High-Throughput Low-Latency Analytics

Facebook Insights
9GB of page metrics/s
In less than 10 s

Google Zeitgeist
40K user queries/s
Within ms

Feedzai
40K card trans/s
In 25 ms

NovaSparks
150M stock options/s
In less than 1 ms
Exploit Single-Node Heterogeneous Hardware

Servers with CPUs and GPUs now common
- 10x higher linear memory access throughput
- Limited data transfer throughput

Use both CPU & GPU resources for stream processing
CQL: **SQL-based declarative language** for continuous queries [Arasu et al., VLDBJ’06]

Credit card fraud detection example:

- Find attempts to use same card in different regions within 5-min window

```sql
select distinct W.cid
from Payments [range 300 seconds] as W,
Payments [partition-by 1 row] as L
where W.cid = L.cid and W.region != L.region
```

CQL offers correct window semantics
Challenges & Contributions

1. How to parallelise sliding-window queries across CPU and GPU?
   Decouple query semantics from system parameters

2. When to use CPU or GPU for a CQL operator?
   Hybrid processing: offload tasks to both CPU and GPU

3. How to reduce GPU data movement costs?
   Amortise data movement delays with deep pipelining
Problem: Window semantics affect system throughput and latency

– Pick task size based on window size?

Window-based parallelism results in redundant computation
How to Parallelise Window Computation?

Problem: Window semantics affect system throughput and latency

– Pick task size based on window size? On window slide?

Slide-based parallelism limits GPU parallelism
Idea: Decouple task size from window size/slide

- Pick based on underlying **hardware features**
  - e.g. PCIe throughput

- Task contains one or more **window fragments**
  - E.g. closing/pending/opening windows in \( T_2 \)
Merging Window Fragment Results

Idea: Decouple task size from window size/slide
- Assemble window fragment results
- Output them in correct order

Worker A stores $T_1$ results, merges window fragment results and forwards complete windows downstream
Challenges & Contributions

1. How to parallelise sliding-window queries across CPU and GPU?
   Decouple query semantics from system parameters

2. When to use CPU or GPU for a CQL operator?
   Hybrid processing: offload tasks to both CPU and GPU

3. How to reduce GPU data movement costs?
   Amortise data movement delays with deep pipelining
SABER’s Hybrid Stream Processing Model

Idea: Enable tasks to run on both processors
– Scheduler assigns tasks to idle processors

Past behavior:

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q_A</td>
<td>3 ms</td>
<td>2 ms</td>
</tr>
<tr>
<td>Q_B</td>
<td>3 ms</td>
<td>1 ms</td>
</tr>
</tbody>
</table>

Task Queue:

First-Come First-Served

FCFS ignores effectiveness of processor for given task
Heterogeneous Look-Ahead Scheduler (HLS)

Idea: Idle processor skips tasks that could be executed faster by another processor
– Decision based on observed query task throughput

Past behavior:

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>QA</td>
<td>3 ms</td>
<td>2 ms</td>
</tr>
<tr>
<td>QB</td>
<td>3 ms</td>
<td>1 ms</td>
</tr>
</tbody>
</table>

Task Queue:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>QA</td>
<td>QA</td>
<td>QB</td>
<td>QA</td>
<td>QB</td>
<td>QB</td>
</tr>
</tbody>
</table>

HLS

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>T3</td>
<td>T7</td>
<td>T10</td>
<td></td>
</tr>
<tr>
<td>GPU</td>
<td>T1</td>
<td>T2</td>
<td>T4</td>
<td>T5</td>
</tr>
</tbody>
</table>
The SABER Architecture

- **Dispatching stage**: Dispatch fixed-size tasks
- **Scheduling & execution stage**: Dequeue tasks based on HLS
- **Result stage**: Merge & forward partial window results

- **Java**: 15K LOC
- **C & OpenCL**: 4K LOC
Is Hybrid Stream Processing Effective?

Different queries result in different CPU:GPU processing split that is hard to predict offline.

- `select`
- `group-by_{avg}`
- `aggr_{avg}`
- `group-by_{avg}`
- `group-by_{cnt}`

Throughput (10^6 tuples/s)

- **Cluster Mgmt.**
- **Smart Grid**
- **LRB**

**SABER (CPU contrib.)**
- Intel Xeon 2.6 GHz
- 16 cores

**SABER (GPU contrib.)**
- NVIDIA Quadro K5200
- 2,304 cores
Is Hybrid Stream Processing Effective?

**Aggregate throughput of CPU and GPU always higher than its counterparts**

- **GPU is faster**
- **CPU is faster**
- **Not additive due to queue contention**

![Chart showing throughput comparison between CPU and GPU for Aggregation, Group-by, and θ-join operations.](chart.png)
Is Heterogeneous Look-Ahead Scheduling Effective?

$W_1$ benefits from static scheduling but HLS fully utilises GPU:
- GPU also runs $\sim 1\%$ of group-by tasks

$W_2$ benefits from FCFS but HLS better utilises GPU:
- HLS CPU:GPU split is 1:2.5 for project and 1:0.5 for aggr

<table>
<thead>
<tr>
<th>$\pi$</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>6x</td>
<td>5x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi$</td>
<td>1.5x</td>
<td>1.5x</td>
</tr>
</tbody>
</table>

Throughput (GB/s)

- FCFS
- Static
- HLS
Window processing model
Decouples query semantics from system parameters

Hybrid stream processing model
Can achieve aggregate throughput of heterogeneous processors

Hybrid Look-ahead Scheduling (HLS)
Allows use of both CPU and GPU opportunistically for arbitrary workloads

Thank you! Any Questions?
Alexandros Koliouisis
github.com/lsds/saber