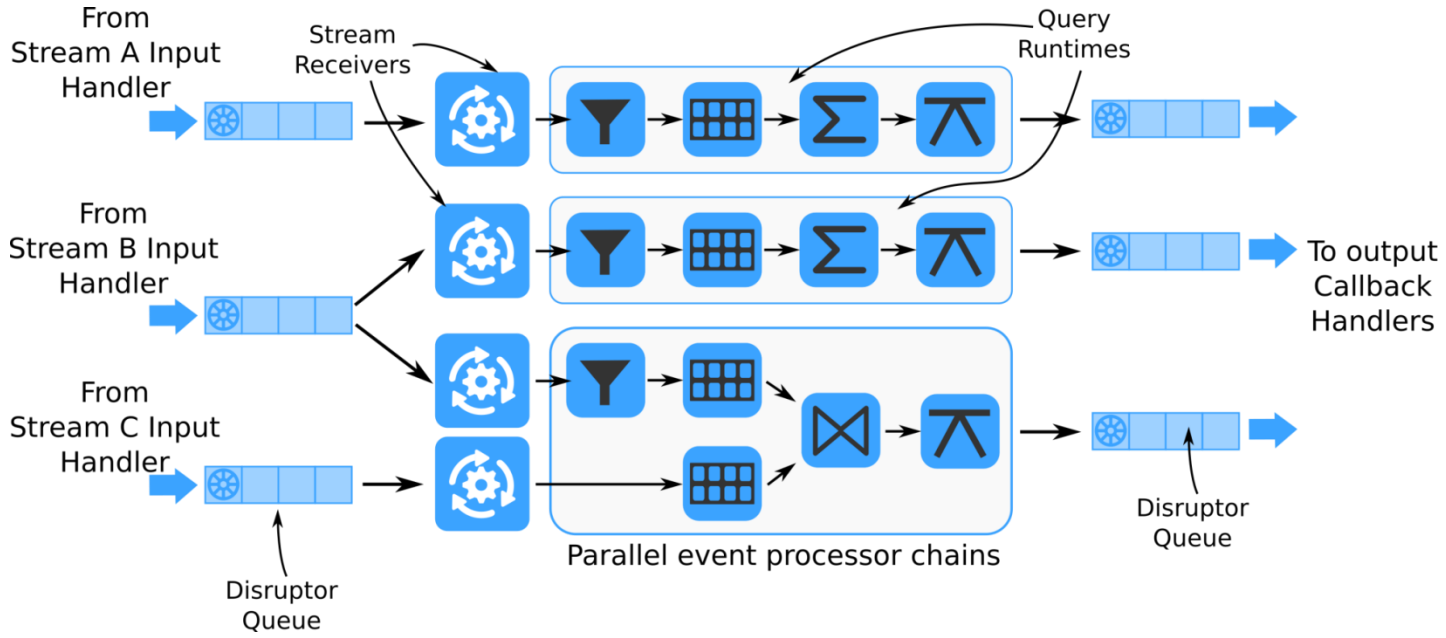


- Sequential processing does not scale with event window size
- Parallel processing can improve performance
- But needs synchronized access to event windows

