SQL Query Processing Using an Integrated FPGA-based Near-Data Accelerator in ReProVide

Demo Paper

Lekshmi B.G., Andreas Becher, Klaus Meyer-Wegener, Stefan Wildermann, Jürgen Teich
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Erlangen, Germany
{lekshmi.bg.nair,andreas.becher,klaus.meyer-wegener,stefan.wildermann,juergen.teich}@fau.de

ABSTRACT

In this demo, we explain the working of ReProVide, a framework that integrates an on-the-fly reconfigurable FPGA-based SoC architecture with a DBMS for accelerated query processing. For this, a capability-aware optimization that can optimize and partition the queries is demonstrated. This optimization for the hardware is based on its capabilities. Our hardware can also generate statistics of the data while executing a query. In contrast to the existing approaches, this does not have any additional costs in terms of execution time. We will demonstrate how these statistics are later used by the DBMS to select the best plan from the search space using accurate cost values.

1 INTRODUCTION

Current trends in hardware technologies such as manycores, GPUs and field-programmable gate arrays (FPGAs) for data processing, NVRAM and open-channel SSDs for storage solutions and RDMA-capable high-speed network solutions are interesting candidates for the acceleration of database query processing on Big Data. In this scenario, the German DFG Priority Program no. 2037[1] on "Scalable Data Management for Future Hardware", funds research projects to develop new architectural concepts for data management systems that support both current as well as future hardware technologies. Our Reconfigurable data provider (ReProVide) project is one of them. The goal of the ReProVide project is to provide a new FPGA-based smart storage solution together with query optimization techniques, which considers the capabilities of the hardware for the scalable and powerful near-data processing of Big Data. The benefits of FPGA technology are pipelined processing of data at line rate (at least for non-blocking operators), energy-efficiency and speedup through parallelization, as well as the option of dynamic adaptation of the hardware through reconfiguration. In the ReProVide project, a generic FPGA architecture named ReProVide processing unit (RPU) for the efficient processing of database queries is developed. The goal of such an architecture is to serve as intelligent storage system and reconfigurable data (pre-) processing interface between diverse data sources and host systems requesting data from these sources. This can reduce network utilization and host workload as well as saving power to a great extent.

Integrating smart storage like RPUs into a database management system (DBMS) requires novel optimization techniques to (a) decide which operations are worthwhile to assign to the RPU, and which are not (*query partitioning* [2]) and (b) to determine how to map and execute the assigned (sub-)queries or database operators on the RPU (*query placement* [1]). For query partitioning, the DBMS query optimizer must take the characteristics of an RPU system into account. This involves considering which particular operators are actually supported by the RPU as not all operators and data types can be accelerated at line rate on FPGAs. It also includes using novel cost models for estimating the RPU performance which consider hardware reconfiguration overheads and make use of hardware-generated statistics of the data it currently stores. We therefore also investigate a novel hierarchical (multi-level) query-optimization technique where the global optimizer of the DBMS and the architecture-specific local optimizer of the RPU work together to obtain scalable query processing [2]. A bidirectional hint interface (see Fig. 1b) exchanging hints between global and local optimizer enables scalable optimization in both directions. i. e., globally and locally.

In this demo, we show how DBMS and RPU work together to process database queries with the goal of minimizing the overall execution time. We will demonstrate how our techniques abstract away the complex decision space of how to partition and execute queries on heterogeneous hardware. We furthermore will explain the underlying hardware concepts and optimization techniques.

2 SYSTEM OVERVIEW

2.1 ReProVide

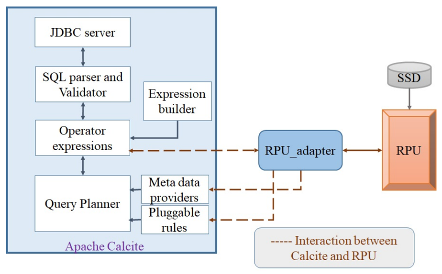
The ReProVide system is a distributed/federated data-processing system which includes Apache Calcite as the DBMS as well as FPGA-based hardware (RPU) as the co-processor and the remote data storage (see Fig. 1a). ReProVide follows the idea of near-data processing, which means the RPU has the query-processing capabilities (near to data storage), that can be used for data (pre-)processing.

