Datenbank Spektrum
https://doi.org/10.1007/s13222-018-0294-9

SCHWERPUNKTBEITRAG

Integration of FPGAs in Database Management Systems: Challenges and Opportunities

Andreas Becher[1] · Lekshmi B.G.[1] · David Broneske[2] · Tobias Drewes[2] · Bala Gurumurthy[2] · Klaus Meyer-Wegener[1] · Thilo Pionteck[2] · Gunter Saake[2] · Jürgen Teich[1] · Stefan Wildermann[1]

Received: 1 June 2018 / Accepted: 14 August 2018

Abstract
In the presence of exponential growth of the data produced every day in volume, velocity, and variety, online analytical processing (OLAP) is becoming increasingly challenging. FPGAs offer hardware reconfiguration to enable query-specific pipelined and parallel data processing with the potential of maximizing throughput, speedup as well as energy and resource efficiency. However, dynamically configuring hardware accelerators to match a given OLAP query is a complex task. Furthermore, resource limitations restrict the coverage of OLAP operators. As a consequence, query optimization through partitioning the processing onto components of heterogeneous hardware/software systems seems a promising direction. While there exists work on operator placement for heterogeneous systems, it mainly targets systems combining multi-core CPUs with GPUs. However, an inclusion of FPGAs, which uniquely offer efficient and high-throughput pipelined processing at the expense of potential reconfiguration overheads, is still an open problem. We postulate that this challenge can only be met in a scalable fashion when providing a cooperative optimization between global and FPGA-specific optimizers. We demonstrate how this is addressed in two current research projects on FPGA-based query processing.

Keywords Database query processing · Hardware acceleration · Query optimization · FPGA · OLAP

1 Introduction

With the increasing volume, velocity, and variety of nowadays data, the available parallelism and multiplicity of emerging computer systems has to be exploited efficiently in order to process these data. Consequently, we observe an increasing research focus on accelerating database query processing with multi-core CPUs and attached co-processors. In addition to GPUs, FPGAs are deployed more and more for data-processing contexts. Prominent examples are Microsoft's Catapult [38], Baidu's FPGA-based data analysis [17], and cloud services, e. g. the Amazon clouds. While drastic performance improvements can be expected when using heterogeneous architectures, they are not guaranteed per se.

FPGAs consist of a regular array of programmable logic blocks that can be dynamically configured to implement complex hardware modules. Such reconfigurable hardware

This work has been supported by the German Research Foundation (DFG) as part of the Priority Programm 2037 (Grant No.: ME 943/9, PI447/9, SA 465/51, TE163/21, WI4415/1).