Outline

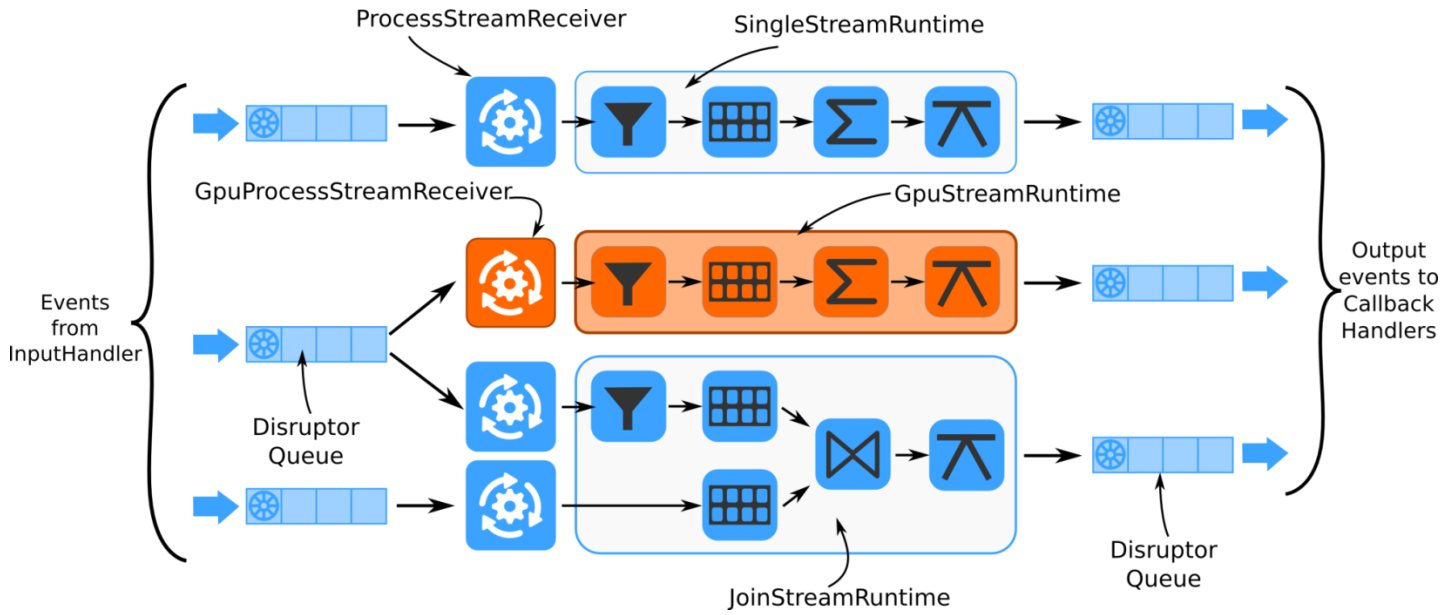
- Motivation
  - CEP Performance
  - Growth of Digital Universe
  - Improving CEP performance
- Proposed Solution
  - GPU Event Processing Library
  - Siddhi CEP Changes
- Evaluation and Results
- Conclusions and Future work