Apache Calcite[2] is a dynamic data-management framework that provides query processing, optimization, and query-language support to many data-processing system [5]. It is a complete query-processing system, except for data storage. The framework that Calcite provides for building databases allows to extend its set of rules, cost models, relational expressions, and user-defined functions. This property of Calcite has encouraged us to choose it as our DBMS for this project. Calcite provides data-provider federation through adapters. Hence it is possible to connect any number of databases to Calcite using their own adapter.

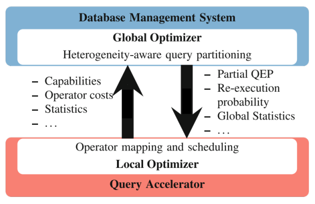
An RPU is implemented on a programmable SoC (multi-core CPU + FPGA) that serves as a data provider accessible via an Ethernet network. The data is stored in non-volatile storage such as solid-state disks (SSDs) and in volatile memory such as DDR-SDRAM, which is directly attached to and managed by the platform and can be processed by accelerators implemented

[1]https://www.dfg-spp2037.de/

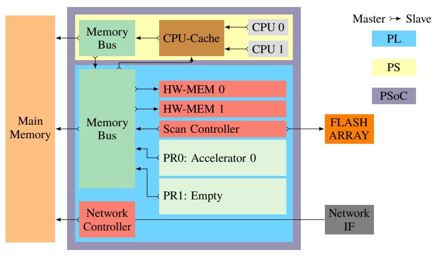
[2]https://calcite.apache.org/



(a) ReProVide architecture



(b) Interplay of Global and Local Optimizer



(c) The architecture of an RPU [1]

Figure 1: (1a) shows the architecture diagram of ReProVide system. (1b) gives the interaction between co-operating optimizers in ReProVide. (1c) illustrates the RPU with the FPGA part (programmable logic (PL)) of the system on chip (SoC) colored blue and the processing system (PS) with its dual core processor colored yellow. Two partially reconfigurable regions (PRs) are available with *PR0* being configured with *Accelerator 0*.

in the Programmable Logic (PL/FPGA) part of the system. This design also enables an RPU to locally optimize the storage layout of the data with regards to availability, access latency, power consumption, and the access patterns of the hardware accelerators.

The interaction of the DBMS and ReProVide is sketched in Fig. 1b. Global optimization applies rules to the query-evaluation plan (QEP), with the major goal of partitioning the QEP into sub-trees and assigning some of them to the RPU (query partitioning). The local optimizer is responsible for finding the best implementation of a sub-tree on the RPU (query placement [1]).

The execution of a query mainly consists of three phases (see Fig. 2) : In the first phase, the local optimizer provides information on its capabilities. The global optimizer uses this information to decide, which operations of a QEP can be assigned to the RPU. Based on this the query will be partitioned. For finding the best partitioning, statistics of the data stored in the RPU can be requested. Generation of such statistics on streaming hardware comes at almost no cost [3] and can even be refined when subsequently accessing the same data (see section 4.2).

In the second phase, the sub-tree of QEP (partition) is passed from the global optimizer to the local optimizer, including some hints (e.g. re-execution probability, upper latency bound, a time budget for running the local optimizer). The local optimizer then selects the required accelerators, maps query operations to these accelerators, and schedules the reconfiguration of the accelerators specific to the operations that it receives. Based on the hints, the local optimizer can adjust buffer sizes for higher throughput but decreased latency until the first data is sent.

In the third and final phase, the query is executed and the results are returned to the host system.

2.2 RPU

The RPU constitutes a heterogeneous hardware/software system, as shown in Figure 1c. The table management is executed on one core of the processing system (PS). While the processing system could also process the full variety of operators and types, its performance may be limited. Thus, handling of the data is mostly dealt with within the programmable logic (PL). Requests are received via a high-speed network interface and forwarded to the management software for further processing. To relieve the processing system from the task of result transmission, a specialized Network Controller implemented in the static system allows sending the resulting data to the requesting host with minimum intervention from the management software.

