Spark SQL acceleration on FPGA using Apache Arrow and OpenCAPI

Ákos Hadnagy
Accelerated Big Data Systems group
Delft University of Technology, NL
abs.ewi.tudelft.nl
Outline

- The Challenge
- Solution stack
- Integration
- Proof-of-concept
- Results
- Conclusion

- The challenge of accelerating Big Data Analytics
- Spark SQL & Apache Arrow
- Proof-of-concept application
- Results & lessons learned
The Challenge

- Data Scientists & Application Developers
- Hardware Engineers

Knowledge Gap

Big Data Analytics Platforms
- Apache Spark
  - High-level abstractions
  - Distributed execution
  - Resilience

Computing Hardware
- FPGA
  - Low-level hardware description
  - HLS
  - Requires a wide range of expertise
The Challenge

- Solution stack
- Integration
- Proof-of-concept
- Results
- Conclusion

Host-side integration

Memory layout
API

Host-accelerator interface

OpenCAPI™

On-chip architecture

Control structure
Performance scaling
IP reuse
Solution stack

- The Challenge
- Solution stack
- Integration
- Proof-of-concept
- Results
- Conclusion

- Language-agnostic columnar in-memory format
- Hardware-friendly
- Contiguous
- Zero-copy IPC
- Computational libraries

APACHE ARROW

Traditional memory buffer

1331246660
3/8/2012 2:44PM
99.155.155.225
Row 1

1331246351
3/8/2012 2:38PM
65.87.165.114
Row 2

1331244570
3/8/2012 2:09PM
71.10.106.181
Row 3

1331261196
3/8/2012 6:46PM
76.102.156.138
Row 4

Arrow memory buffer

1331246660
3/8/2012 2:44PM
99.155.155.225
session_id

1331246351
3/8/2012 2:38PM
65.87.165.114
timestamp

1331244570
3/8/2012 2:09PM
79.155.155.225
source_ip

1331261196
3/8/2012 6:46PM
76.102.156.138

Host-side integration
Host-accelerator interface
On-chip architecture
Solution stack

- The Challenge
- Solution stack
- Integration
- Proof-of-concept
- Results
- Conclusion

Solution stack

- The Challenge
- Solution stack
- Integration
- Proof-of-concept
- Results
- Conclusion

Solution stack

- The Challenge
- **Solution stack**
- Integration
- Proof-of-concept
- Results
- Conclusion

Host-side integration

- Memory layout
- API

Host-accelerator interface

- OpenCAPI

On-chip architecture

- Control structure
- Performance scaling
- IP reuse

TU Delft
Integration

- module for processing **structured** data with **declarative** queries
- optimizes query before executing

Integration

Feature SPARK-27396 of Spark 3.0
“Columnar Processing Support in public APIs”

- The Challenge
- Solution stack
- Integration
- Proof-of-concept
- Results
- Conclusion
Integration

Adding custom operators to Spark

- The Challenge
- Solution stack
- Integration
- Proof-of-concept
- Results
- Conclusion

Proof-of-concept

**Total duration of all trips conducted by companies starting with the letter ‘b’?**

<table>
<thead>
<tr>
<th>company</th>
<th>trip_seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue Ribbon Taxi Association Inc</td>
<td>236</td>
</tr>
<tr>
<td>Leonard Cab Co</td>
<td>2119</td>
</tr>
<tr>
<td>Blue Ribbon Taxi Association Inc</td>
<td>1458</td>
</tr>
<tr>
<td>Setare Inc</td>
<td>2374</td>
</tr>
</tbody>
</table>

Chicago Taxi Trips, 156 M trips, 2013 to 2019

(https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew)
Proof-of-concept

- The Challenge
- Solution stack
- Integration
- Proof-of-concept
- Results
- Conclusion

```
SELECT SUM('trip_seconds')
FROM 'taxi-company.parquet'
WHERE 'company' rlike '[bB].**'
```

**SQL Query**

**Physical Plan**

**Modified Physical Plan**
Proof-of-concept

- The Challenge
- Solution stack
- Integration
- Proof-of-concept
- Results
- Conclusion

For each ColumnarBatch, memory addresses of arrow buffers are extracted. These addresses are passed to the C++ Arrow abstraction, which then builds a C++ Arrow abstraction.

Fletcher::Kernel::Start

Java/Scala

JNI

Iterator of “intermediate” sums
Proof-of-concept

- The Challenge
- Solution stack
- Integration
- **Proof-of-concept**
- Results
- Conclusion

![Diagram]

- Host
- FPGA
- Schema
- Fletchgen
- OpenCAPI
- FLETCHER
- Spark
- Strings
- Integers
- Results
- Kernel
Proof-of-concept

How can we build or generate it?

- Tydi interfaces [5]
- Operations on streams
- Library components (Regex match)
- Hardware templates for parallel patterns: FilterStream, MapStream, ReduceStream

Composition language: **Tydal**
Proof-of-concept

- The Challenge
- Solution stack
- Integration
- Proof-of-concept
- Results
- Conclusion

Structural implementation

```javascript
impl compositions.Top_level structural {
    matcher: primitives.RegexMatcher[regex := "[bB].*"],
    matcher.in <= this.chars;
    filter_stage: FilterStream(matcher.out);
    reduce.stage: ReduceStream(reduce.op: primitives.Sum);
    filter.stage.in <= this.numbers;
    filter.stage <= reduce.stage;
    this.out <= reduce.stage.out;
}
```

9 lines vs. ~600 lines of boilerplate VHDL
Results

- String stream: 20 bytes/cycle
- Kernel performance:
  - 135M records/s
  - 4.04 GB/s
  - FPGA ~10%
  - Fletcher + kernel: ~3.5%
- Overall speedup: 8x
- 6 kernels would saturate OpenCAPI

*POWER9 system (44 cores, 128 GB RAM, AlphaData ADMPCIE9H7 FPGA); Spark local mode (1 worker) with 16 GB memory assigned, 1 million rows per batch
Conclusions

- Complete workflow for accelerating Big Data workloads
- Integration with Apache Spark using custom executors
  - PoC accelerator: 8x speedup
- Suitable abstractions in every layer -> Structural composition of accelerator HW!
- Mapping of data flow components to HW using library components and HW templates
  - Code size reduction
  - IR for code generation and synthesis
Questions

Ákos Hadnagy
A.Hadnagy@student.tudelft.nl
Accelerated Big Data Systems group
Delft University of Technology, NL
abs.ewi.tudelft.nl

Zaid Al-Ars
Z.Al-Ars@tudelft.nl