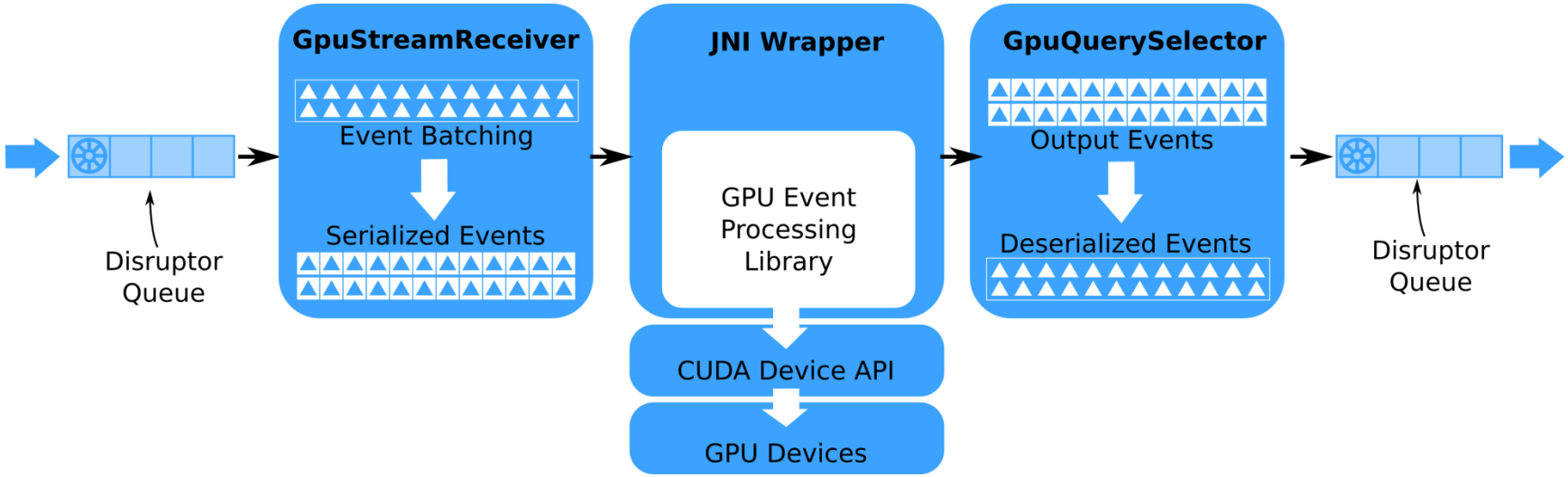
Siddhi Query Model



- ExecutionPlan – Use case
  - Single threaded and multi-threaded
- Query Runtimes – Each Queries
  - Event processor pipeline
- Input event queue and Stream Receivers

GpuQueryRuntime





GpuQueryRuntime - Configurations

- Integrate GPU processing through query annotations
  - **@gpu** – annotation to GPU parameters

```
@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;
```