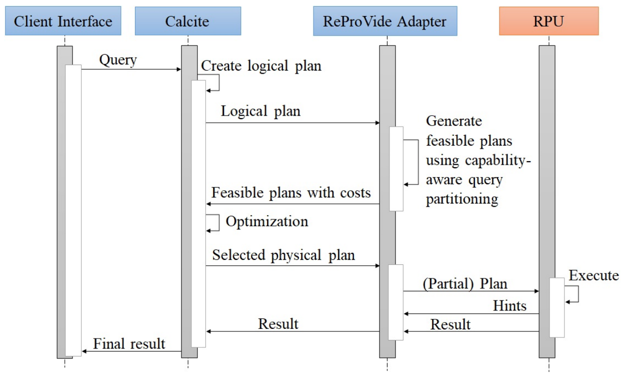


Figure 2: Sequence Diagram for the demonstrated query processing

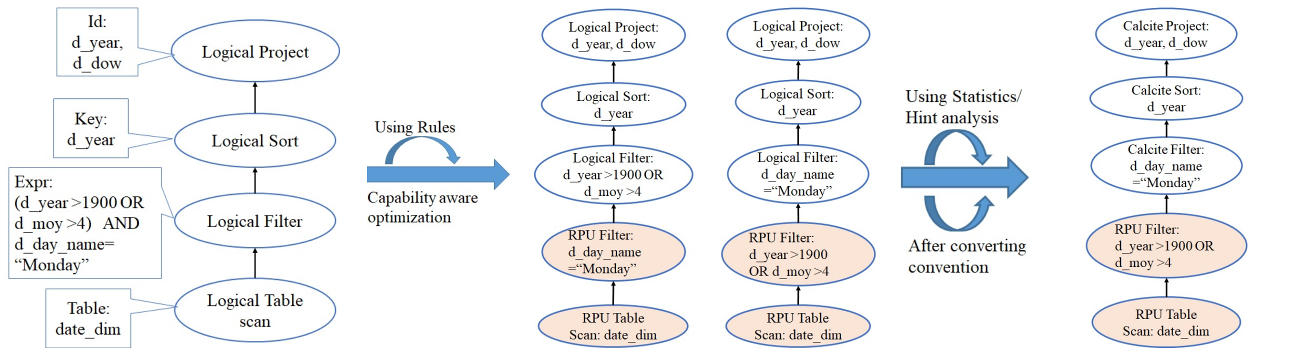
RPUs include partially reconfigurable regions (PR) into which hardware accelerators can be dynamically loaded and which can be configured to process operations on streaming data. By means of hardware reconfiguration, different hardware accelerators can be loaded onto the FPGA for processing the locally-stored data before transmitting it to the requesting DBMS.

A dedicated and parameterizable hardware component called *Scan Controller* is responsible for data loading. The controller also has direct access to the attached FLASH storage to stream the relevant data to the accelerators. RPUs make use of a library of pre-synthesized reconfigurable hardware accelerators for query placement as the dynamic synthesis of a new hardware accelerator (query compilation) can take from minutes up to multiple hours. As such, the capabilities of a RPU are determined by the accelerators in the library: Operators for which no accelerators are available in the library cannot be executed on the RPU. Operators available span from simple filters on integer values to more complex filters such as regular expressions. Also accelerators for hash-joins have been developed. Please see [4, 15].

2.3 RPU-enabled DBMS

The implemented RPU-enabled DBMS is prototyped using Apache Calcite. We have implemented the *Global Optimizer* as follows.