✉ Andreas Becher
andreas.becher@fau.de

Lekshmi B.G.
lekshmi.bg.nair@fau.de

David Broneske
david.broneske@ovgu.de

Tobias Drewes
tobias.drewes@ovgu.de

Bala Gurumurthy
bala.gurumurthy@ovgu.de

Klaus Meyer-Wegener
klaus.meyer-wegener@fau.de

Thilo Pionteck
thilo.pionteck@ovgu.de

Gunter Saake
gunter.saake@ovgu.de

Jürgen Teich
juergen.teich@fau.de

Stefan Wildermann
stefan.wildermann@fau.de

[1] Friedrich-Alexander-Universität Erlangen Nürnberg, Nuremberg, Germany

[2] Otto-von-Guericke-University, Magdeburg, Germany

Published online: 10 September 2018

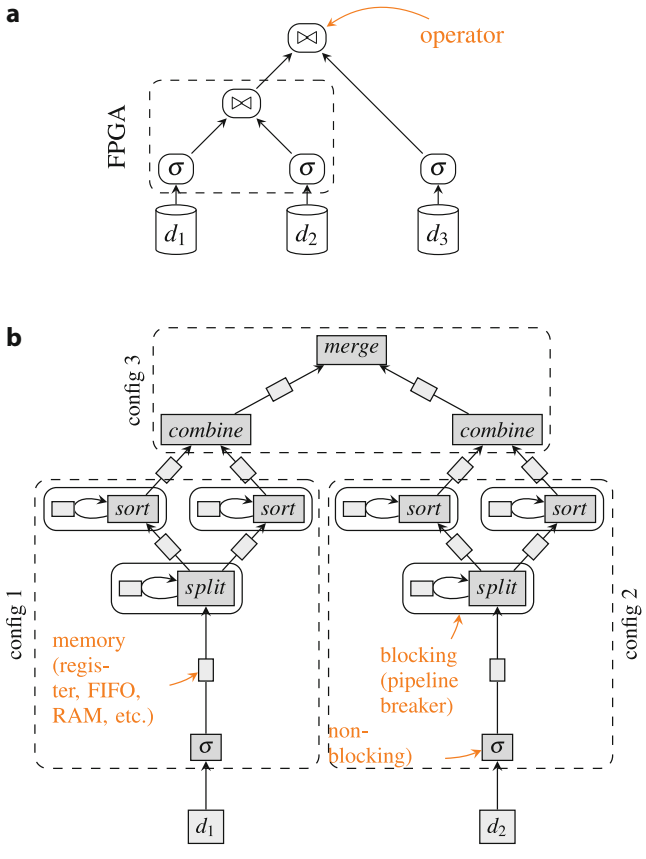

Fig. 1 Example of a **(a)** QEP with a sub-query selected for FPGA acceleration (*dashed lines*) and a **(b)** pipelined and parallelized hardware architecture that directly implements the dataflow of the sub-query. **(a)** Query Evaluation Plan (QEP), **(b)** Dedicated hardware pipeline for QEP

has the ability to be tailored to a specific task. FPGAs show their full strength when parallelism and pipelining can be jointly exploited through provision of high-throughput hardware pipelines, i. e., for processing streams of data. Here, the high energy and resource efficiency compared to CPUs and GPUs stems from the specificity of the hardware circuit and the locality of deeply pipelined processing components. These opportunities are compensated by a factor of 10 lower achievable clock rates than modern CPUs and GPUs. Today, FPGAs are increasingly used for different data-analytics tasks, including data-stream processing, machine learning, extract-transfer load and other data transformation processes.

This article discusses the benefits and challenges of using FPGAs for *online analytical processing (OLAP)*. OLAP queries can be represented by a *query evaluation plan (QEP)* that represents the operators and the dataflow of the underlying computations (illustrated in Fig. 1a). FPGAs provide hardware reconfigurability combining the advantages of hardware efficiency with programmability at the circuit-level, thus allowing to directly implement the dataflow according to the QEP, as illustrated in Fig. 1. All non-blocking operators can process the dataflow in a pipelined fashion at the speed of the I/O interface. In some cases, blocking operators like join and sort operations (pipeline breakers) can be evaluated by highly parallelized and specialized hardware implementations, as exemplified for a parallel sort-merge join implementation in Fig. 1b. In summary, the benefits for OLAP processing using FPGAs are:

- Pipelined processing of non-blocking operators at I/O rate.
- Energy efficiency and reduction of power consumption and/or ...
- ... speedup through parallel and specialized hardware implementations of OLAP operators.
- Resource efficiency by taking workload from processors and providing query-specific hardware acceleration.

However, in principle, the queries submitted to a database system may vary considerably and can be arbitrarily complex. Unlike sequential processing on CPUs, a direct hardware implementation may not be possible due to the limited amount of programmable logic blocks on FPGAs. Consequently, we believe that the benefits of FPGAs in DBMSs can only be utilized by carefully exploiting the synergy among three major concepts:

- Exploitation of scalable and heterogeneous hardware/software target architectures;
- Query partitioning and optimization;
- Dynamic hardware reconfiguration.

In this article, we survey how far related work is tackling these issues in Sect. 2. Sect. 3 summarizes open challenges of integrating FPGAs into heterogeneous DBMSs. Sect. 4 gives an overview of our approaches to cope with these challenges, before we give a conclusion.