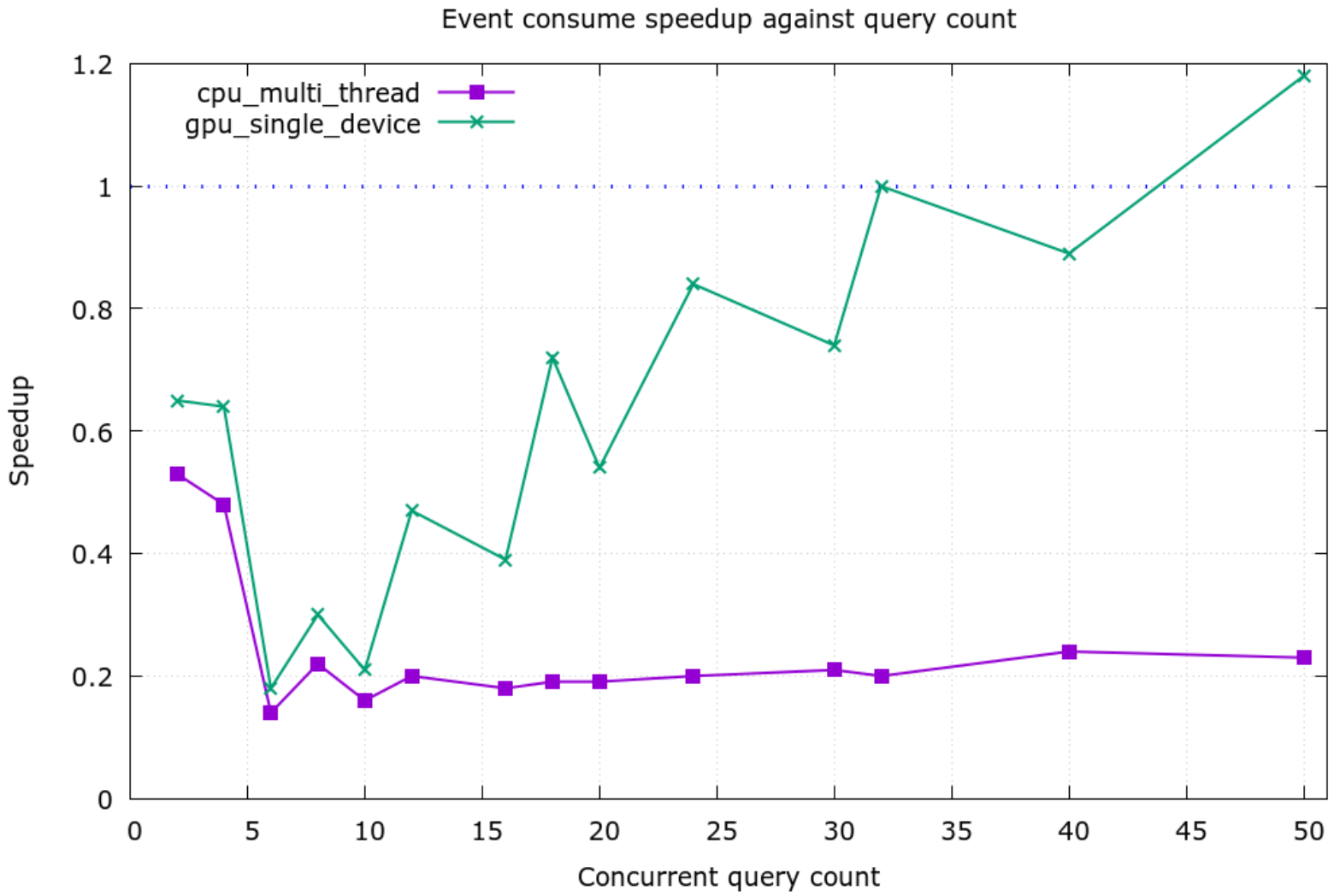
Outline

- Motivation
  - CEP Performance
  - Growth of Digital Universe
  - Improving CEP Performance
- Proposed Solution
  - GPU Event Processing Library
  - Siddhi Changes
- Evaluation and Results
- Conclusions and Future work

Performance Evaluation

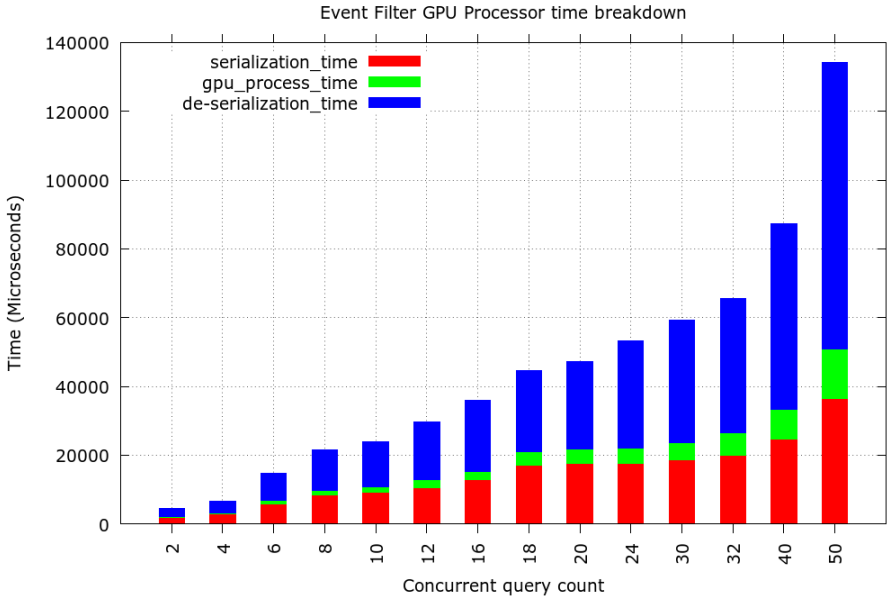
- Used a publically available real-world workload
  - ACM DEBS 2013 – Grand challenge dataset
  - High velocity sensor data of a soccer game
- Measured;
  - Input event consume rate
  - Input event queue publish latency
  - Individual event processing throughput of QueryRuntimes
  - Average event serialization / de-serialization times
  - Average GPU event processing time (memCopy+processing)
- CUDA kernel profiling
  - Individual time measurements for each functions
  - Memory copy times
- Environment
  - GPU devices - Nvidia GeForce GTX 480
  - Host - 64bit Intel Core i7 950 CPU (8 core)