Calcite allows to extend its rules for query optimization via adapters. The global optimizer must be tuned to act as a capability-aware optimizer by defining some additional rules (push-down



Id: d_year, d_dow
Logical Project
Key: d_year
Logical Sort
Expr: (d_year>1900 OR d_moy>4) AND d_day_name= "Monday"
Logical Filter
Table: date_dim
Logical Table scan
Using Rules
Capability aware optimization
Logical Project: d_year, d_dow
Logical Sort: d_year
Logical Filter: d_year>1900 OR d_moy>4
RPU Filter: d_day_name ="Monday"
RPU Table Scan: date_dim
Logical Project: d_year, d_dow
Logical Sort: d_year
Logical Filter: d_day_name ="Monday"
RPU Filter: d_year>1900 OR d_moy>4
RPU Table Scan: date_dim
Using Statistics/ Hint analysis
After converting convention
Calcite Project: d_year, d_dow
Calcite Sort: d_year
Calcite Filter: d_day_name ="Monday"
RPU Filter: d_year>1900 OR d_moy>4
RPU Table Scan: date_dim

(a) Logical plan. **(b) Feasible plans.** **(c) Selected physical plan.**

Figure 3: Query optimization by global optimizer.

rules) to consider the capabilities of RPU during the query optimization. These push-down rules define required operations and attributes which can be pushed down to RPU. The main purpose of the attribute push-down is to avoid unnecessary data transfer over the network and thus to reduce network traffic as well as host workload. These push-down rules are not RPU specific, and can be applied with any hardware configuration connected with Calcite but require the capabilities of the connected co-processor. As an example, consider the query below:

```
SELECT d_year, d_dow
FROM date_dim
WHERE d_day_name="Monday" AND
(d_year > 1900 OR d_moy > 4 )
ORDER BY d_year
```

Fig. 3a. shows the logical plan of the given query. The global optimizer (GO) partitions the query using push-down rules. In this example, the RPU is only capable (a) of doing Filter operations in the Where clause, so the Sort operation cannot be assigned to the RPU, and (b) it can either do String comparison or Integer comparison at a time. Hence, the Filter node is partitioned into a String comparison and an Integer comparison node. Only one can be pushed down to the RPU, so two plans are generated acc. to Fig.3b. Based on cost models and statistics, the GO now selects one alternative and converts it into a physical plan (see Fig. 3c).

3 EVALUATION

Initial measurements of our system were performed on Intel Core i9-7920x processor with CPU 2.90GHz × 24 and Memory 64 GiB. For comparison, PostgreSQL executed on the same system is used as a baseline for comparison. Please note that for the PostgreSQL baseline the data is available locally and, in contrast to our setup, has no network overhead involved. For the ReProVide setup, the RPU is connected via a 1Gb/s Ethernet. Both, the RPU and PostgreSQL are connected to Calcite using their respective software drivers. To evaluate the performance of our proposed system, we chose two different SELECT statements and varied the WHERE clause to test the influence of selectivity. As Table we used the date dimension of the TPC-DS benchmark suite. Figure 4 depicts the relative execution time difference of ReProVide versus PostgreSQL. Due to the run-to-completion implementation of this early version of the RPUs, we can observe a slowdown for the query (blue) with bigger tuple sizes of the result table: Overhead for allocating and synchronizing buffers depends on their sizes.

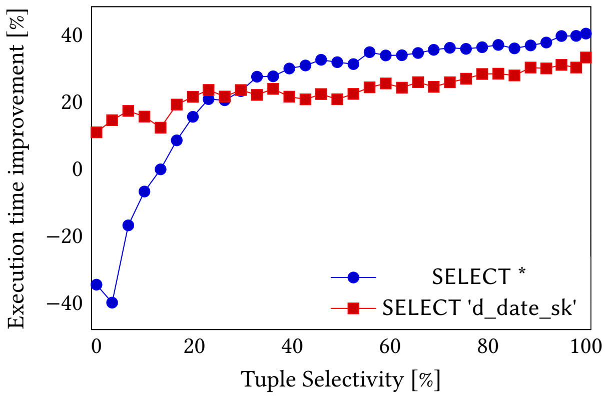


Figure 4: Relative execution time improvement vs. Tuple Selectivity. The improvement is calculated as $\frac{time_{PostgreSQL} - time_{ReProVide}}{time_{PostgreSQL}}$. One query (blue) selects all available attributes of the TPC-DS date dimension while the second query (red) only selects a single attribute.