2 Related Work

2.1 FPGAs for Query Processing

2.1.1 Designs for Accelerating Specific Operators

In recent years, many investigations have evaluated and quantified the benefits of using FPGAs to accelerate database operators. FPGA acceleration of *sorting*, which is an integral part of many database operations, has gained much attention, see e. g., [11, 32, 44]. Another focus of research has been on the hardware implementation of *join* operators. Similar to most other database operators, a multitude of algorithms for performing a join exist (e. g., hash join, sort-merge join, etc.). Halstead et al. [13] propose an

FPGA accelerator to perform hash joins. They are able to achieve an about 11 times higher throughput on an FPGA compared to a commercial software implementation executed on a multi-core CPU. Casper and Olukotun [11] propose an FPGA implementation of a sort-merge join, which achieves a throughput of 6.45 GB/s. In comparison, an implementation on a multi-core CPU achieves a throughput of 1 GB/s according to [24], and [19] reported a throughput of 4.6 GB/s for a GPU implementation. Most notably, these results show two major potentials of using FPGAs for accelerating database operations: First, it is often possible to achieve I/O-rate processing, leading to a higher overall throughput. Second, while the reported results of CPUs and GPUs are obtained with clock frequencies of over 3.2 GHz and 1.5 GHz, resp., FPGA implementations are often only clocked at $\sim$ 200 MHz. Hence, a higher energy efficiency can be accomplished with FPGAs [2, 29].

Another computationally intensive operator in database query processing is the *selection* of records based on regular expressions. Here, software solutions for both GPUs and multi-core CPUs quickly become computation-bound with increasing complexity of the regular expressions. There is a large body of work that uses regular-expression matching on FPGAs, based on the work of Sidhu and Prasanna [41]. On the downside, they evaluate only a single or a few fixed regular expressions, whereas the database users compose regular expressions at run-time. To address this issue, István et al. [18] and Becher et al. [5] propose run-time parametrizable regex operators.

Today's DBMSs use a multiplicity of algorithms to implement each operator: Which algorithm suits a given query best in terms of execution time often depends on the selectivity and the amount of data to be processed [39, 40]. Therefore, when a query is received, the optimizer selects the best algorithm to achieve the goal of either execution speed or throughput. However, the static FPGA designs for database operators discussed above all suffer from the drawback that they do not allow for run-time algorithm selection to support instantaneous query optimizations. On the other hand, FPGAs allow for *(partial) dynamic reconfiguration* of logic blocks and interconnect, thus enabling the exchange of hardware modules at run-time, lasting less than a millisecond down to some tenths of a millisecond [3].

Ueda et al. [46] present an FPGA-based join accelerator that can switch between hash join and merge join by means of reconfiguration, allowing more resources to be used for each accelerator than in a combined static design. Instead of holding both implementations on the FPGA simultaneously, it is possible to change between both implementations by reconfiguring the FPGA at run-time. A query optimizer should then be able to select the most appropriate implementation based on a cost model. Koch and Torresen [26] propose to make use of partial reconfiguration for sort operations to adjust the resource allocation of a FIFO-based merge sorter and a tree-based merge sorter depending on the problem size.

Recent work [1] on accelerating joins on compute clusters with multi-core nodes has shown that parallel join methods can scale (almost) linearly with the number of cores. In this context, FPGAs may not only be used to partition the data for the parallel instances, as, e. g., described by Kara et al. [20]. However, comparable scalability may also be expected when parallelizing join operators on single or even clustered FPGA nodes.

2.1.2 Static Designs for Accelerating Queries

All the cases given above investigate the acceleration of a single database operator. Queries without the accelerated operator in their QEP cannot benefit from FPGA acceleration. Furthermore, if an operator not available on the FPGA turns out to be the bottleneck of the query, the other accelerated operators do not give a speed benefit. Therefore, some work considers operator pipelines for accelerating whole queries instead of just a single operator. Glacier [30, 31], for example, is a query-to-hardware compiler. For a given query, it automatically instantiates the required operators from a component library and generates a data-stream-processing accelerator, which is tailored to the given query. It is not intended for acceleration in systems with frequently changing queries.

Sukhwani et al. [45] propose a hardware/software co-design including an FPGA accelerator that consists of a feed-forward pipeline of hardware kernels for selection, projection, and sorting. It can be used for the acceleration of arbitrary queries via run-time parametrization in a tuple-at-a-time processing fashion, and pre-processed data is forwarded to the CPU-based host system for post-processing. This approach demonstrates that it is reasonable to couple FPGA-based hardware acceleration with software performing the remaining operations not supported by the FPGA.