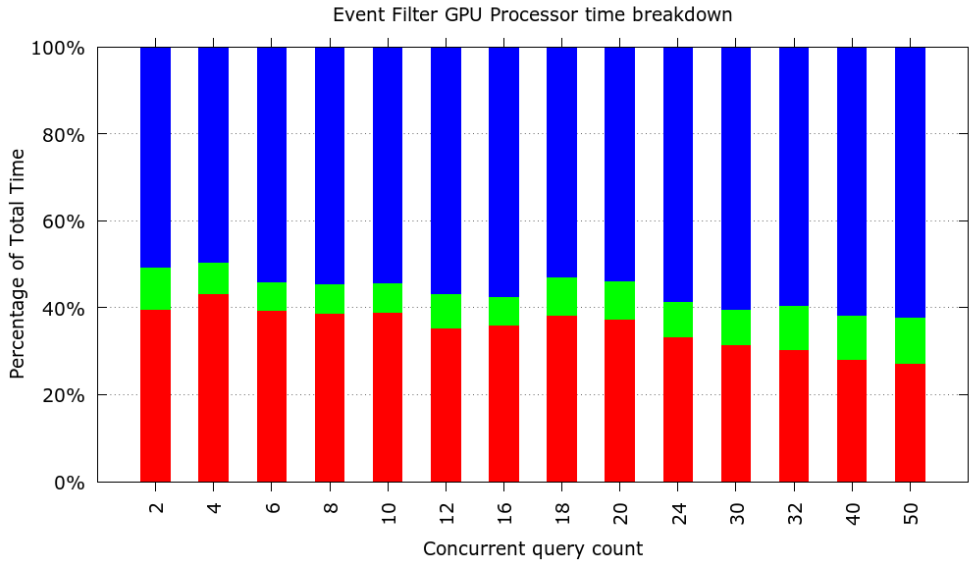
Filter Query Performance



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

Filter Query Performance

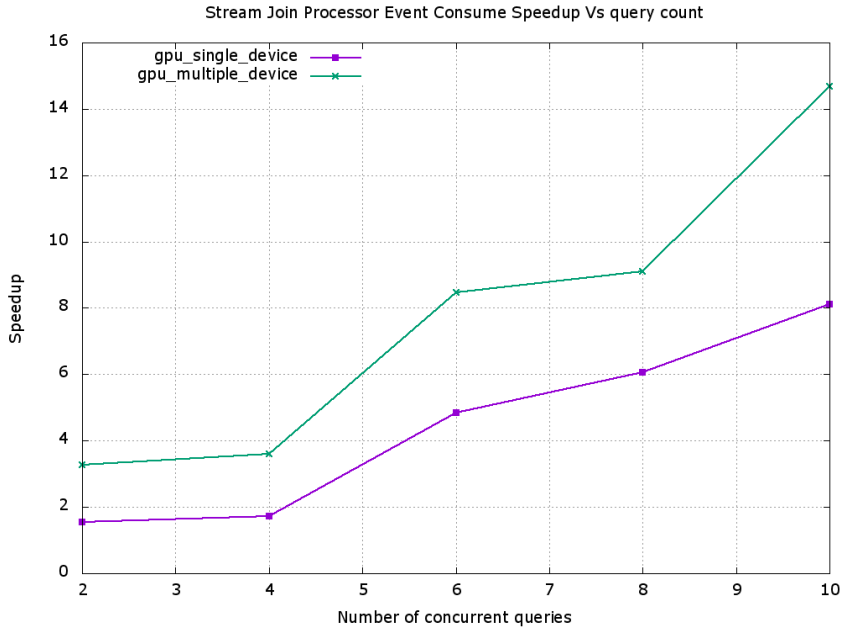


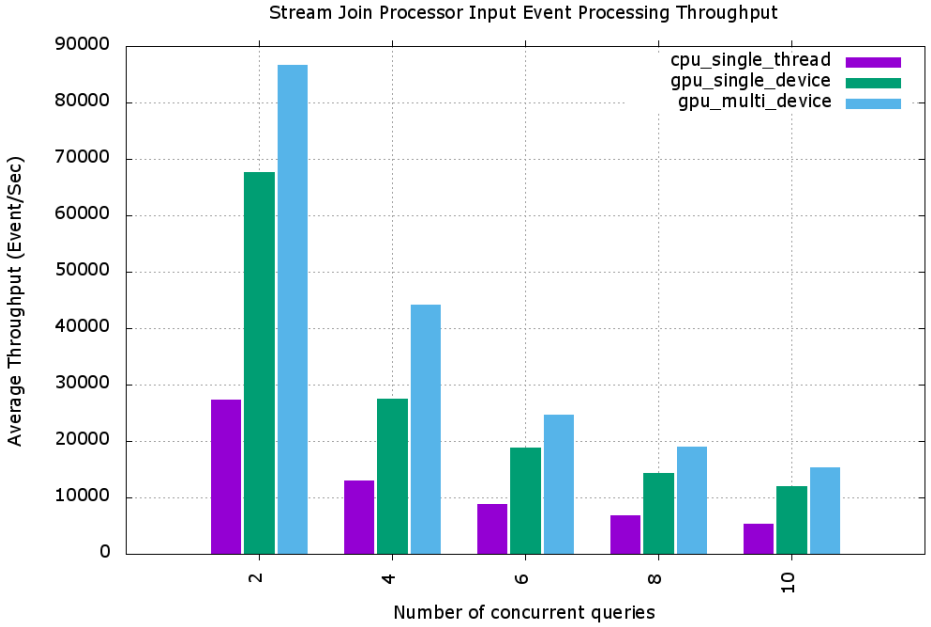


