Accelerating Pattern Matching Queries in Hybrid CPU-FPGA Architectures

David Sidler, Zsolt István, Muhsen Owaida, Gustavo Alonso
Dept. of Computer Science, ETH Zürich
Increasing amount of user generated data
Increasing amount of user generated data

<table>
<thead>
<tr>
<th>Query (WHERE clause)</th>
<th>Response time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>MonetDB</td>
</tr>
<tr>
<td>LIKE '%Alan%Turing%Cheshire%'</td>
<td>0.02</td>
</tr>
<tr>
<td>REGEXP_LIKE('Alan.*Turing.*Cheshire')</td>
<td>0.36</td>
</tr>
</tbody>
</table>

Databases are not suitable for complex text queries!
Increasing amount of user generated data

<table>
<thead>
<tr>
<th>Query (WHERE clause)</th>
<th>Response time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>MonetDB</td>
</tr>
<tr>
<td>LIKE '%Alan%Turing%Cheshire%'</td>
<td>0.02</td>
</tr>
<tr>
<td>REGEXP_LIKE('Alan.*Turing.*Cheshire')</td>
<td>0.36</td>
</tr>
</tbody>
</table>

Databases are not suitable for complex text queries!
Accelerators to the rescue

- Using GPUs [1,2] or Xeon Phi [3] to accelerate string matching:
  - High speed-up
  - Data already on accelerator or data movement reduces acceleration benefit
  - Change of data layout
  - Performance depends on pattern complexity

Accelerators to the rescue

- Using GPUs [1,2] or Xeon Phi [3] to accelerate string matching:
  - High speed-up
  - Data already on accelerator or data movement reduces acceleration benefit
  - Change of data layout
  - Performance depends on pattern complexity

- Integration into database engine often unclear

Accelerators to the rescue

- Using GPUs [1,2] or Xeon Phi [3] to accelerate string matching:
  - High speed-up
  - Data already on accelerator or data movement reduces acceleration benefit
  - Change of data layout
  - Performance depends on pattern complexity
- Integration into database engine often unclear

Data partitioning/movement hinders wide-spread adoption of database accelerators!

New hybrid architectures are emerging

IBM Power8 + CAPI

Source: Heterogeneous computing on POWER, Cesar Diniz Maciel, IBM

Intel Xeon+FPGA

Source: Intel Xeon+FPGA Platform for the Data Center, PK Gupta, Intel
New hybrid architectures are emerging

IBM Power8 + CAPI

Source: Heterogeneous computing on POWER, Cesar Diniz Maciel, IBM

Intel Xeon+FPGA

Source: Intel Xeon+FPGA Platform for the Data Center, PK Gupta, Intel

Eliminate the issue of data movement/partitioning
Intel Xeon+FPGA prototype platform

Version 1 (used in this work)

Read-heavy: 6.5 GB/s
Read/Write: 3 GB/s

Disclaimer
This is an experimental system provided by Intel any results presented are generated using pre-production hardware and software, and may not reflect the performance of production or future systems.
Intel Xeon+FPGA prototype platform

Version 1 (used in this work)

- Memory
- Mem. controller
  - Xeon E5
- QPI
- FPGA cache
  - User Logic
  - Stratix V
- Read-heavy: 6.5 GB/s
- Read/Write: 3 GB/s

Version 2

- Larger bandwidth (1xQPI, 2xPCI)
- Larger FPGA
- FPGA in same package (single socket)

Disclaimer
This is an experimental system provided by Intel; any results presented are generated using pre-production hardware and software, and may not reflect the performance of production or future systems.
Intel Xeon+FPGA prototype platform

Version 1 (used in this work)

- Memory
  - Read-heavy: 6.5 GB/s
  - Read/Write: 3 GB/s

Version 2

- Larger bandwidth (1xQPI, 2xPCI)
- Larger FPGA
- FPGA in same package (single socket)

Disclaimer

This is an experimental system provided by Intel any results presented are generated using pre-production hardware and software, and may not reflect the performance of production or future systems.
FPGA (Field Programmable Gate Array)

- Reprogrammable, load arbitrary circuits onto the FPGA
- Once programmed acts similar to an integrated circuit (lower frequency)
- Logic blocks (around 100,000)
- Fast on-chip memory (36K each)
Parameterizable Regular Expression Engine
Regular Expression in Hardware

- Regex can be mapped to a Non-deterministic finite automata (NFA)
- NFAs can be efficiently executed on FPGAs [4,5]

Regular expression: \((ab^+|ba^+)c\)

Input:

Regular Expression in Hardware

- Regex can be mapped to a Non-deterministic finite automata (NFA)
- NFAs can be efficiently executed on FPGAs [4,5]

Regular expression: \((ab^+ | ba^+)c\)

Input:

```
S0 → S1 → S2 → S3 → S4
```

```
S0 → S1 → S2 → S3 → S4 → S5
```

Regular Expression in Hardware

- Regex can be mapped to a Non-deterministic finite automata (NFA)
- NFAs can be efficiently executed on FPGAs [4,5]