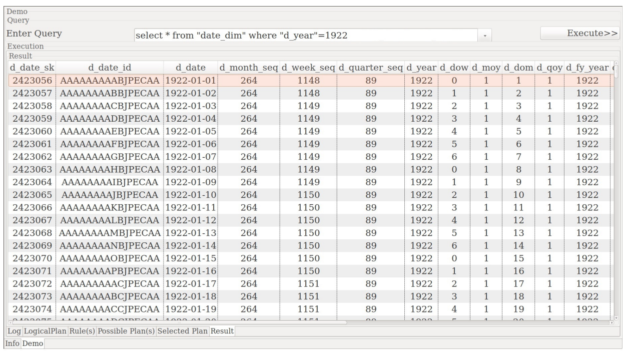
This overhead is greatly reduced if only a small part (columns) will be present in the intermediate result (red) or the selectivity rises and the buffers become less and less oversized. Furthermore, as the RPU is network connected there is an initial latency to send the request and receive the results which PostgreSQL running in the same host doesn't suffer from. As one can see, even with PostgreSQL having these benefits and higher data transfer rates, our approach already shows benefits in terms of execution time reduction of up to 40%.

4 DEMONSTRATION WALKTHROUGH

For the demonstration, we will present all the above mentioned concepts to the conference attendees. They will be explained how to use our framework for executing their own queries. They do not require any prerequisite understanding about any of the hardware or software that we are using for demonstration. The demo includes a laptop where the DBMS (particularly Calcite) is running which will be connected with an FPGA board (RPU) using an Ethernet cable.

4.1 Query partition and execution

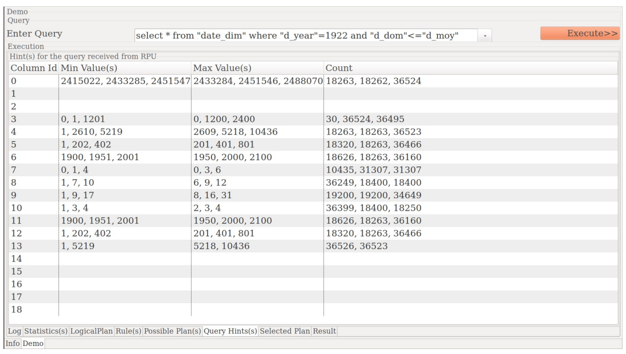
The conference attendees can select/ frame a query that they want to execute in our framework. When the attendee types in a query and presses the execute button, the query execution log will be displayed. The push-down rules are listed and possible execution plans generated by the global optimizer using those push-down rules are visualized. Furthermore, the selected plan for the execution and, finally, the result of the query (as depicted by Fig 5) are shown.