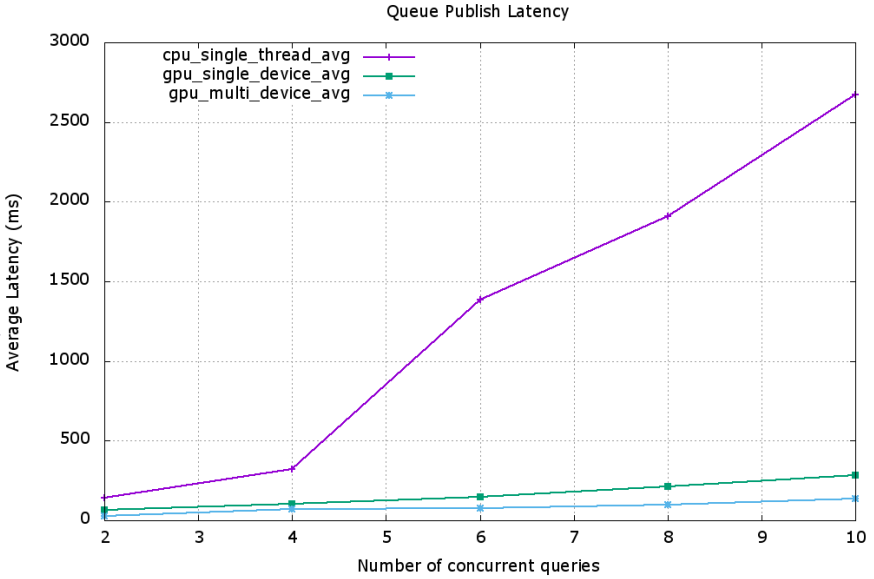
GPU Kernel	Time(%)	Average Time
`[CUDA memcpy HtoD]`	66.73	72.938us
`ProcessEventsFilterKernel(KernelParameters*, int)`	29.75	32.542us
`[CUDA memcpy DtoH]`	3.52	3.849us