Sidler et al. [42] propose a system architecture consisting of a CPU extended by an FPGA. The FPGA has accelerators for the Skyline operator, stochastic gradient descent, and regular expression matching. The system runs with the in-memory DBMS MonetDB and accelerators are integrated into the DBMS. Thus, hardware operators are just additional operators for the DBMS and can be selected by the query optimizer. Here, it is also possible to schedule multiple queries on the same hardware operators [34].

Baidu [17] also implements specific accelerators for different SQL core operators on the FPGA: filter, sort, aggregate, join, and group by. Baidu's query optimizer can select hardware accelerators for calculating parts of the query, and the processing is then carried out on the accelerator.

2.1.3 Reconfigurable Designs for Accelerating Queries

A drawback of the approaches mentioned in the previous subsection is the *static* hardware-accelerator architecture: It is not possible to adapt it to queries. Rather, the QEP must be adapted to match the hardware. For example, queries with many predicates requiring more hardware units for predicate evaluation than available in the accelerator cannot be mapped directly to the FPGA. Instead, the optimizer decomposes them into sub-operations, each fitting to the accelerator, and processes them sequentially. Operators not being part of the accelerator must be processed in software. Thus queries with complex operations, such as hash joins and regular-expression matching, cannot benefit from FPGA acceleration.

In this context, dynamic reconfiguration can provide a viable means to adapt the hardware to the query by exchanging acceleration modules at run-time. This provides query-specific acceleration optimized for the respective use case, while at the same time it achieves a more efficient utilization of the limited FPGA resources.

Wang et al. [48] investigate this potential of run-time reconfiguration for query processing. They show that even for the same query it makes sense to provide multiple accelerators and reconfigure between them while processing the query. Again, each accelerator can utilize the FPGA resources exclusively to increase parallelism and thus decrease execution time. However, due to the overhead of reconfiguration, only large datasets can benefit from it, since the reduction in execution time is larger than the reconfiguration time. Wang et al. further propose a methodology for automatically generating multiple QEPs for a given query. Each consists of a set of hardware accelerators and a schedule of how to switch between them. At run-time, a plan is chosen from the set of generated evaluation plans, so that it has the shortest execution time according to a cost model. While this validates the benefit of adapting the hardware to the query by means of reconfiguration, it has the major drawback that each accelerator is tailored to a specific query. Synthesizing an accelerator can take from minutes to hours. Consequently, it is infeasible to assemble accelerators for ad-hoc queries at run-time. Therefore, this approach can only be applied for queries that are already known at design time. Furthermore, the system only supports full reconfiguration of the FPGA. This implies, first, a larger reconfiguration overhead, as the time increases linearly with the size of the FPGA area to be reconfigured (up to seconds according to [48]). Second, as the data contained in FPGA memory typically is lost during full reconfiguration, an additional overhead occurs for transferring the local memory contents back to the CPU-based host system.

Ziener et al. [52] present a methodology for *on-the-fly data-path generation* of a query-stream pipeline. Query plans can be accelerated by composing and placing pre-synthesized modules (e. g., aggregation and restriction operators) on the FPGA by means of partial reconfiguration. Furthermore, this has enabled the use of column-oriented operators, which is also used by Kara et al. [20].

2.2 Heterogeneous DBMS

The systems described above feature FPGA accelerators complementing the host CPUs. Due to different computation models, they are considered to be heterogeneous. From an architectural view, the structure is similar to systems that incorporate GPUs as co-processors in addition to CPUs [15, 16, 51, 7]. Such heterogeneous systems pose specific requirements on algorithmic design, query optimization, and cost models for DBMSs. In the following we will describe these requirements by means of the state of the art for such heterogeneous systems.

2.2.1 Optimizing Operators for Heterogeneous Hardware

Tuning operators to a single co-processor has already attracted much attention (c. f., for GPU [15, 19, 25, 43], for Xeon Phi [36, 37], and for FPGA [29, 32]). However, a holistic approach should be able to work on arbitrary co-processors. There are two common implementation approaches. The straight-forward one is the *hardware-oblivious* approach [16], where operators are implemented against a parallel-programming library, e. g., OpenCL. This approach allows for portability of operator code, but not of performance [40]. Hence, a *hardware-sensitive* approach should be used, where the code is optimized for each co-processor separately. DBMSs following a hardware-sensitive approach use a primitive-based execution or query-compilation strategies.

