A Scalable Complex Event Processing Framework for Combination of SQL-based Continuous Queries and C/C++ Functions

Takashi Takenaka, Masamichi Takagi
Hiroaki Inoue
NEC Corporation

August 30, 2012
Today’s Key Message

Synthesizing SQL with C to event processing on FPGAs
Outline

• Background

• Our work

• Evaluation

• Conclusion
Complex Event Processing (CEP)

- Real-world
- Real-Time Events

CEP

- Retrieve important information
- Patterns from Historical Data

- Decision Making
- Response
- Control

IDC: $10B (software) by 2014
CEP Applications

Financial Algorithmic Trading

Active diagnostics of facilities

Fraud detection: web commerce, credit card

Compliance reporting and monitoring

Track and Trace: Patients, packages

...
Capture a trend:
volume is starting high then it reaches 80%, then
Calculate "volume-weighted average of price (VWAP)" during the period.
Performance requirement

<table>
<thead>
<tr>
<th>Monitoring</th>
<th>Manufacturing</th>
<th>Financial</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web analysis</td>
<td>Data warehouse</td>
<td>Relational Database</td>
<td>Events per second</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 h</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Limit of SW solution</td>
</tr>
</tbody>
</table>

(Torsten Grabs, “Introduction to Microsoft SQL Server 2008 R2 StreamInsight”)
# Hardware CEP

<table>
<thead>
<tr>
<th></th>
<th>Software CEP</th>
<th>Hardware CEP [17,21]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>Low 😞 (0.12Gbps) [12]</td>
<td>High 😊 (20Gbps)</td>
</tr>
<tr>
<td><strong>Application range</strong></td>
<td>Broad 😊</td>
<td>Limited 😞</td>
</tr>
</tbody>
</table>

**Events**

- Software CEP: Server
- Hardware CEP: FPGA NIC

---

**Notes:**
- Hardware CEP offers significantly higher performance compared to Software CEP.
- Hardware CEP is more suitable for applications requiring high bandwidth and real-time processing.
Major HW CEP Requirements

- Programmability
- Scalability
Programmability

Software CEPs

- Sybase CCL
- Oracle CQL
- IBM SPL
- EsperTech EPL
- StreamBase
- StreamSQL
- ... etc.

SQL + User-defined functions (C/Java)
Scalability

Event sources

CEP

Results

100 sensor nodes

1k sensor nodes

10k Listed issues in NYSE

Bridge monitoring

Factory monitoring

Financial Trading

Multiple streams
<table>
<thead>
<tr>
<th></th>
<th>Woods [21]</th>
<th>Inoue [17]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>SQL-based</td>
<td>C-based</td>
</tr>
<tr>
<td>Performance</td>
<td>Good (1Gbps)</td>
<td>Good (20Gbps)</td>
</tr>
<tr>
<td>Programmability</td>
<td>No user functions</td>
<td>No SQL interface</td>
</tr>
<tr>
<td>Scalability</td>
<td>Limited (&lt; 1K)</td>
<td>No</td>
</tr>
</tbody>
</table>
Summary of background

Programmability

Scalability

SQL-based

C-based

[21]

[17]
Outline

• Background

• Our work

• Evaluation

• Conclusion
Our Goal

Scalability

10k

1k

1

Programmability

SQL-based [21]

C-based [17]

SQL with C

[266x487]

[541x368]

[161x269]

[210x236]

[358x167]

[388x133]

[300x23]
Two Technical Points

1. SQL interface on the top of C-to-HDL compiler

2. Scalable architecture for multiple streams
Performance came from ... Current “C-to-HDL compiler” technology generates well-optimized circuits

SQL-based: 1Gbps  <  C-based: 20Gbps

SQL-based CEP language

SQL to HDL compiler

C-based CEP language

C to HDL compiler

No industry tool  Use industry tools
Basic Strategy

SQL-based CEP language

SQL to HDL compiler

C-based CEP Language

C to HDL compiler

This work

SQL to C interface

C-based CEP Language

C to HDL compiler

[21]

[17]
SQL’s primitives

Selection

```sql
SELECT *  
WHERE volume > 3200  
From Stock
```

Window, Aggregation

```sql
SELECT stock_id, SUM(volume) AS sum  
From Stock [ROWS 4 PRECEDING]
```

User functions

```sql
SELECT stock_id, calc_vwap() AS vwap  
From Stock [ROWS 4 PRECEDING]
```
Translation rule: selection

Finding events whose volume is greater than 3,200.

**SQL**

```
SELECT *
WHERE volume > 3200
From Stock
```

**C**

```
while(true) {
    entry = event_fifo_in.read();
    if (entry.volume > 3200) {
        event_fifo_out.write(entry);
    }
}
```
Calculating sum of volume of latest 4 events.

**SQL**

```
SELECT stock_id, SUM(volume) AS sum
From Stock [ROWS 4 PRECEDING]
```

**C**

```c
#define WINDOW_SIZE 4
evin_t win[WINDOW_SIZE];
while(true) {
    for(i=1;i<WINDOW_SIZE;i++)
        win[i] = win[i-1];
    win[0] = event_fifo_in.read();
    result.sum = calc_sum_volume(win);
    event_fifo_out.write(result);
}
```
Calculating “volume-weighted average of price” of latest 4 events.

**SQL**

```sql
SELECT stock_id, calc_vwap() AS vwap
From Stock [ROWS 4 PRECEDING]
```

**C**

```c
while(true) {
    ...
    result.vwap = calc_vwap(win);
    event_fifo_out.write(result);
}
```
After translated to C, …

C-to-HDL compilers generate well-parallelized and pipelined circuits.

```c
for(i=0;i<WINDOW_SIZE;i++)
    sum += win[i];
result = sum;
```
## Summary of SQL-to-C interface

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Window</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Matching</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Limited</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>User function</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multiple streams</td>
<td>Limited</td>
<td>No</td>
<td>?</td>
</tr>
</tbody>
</table>
Two Technical Points

1. SQL interface on the top of a C-to-HDL compiler

2. Scalable architecture for multiple streams
Multiple streams

CEP is required to receive multiple streams and to perform event processing.

Stream for stock A
Stream for stock B
Stream for stock C

Naive replication is not applicable for > 100 streams
Observation

Multiple streams are usually interleaved into a stream on a high-speed link

→ an event arrives at a time
Interleaved multiple-context architecture

Interleaved multiple streams

A B C A B ...
## Summary of our work

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Window</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Matching</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Limited</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>User function</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multiple streams</td>
<td>Limited</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Evaluation platform

<table>
<thead>
<tr>
<th>FPGA</th>
<th>Xilinx XC5VLX330T-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAD</td>
<td>NEC CyberWorkBench, Xilinx ISE 12.2</td>
</tr>
</tbody>
</table>
Captures a trend where volume is starting high then it reaches 80%, then calculates “volume-weighted average of price (VWAP)” during the period.
SELECT stock_id, vwap
FROM Stock
MATCH_RECOGNIZE (
    PARTITION BY stock_id
    MEASURES C.stock_id AS stock_id
    C.vwap AS vwap
    PATTERN (A B+ C)
    DEFINE A AS vol_high()
    B AS pri_stbl()
    C AS vol_down()
)
Scalability

FPGA Usage = max(block mem usage/block mem avail, slice usage/slice avail) * 100

500x improvement
Outline

• Background
• Our work
• Evaluation
• Conclusion
Our work

Scalability

Programmability

10k

1k

1

Woods [21]

Inoue [17]

SQL with C