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

Join Query Performance

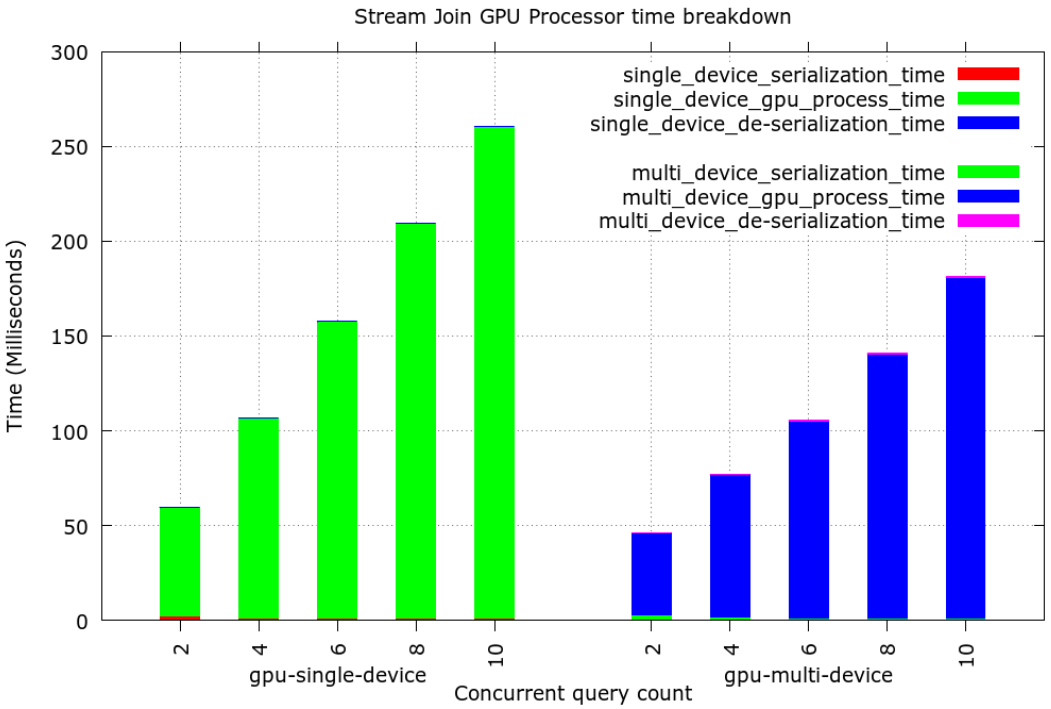






- More than 2 times increased throughput in GPU processing than single thread processing
- Less event queue build-up for GPU processing