**Regular expression:** \((ab+|ba+)c\)

**Input:** a

---


Regular Expression in Hardware

- Regex can be mapped to a Non-deterministic finite automata (NFA)
- NFAs can be efficiently executed on FPGAs [4,5]

Regular expression: \((ab^+|ba^+)c\)

Input: \(ad\)

Regular Expression in Hardware

- Regex can be mapped to a Non-deterministic finite automata (NFA)
- NFAs can be efficiently executed on FPGAs [4,5]

Regular expression: \((ab+|ba+)c\)

Input: ada

Regular Expression in Hardware

- Regex can be mapped to a Non-deterministic finite automata (NFA)
- NFAs can be efficiently executed on FPGAs [4,5]

Regular expression: \((ab^+|ba^+)c\)
Input: adab

Regular Expression in Hardware

- Regex can be mapped to a Non-deterministic finite automata (NFA)
- NFAs can be efficiently executed on FPGAs [4,5]

Regular expression: \((ab+|ba+)c\)
Input: adabb

Regular Expression in Hardware

- Regex can be mapped to a Non-deterministic finite automata (NFA)
- NFAs can be efficiently executed on FPGAs [4,5]

**Regular expression:** \((ab+|ba+)c\)

**Input:** adabbb

![Non-deterministic finite automata diagram]

Regular Expression in Hardware

- Regex can be mapped to a Non-deterministic finite automata (NFA)
- NFAs can be efficiently executed on FPGAs [4,5]

**Regular expression:** \((ab^+|ba^+)c\)

**Input:** adabbbc

---

Complexity vs Hardware resources

Regular expression: SIGMOD.*(Chicago|Raleigh)

- Resource usage and routing are crucial factors in FPGA development.
- FPGA resource usage grows with regular expression complexity.
- If the NFA becomes too large routing/connecting its resources might not be possible.

⇒ Compress the NFA
NFA compression

Regular expression: SIGMOD.*(Chicago|Raleigh)

**NFA compression**

**Regular expression:** \(\text{SIGMOD.*}(\text{Chicago}|\text{Raleigh})\)

Extracted sequences:
- SIGMOD
- Chicago
- Raleigh

NFA compression

Regular expression: SIGMOD.*(Chicago|Raleigh)

Extracted sequences:
- SIGMOD
- Chicago
- Raleigh

NFA compression

Regular expression: \text{SIGMOD}.*(Chicago|Raleigh)

Extracted sequences:
- SIGMOD
- Chicago
- Raleigh

- Decouple character encoding from state transitions in NFA [6]

Character Encoder

- Enables compression of NFA by chaining characters into sequences
- Can check for ranges by comparing upper and lower value
- Can support case-insensitivity or collations (e.g., a, ae, ä)
Character Encoder

- Enables compression of NFA by chaining characters into sequences
- Can check for ranges by comparing upper and lower value
- Can support case-insensitivity or collations (e.g., a, ae, ä)

Character Encoder can be parametrized at runtime.
Runtime parametrization

Regular expression: \((a|b) . c\)

State Graph (fully connected)

Encoder

State Transitions

Triggers

Characters

State Graph (fully connected)
Runtime parametrization

Regular expression: $(a|b)^*c$

State Graph (fully connected)
Runtime parametrization

Regular expression:
\((a|b)\).*c

State Graph (fully connected)

Input character

Characters

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>'a'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'b'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'c'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Triggers

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

State Transitions

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Runtime parametrization

Regular expression:

\[(a|b)*c\]

State Graph (fully connected)

Characters

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>'a'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'b'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'c'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Triggers

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

State Transitions

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Configuration vector

Characters

C1  C2  C3  C4

'a' 'b' 'c'

Triggers

S1  C1  C2  C3  C4
S1  1   1   0   0
S2  0   0   0   0
S3  0   0   1   0
S4  0   0   1   0

State Transitions

S1  S2  S3  S4
S1  1   0   0   0
S2  0   0   0   0
S3  0   0   0   0
S4  1   0   0   0

Parametrization

0x61  0x62  0x63  0x00  0xC0  0x32  0x80  0x08  \ldots

Configuration vector

Systems Group, Dept. of Computer Science, ETH Zürich

SIGMOD 2017 | May 16, 2017 | 13 / 32
Configuration vector

Characters

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>'a'</td>
<td>'b'</td>
<td>'c'</td>
<td></td>
</tr>
</tbody>
</table>

Triggers

<table>
<thead>
<tr>
<th>S1</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

State Transitions

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Configuration of Regex Engine takes only 2 cycles.
Assembly of a Regex Engine

Input Fifo -> NFA -> Result Fifo

String router

Input Fifo -> NFA -> Result Fifo

... -> ... -> ...

Result Merger

Match, pos=12

No Match

JohnSmith123

Barbara0

JohnSmit

ara0Alex

h123Barb
Integration into Database
Integration into MonetDB

- Column store
- Simple data layout
- Minimize memory bandwidth overhead
- UDF can operate on columns
- Strings are stored in a heap

<table>
<thead>
<tr>
<th>OID</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
System Overview

Centaur [7] bridges the gap between the database and the hardware operators. Each engine can process at 6.4 GB/s.