Primitive-Based Execution: Data-parallel primitives have been suggested by He et al. as a means to implement database operators in a CPU/GPU system [15]. The idea is to split an operator into smaller functions. These functions can then easily be tuned for any co-processor, allowing optimizations orthogonal to the original operator. Such optimization possibilities for heterogeneous co-processors have already been shown by Pirk et al. for vector algebra on heterogeneous systems [35].

Query Compilation for Heterogeneous Hardware: Query compilation is an alternative to the operator-based execution of queries [33]. Here, a query is split into execution units fusing operator code. The dataflow allows data to be pipelined up to a pipeline breaker (a blocking operation like join or aggregation). To enable query compilation for heterogeneous co-processors, Breß et al. present Hawk [10] and Funke et al. present HORSEQC [12] as hardware-adaptive query compilers. Besides other optimizations, Hawk

employs code optimizations to tune operator code for the used co-processor and HORSEQC fuses operators.

2.2.2 Query Optimization in Heterogeneous DBMSs

Query optimization is a crucial part in DBMSs due to SQL's declarative nature. Traditional query optimization covers plan optimization and algorithm selection for a single processor. In a heterogeneous system, however, a multitude of new optimization possibilities arise (e. g., co-processor selection). In general, DBMSs either employ global query optimization assigning operators at optimization time or dynamic optimization at run-time. The latter can be achieved by *query chopping* [9], where operators are assigned to co-processors right when all their inputs are ready (i. e., all child operators are finished). While query chopping takes into account the load of the co-processors, it is a greedy exploitation, which may miss the global optimum that optimizers could find across processors.

Global optimization needs to understand the capabilities of the co-processors. Karnagal et al. have conducted varies studies in this area [21, 23]. According to them, query processing on heterogeneous co-processors is based on characteristics like data size, computational complexity of operators, co-processor properties, and execution times of physical operators. They propose a Heterogeneity-aware physical Operator Placement (HOP) by assigning the physical operators to co-processors at run-time using a cost model, which calculates the execution cost based on operator behavior, hardware characteristics, and run-time information. HOP can find the best operator placement using a cost model, for any input-parameter values. Breß et al. also address the operator-placement problem in [6] using the optimizer library HyPE, which follows a greedy and cost-based approach.

However, the operator-placement strategy faces some challenges such as an increasing number of operators, which leads to a search-space explosion, and an uncertainty in cardinality estimation for intermediate results. Karnagal et al. tackle this issue with a greedy technique, the so-called *strong placement*, which assigns an operator to one co-processor and places the remaining operators randomly by *majority voting* [22]. Since both technologies rely on a greedy strategy, the optimal solution may not be produced.

An adaptive placement approach was later proposed in [23] to tackle the unexpected behavior of an operator, caused by a complex query structure, large input data, and the location of intermediate results. In this technique, the optimization and execution of a part of the query, called execution island, is implemented. Since the optimization is limited to a part of the query, cardinalities of intermediate results and execution time can be estimated with high precision. Although this regional approach comes close to a global optimization, a tuple-at-a-time model would not benefit from it [23].

Considering *energy-efficient query processing* along with performance constraints is also a challenge in query optimization. In this case, an optimizer must find QEPs that meet the performance goals while consuming as little energy as possible w.r.t. available (co-)processors. Lang et al. have designed a framework for query optimization that considers energy efficiency along with performance constraints. It works well for simple queries, but not for complex and concurrent queries [27]. Later, Zichen Xu et al. introduced a Power-Energy-Time (PET) framework for the PostgreSQL kernel [49]. They point out that the energy-efficient QEPs may not have the shortest processing time. Ungethüm et al. proposed a hardware/software-co-design approach on the *Tomahawk* architecture [47], where they utilize energy-efficient customizable instruction-set processors. Becher et al. [2] propose an FPGA-based hardware software co-design for energy-efficient hash join operation.

Global query optimization in heterogeneous DBMSs greatly depends on the dynamically changing volume of data, the execution time of the query, the workload characteristics, and the unpredictable cardinality of data. The research addressing these characteristics in a fast as well as cost- and energy-efficient way is still at its peak.

2.2.3 Cost Models for Operators on Heterogeneous Hardware

In the presence of heterogeneous co-processors, costs for operators on different co-processors have to be taken into account as well as transfer times between processors. There are two ways to determine the costs of an operator or query plan: using an algebraic or a learning-based cost model.