Enter Query: select * from "date_dim" where "d_year"=1922

d_date_sk	d_date_id	d_date	d_month_seq	d_week_seq	d_quarter_seq	d_year	d_dow	d_moy	d_dom	d_qoy	d_fy_year
2423056	AAAAAAAAABJPECAA	1922-01-01	264	1148	89	1922	0	1	1	1	1922
2423057	AAAAAAAABBJPECAA	1922-01-02	264	1148	89	1922	1	1	2	1	1922
2423058	AAAAAAAACBJPECAA	1922-01-03	264	1149	89	1922	2	1	3	1	1922
2423059	AAAAAAAADBJPECAA	1922-01-04	264	1149	89	1922	3	1	4	1	1922
2423060	AAAAAAAAEBJPECAA	1922-01-05	264	1149	89	1922	4	1	5	1	1922
2423061	AAAAAAAAFBJPECAA	1922-01-06	264	1149	89	1922	5	1	6	1	1922
2423062	AAAAAAAAGBJPECAA	1922-01-07	264	1149	89	1922	6	1	7	1	1922
2423063	AAAAAAAAHBJPECAA	1922-01-08	264	1149	89	1922	0	1	8	1	1922
2423064	AAAAAAAAIBJPECAA	1922-01-09	264	1149	89	1922	1	1	9	1	1922
2423065	AAAAAAAAJBJPECAA	1922-01-10	264	1150	89	1922	2	1	10	1	1922
2423066	AAAAAAAAKBJPECAA	1922-01-11	264	1150	89	1922	3	1	11	1	1922
2423067	AAAAAAAALBJPECAA	1922-01-12	264	1150	89	1922	4	1	12	1	1922
2423068	AAAAAAAAMBJPECAA	1922-01-13	264	1150	89	1922	5	1	13	1	1922
2423069	AAAAAAAANBJPECAA	1922-01-14	264	1150	89	1922	6	1	14	1	1922
2423070	AAAAAAAAOBJPECAA	1922-01-15	264	1150	89	1922	0	1	15	1	1922
2423071	AAAAAAAAPBJPECAA	1922-01-16	264	1150	89	1922	1	1	16	1	1922
2423072	AAAAAAAAACJPECAA	1922-01-17	264	1151	89	1922	2	1	17	1	1922
2423073	AAAAAAAABCJPECAA	1922-01-18	264	1151	89	1922	3	1	18	1	1922
2423074	AAAAAAAACCJPECAA	1922-01-19	264	1151	89	1922	4	1	19	1	1922

Figure 5: Final results



Enter Query: select * from "date_dim" where "d_year"=1922 and "d_dom"<="d_moy"

Column Id	Min Value(s)	Max Value(s)	Count
0	2415022, 2433285, 2451547	2433284, 2451546, 2488070	18263, 18262, 36524
1			
2			
3	0, 1, 1201	0, 1200, 2400	30, 36524, 36495
4	1, 2610, 5219	2609, 5218, 10436	18263, 18263, 36523
5	1, 202, 402	201, 401, 801	18320, 18263, 36466
6	1900, 1951, 2001	1950, 2000, 2100	18626, 18263, 36160
7	0, 1, 4	0, 3, 6	10435, 31307, 31307
8	1, 7, 10	6, 9, 12	36249, 18400, 18400
9	1, 9, 17	8, 16, 31	19200, 19200, 34649
10	1, 3, 4	2, 3, 4	36399, 18400, 18250
11	1900, 1951, 2001	1950, 2000, 2100	18626, 18263, 36160
12	1, 202, 402	201, 401, 801	18320, 18263, 36466
13	1, 5219	5218, 10436	36526, 36523
14			
15			
16			
17			
18			

Figure 6: Statistics received from RPU

4.2 Online Statistics Visualization

One speciality of the RPU is the possibility to generate statistics such as histograms at almost zero cost during regular query execution runs [3]. The demo will present the refined histograms after new information is available in an event driven way to demonstrate that statistics can be refined without additional cost (see Fig 6). The statistics can then guide the optimizer to select the better plans with more accurate cost estimations.

5 RELATED WORKS

Research about the benefits of modern hardware, especially FPGAs and GPUs, for query accelerations is well exploited in recent years. [11–14] are studying the performance improvement when integrating FPGAs into CPU systems for query acceleration. Contrary, we are introducing a co-operating optimizer, where FPGA and CPU are interacting. Based on this interaction, a capability-aware optimization is implemented.

In [7] the statistics (histogram) have been generated as a side effect of query processing but with additional hardware resources while in our system, statistics are gathered during the execution of partial query[3].

Heterogeneity-aware query optimization has been studied extensively in [6, 8–10]. But other than the approaches they have implemented, a capability-aware optimization using the hints from the attached co-processor is used in our approach to find the best operations for them.

6 CONCLUSION AND FUTURE WORK

A reconfigurable near-data accelerator and a capability-aware global optimizer for a scalable and energy efficient execution of an SQL query processing is shown in this demo. In the current version, we are focusing on the query partition and on-the-fly reconfiguration of accelerators. In the future work, we would like to address multi-query optimization and common sub-expression evaluation.

Acknowledgements. This work has been supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) as part of the SPP 2037.