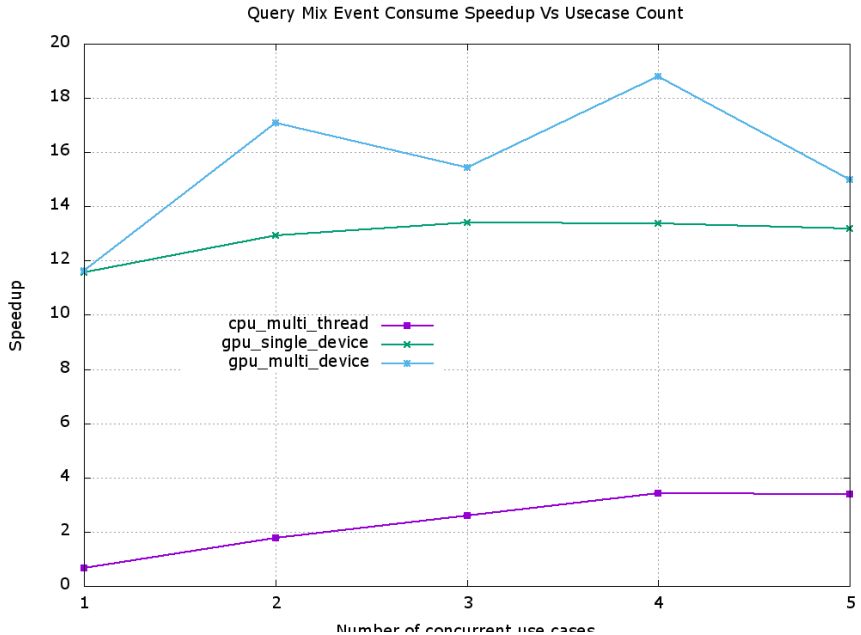
Join Query Performance

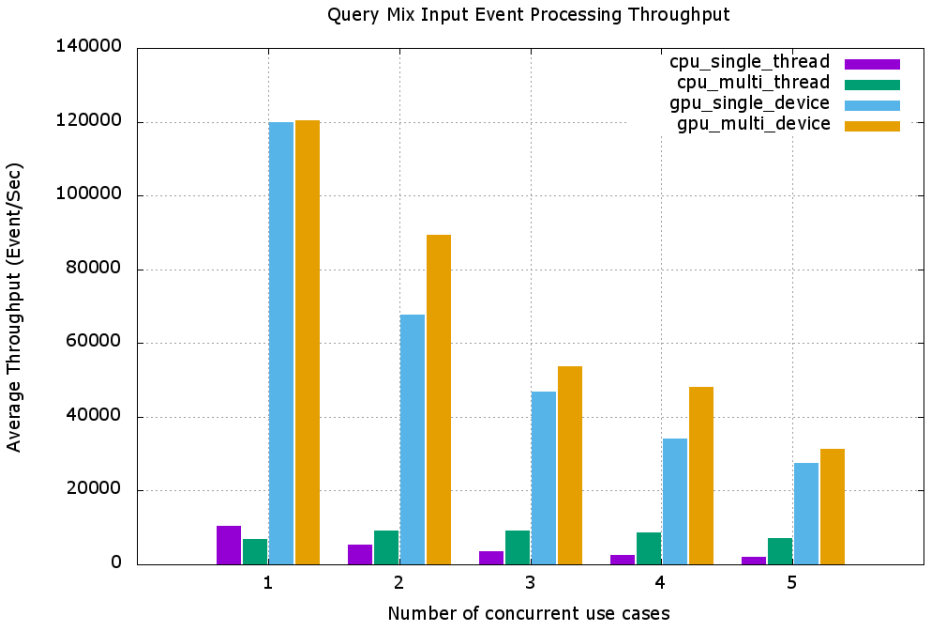


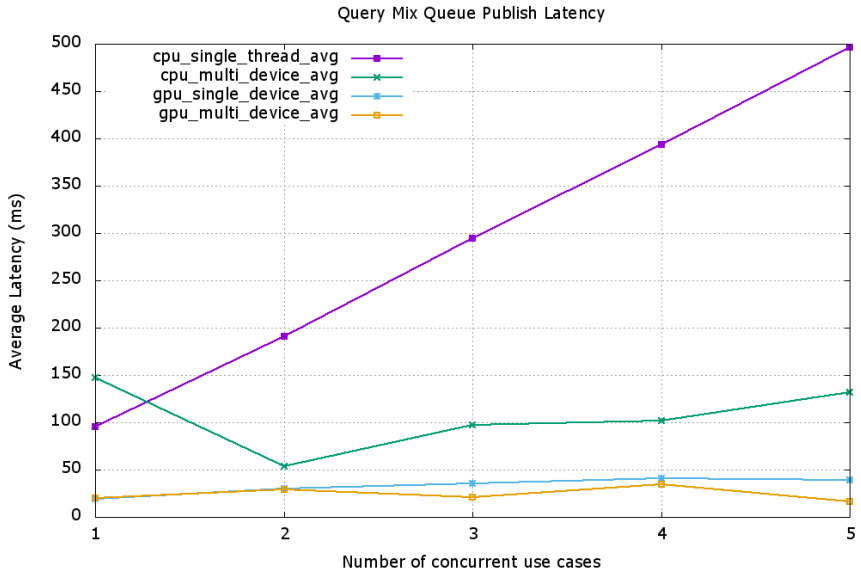
- Processing time breakdown
- event batch size 2048 events.

GPU Kernel	Time(%)	Average Time
[CUDA memcpy DtoH]	84.87	8.5689ms
ProcessEventsJoinRightTriggerCurrentOn	6.97	2.8136ms
ProcessEventsJoinLeftTriggerCurrentOn	5.89	2.3815ms
[CUDA memcpy HtoD]	0.70	139.26us
JoinSetWindowState	0.67	135.86us
FilterKernel	0.89	179.513us
[CUDA memcpy DtoD]	0.01	2.2150us

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
  - About 10x for real-world use cases
  - Significantly less event queue build-up
  - Main CPU free for other tasks
- Not all queries preform well with GPUs
  - All queries should not run on GPUs. Run most complex on GPUs
  - 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
  - Texture memory for store input events
- Automatic Query Configure to Run on GPUs
- Runtime GPU Kernel Generation
  - Optimized GPU kernels for specific case
  - Improved serialization/de-serialization

Thank You!