 Systems Group, Dept. of Computer Science, ETH Zürich
System Overview

Database extended with a HUDF

Database extended with a HUDF

Centaur [7] bridges the gap between the database and the hardware operators

System Overview

Centaur [7] bridges the gap between the database and the hardware operators.

Each engine can process at 6.4 GB/s.


Database extended with a HUDF

MonetDB

Centaur

CPU-FPGA Shared Memory

Job Dist.

Centaur Eng 1

Centaur Eng 2

Centaur Eng 3

Centaur Eng 4

CPU

FPGA

MonetDB columns

Job Queue

UDF

Database extended with a HUDF

Systems Group, Dept. of Computer Science, ETH Zürich
Execution Walkthrough

1. Query containing regular expression is submitted

CPU-FPGA Shared Memory

- MonetDB columns
- Job Queue
- Parameters
- Status

CPU

- MonetDB
- UDF
- Centaur

FPGA

- Centaur
- Job Dist.
- Regex Eng 1
- Regex Eng 2
- Regex Eng 3
- Regex Eng 4

User Query
Execution Walkthrough

1. Query containing regular expression is submitted
2. MonetDB calls the Hardware UDF
Execution Walkthrough

1. Query containing regular expression is submitted
2. MonetDB calls the Hardware UDF
3. UDF converts the regular expression into a configuration vector and allocates the result column
Centaur allocates memory for the job parameters and job status
Centaur allocates memory for the job parameters and job status

Job is enqueued into job queue
Centaur allocates memory for the job parameters and job status

Job is enqueued into job queue

Job Distributor fetches the job from the job queue and assigns it to an idle Regex Engine
7 Regex Engine reads parameters from shared memory, configures itself with the configuration vector and starts execution.
Execution Walkthrough

1. User Query
2. CPU
3. MonetDB
4. UDF
5. Regex Engine
6. Centaur
7. Regex Engine reads parameters from shared memory, configures itself with the configuration vector and starts execution.
8. After termination the done bit is set.
7. Regex Engine reads parameters from shared memory, configures itself with the configuration vector and starts execution.

8. After termination the done bit is set.

9. UDF waits on the done bit and then hands the result column over to MonetDB.
Evaluation
Evaluation - Queries

Q1: \begin{verbatim}
SELECT count(*) FROM address_table
WHERE address_string LIKE '%Strasse%';
\end{verbatim}

Q2: \begin{verbatim}
SELECT count(*) FROM address_table
WHERE REGEXP_LIKE(address_string, '(Strasse|Str\.).*(8[0-9]{4})');
\end{verbatim}

Q3: \begin{verbatim}
SELECT count(*) FROM address_table
WHERE REGEXP_LIKE(address_string, '[0-9]+(USD|EUR|GBP)');
\end{verbatim}

Q4: \begin{verbatim}
SELECT count(*) FROM address_table
WHERE REGEXP_LIKE(address_string, '[A-Za-z]{3}\:[0-9]{4}');
\end{verbatim}
Evaluation - Microbenchmark

Query over a small relation with 10K tuples

Time [ms]

Database
UDF (software part)
Config. Gen.
Centaur
Hardware Processing
Evaluation - Throughput

Number of Clients

Queries/s

MonetDB - FPGA

DBx - FPGA
Evaluation - TPC-H Q13

Scaling factor set to 0.1 due to limited memory space

MonetDB | FPGA
--- | ---
Original | Case-Insensitive
## Comparison to Accelerators

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Regex evaluation</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Complexity indp. perf.</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>TP - local data [GB/s]</td>
<td>60-70</td>
<td>10-15</td>
<td>30-40</td>
<td>25.6*</td>
</tr>
<tr>
<td>TP - host data [GB/s]</td>
<td>–</td>
<td>1-5</td>
<td>–</td>
<td>6.4</td>
</tr>
<tr>
<td>Architecture</td>
<td>fast GDDR memory</td>
<td>fast GDDR memory</td>
<td>GDDR5 memory</td>
<td>specialized core, direct memory access</td>
</tr>
</tbody>
</table>

* Without the memory bandwidth limitation

## Comparison to Accelerators

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Regex evaluation</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Complexity indp. perf.</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>TP - local data [GB/s]</td>
<td>60-70</td>
<td>10-15</td>
<td>30-40</td>
<td>25.6*</td>
</tr>
<tr>
<td>TP - host data [GB/s]</td>
<td>–</td>
<td>1-5</td>
<td>–</td>
<td>6.4</td>
</tr>
<tr>
<td>Architecture</td>
<td>fast GDDR memory</td>
<td>fast GDDR memory</td>
<td>60-70 cores, GDDR5 memory</td>
<td>specialized core, direct memory access</td>
</tr>
</tbody>
</table>

* Without the memory bandwidth limitation

---

**Your next CPU might come with an FPGA!**


---

**Your next CPU might come with an FPGA!**
Visit our Demo!

More Information:
systems.ethz.ch/fpga/db_acceleration

Code on GitHub:
github.com/fpgasystems/dobbiodb