REFERENCES

[1] Andreas Becher, Achim Herrmann, Stefan Wildermann, and Jürgen Teich. 2019. ReProVide: Towards Utilizing Heterogeneous Partially Reconfigurable Architectures for Near-Memory Data Processing. In *BTW 2019 – Workshopband*, Holger Meyer, Norbert Ritter, Andreas Thor, Daniela Nicklas, Andreas Heuer, and Meike Klettke (Eds.). Gesellschaft für Informatik, Bonn, 51–70. https://doi.org/10.18420/btw2019-ws-04

[2] Andreas Becher, BG Lekshmi, David Broneske, Tobias Drewes, Bala Gurumurthy, Klaus Meyer-Wegener, Thilo Pionteck, Gunter Saake, Jürgen Teich, and Stefan Wildermann. 2018. Integration of FPGAs in Database Management Systems: Challenges and Opportunities. *Datenbank-Spektrum* 18, 3 (2018), 145–156.

[3] Andreas Becher and Jürgen Teich. 2019. In situ Statistics Generation within partially reconfigurable Hardware Accelerators for Query Processing. In *15th International Workshop on Data Management on New Hardware (DaMoN) Held with ACM SIGMOD/PODS 2019* (2019-07-01/2019-07-01).

[4] Andreas Becher, Daniel Ziener, Klaus Meyer-Wegener, and Jürgen Teich. 2015. A co-design approach for accelerated SQL query processing via FPGA-based data filtering. In *2015 International Conference on Field Programmable Technology, FPT 2015, Queenstown, New Zealand, December 7-9, 2015.* 192–195. https://doi.org/10.1109/FPT.2015.7393148

[5] Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J Mior, and Daniel Lemire. 2018. Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 221–230.

[6] Sebastian Breß, Bastian Köcher, Max Heimel, Volker Markl, Michael Saecker, and Gunter Saake. 2014. Ocelot/HyPE: optimized data processing on heterogeneous hardware. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1609–1612.

[7] Zsolt Istvan, Louis Woods, and Gustavo Alonso. 2014. Histograms as a side effect of data movement for big data. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 1567–1578.

[8] Tomas Karnagel, Dirk Habich, and Wolfgang Lehner. 2015. Local vs. Global Optimization: Operator Placement Strategies in Heterogeneous Environments.. In *EDBT/ICDT Workshops*. 48–55.

[9] Tomas Karnagel, Dirk Habich, and Wolfgang Lehner. 2017. Adaptive work placement for query processing on heterogeneous computing resources. *Proceedings of the VLDB Endowment* 10, 7 (2017), 733–744.

[10] Tomas Karnagel, Dirk Habich, Benjamin Schlegel, and Wolfgang Lehner. 2014. Heterogeneity-aware operator placement in column-store DBMS. *Datenbank-Spektrum* 14, 3 (2014), 211–221.

[11] Mohammadreza Najafi, Mohammad Sadoghi, and Hans-Arno Jacobsen. 2013. Flexible query processor on FPGAs. *Proceedings of the VLDB Endowment* 6, 12 (2013), 1310–1313.

[12] Muhsen Owaida, David Sidler, Kaan Kara, and Gustavo Alonso. 2017. Centaur: A framework for hybrid CPU-FPGA databases. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 211–218.

[13] David Sidler, Zsolt István, Muhsen Owaida, and Gustavo Alonso. 2017. Accelerating pattern matching queries in hybrid CPU-FPGA architectures. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 403–415.

[14] Louis Woods, Zsolt István, and Gustavo Alonso. 2014. Ibex: an intelligent storage engine with support for advanced SQL offloading. *Proceedings of the VLDB Endowment* 7, 11 (2014), 963–974.

[15] Daniel Ziener, Helmut Weber, Jörg-Stephan Vogt, Ute Schürfeld, Klaus Meyer-Wegener, Jürgen Teich, Christopher Dennl, Andreas Becher, and Florian Bauer. 2016. FPGA-Based Dynamically Reconfigurable SQL Query Processing. *ACM Transactions on Reconfigurable Technology and Systems* 9 (2016), 25:1–25:24. https://doi.org/10.1145/2845087