Algebraic Cost Models: An algebraic cost model is a parameterized function describing the cost for an operator according to the access pattern and the computation an operator causes. The specific parameters are usually determined in a calibration phase at start-up. Algebraic cost models give the most accurate cost estimations for an operator.

Cost models have a long history in DBMSs. For in-memory systems, Manegold et al. proposed a cost model taking cache hierarchies as the new bottleneck into account [28]. For GPUs, there is already a plethora of cost models for different operators [15, 50]. However, a holistic and comprehensive cost model is hard to find for arbitrary co-processor and operator combinations.

Learning-Based Cost Models: Learning-based cost models use machine-learning techniques to estimate the performance of an operator on a co-processor. For heterogeneous co-processors, Breß et al. introduced HyPE [6, 8], and Karnagel presented HOP [21] as learning-based cost estimators.

However, both estimators currently do not take concurrent execution into account.

2.3 Summary

Due to resource limitations, there is no FPGA accelerator that supports all queries in the best way. For flexible and generic coverage of queries, mapping query operators to the available co-processors and to the software of a heterogeneous hardware/software system is necessary. While there is work on operator placement for heterogeneous systems, e. g., by Karnagel et al. [21] and by Breß [7], they mainly target systems combining multi-core CPUs with GPUs. How to extend these systems to FPGAs, which come with very specific overheads when applying reconfiguration between operators, is still an open issue.

3 Challenges

The study of related work clearly shows that an integration of FPGAs into a DBMS is challenging, ranging from general challenges of heterogeneous systems (*query optimization* and *partitioning*) to FPGA-specific challenges (*dynamic reconfiguration*).

3.1 Exploitation of Heterogeneous Hardware/ Software Architectures

Since different hardware concepts (e. g., CPUs, GPUs, and FPGAs) have their individual strengths and weaknesses in data processing, a clever partitioning of data and work has to be done in a heterogeneous system. Although first attempts have been made to schedule and distribute data and operations (c. f., CoGaDB, HyPE), they consider neither the opportunity of hardware reconfiguration of the FPGA nor its stream-oriented processing style. Furthermore, caching strategies will probably not apply to an FPGA with its limited amount of dedicated memory.

3.2 Query Partitioning and Optimization

For a query optimizer in a heterogeneous system, it is challenging and a so far unsolved problem to to find (1) an optimal partitioning of a query on the different heterogeneous co-processors and (2) determine the configuration of the reconfigurable hardware itself.

The first challenge (global view) creates a huge optimization space when considering multiple different co-processors, multiple implementation variants for an operator, different granularities for executing an operator, concurrent resource utilization on a co-processor, and even multiple objectives.

The second challenge (local view) implies a maintenance task that composes and schedules a reconfiguration of the FPGA when a specific function is efficiently supported or frequently used. This involves (a) selecting appropriate co-processors for the given query while considering reconfiguration overhead, (b) mapping query operations to such accelerators, and (c) reconfiguring the hardware to generate data paths for the accelerators that enable stream processing as close as possible to I/O-rate.

3.3 Dynamic Hardware Reconfiguration

The challenge of dynamic reconfiguration is to efficiently adapt FPGAs to a query and to time-multiplex query processing under scarce resources. Especially ensuring that reconfiguration pays off is a huge challenge, since FPGA reconfiguration itself may take a time ranging from 1 ms (partial reconfiguration) to the order of 1 s (full reconfiguration). Thus, overall savings must be higher than the reconfiguration costs. So, either the amount of data processed in one query must be large enough or the query needs to occur frequently enough. Another challenge is that queries use operators for data types with different or even arbitrary lengths (e. g., strings). In order to restrict the space of hardware accelerator configurations to be synthesized and stored, a big challenge here is to identify a representative set of often occurring, but in data types highly parameterizable kernel modules that are loaded and configured at runtime. Finding these modules with enough soft parameters left to avoid a dynamic hardware reconfiguration or module re-synthesis is a central focus and challenge of our ongoing research.

Through the remainder of this paper, we will present our approaches to meet these challenges.

4 Proposed Solutions

A substantial part of the challenges refers to the query optimization for heterogeneous systems that include an FPGA. In the following two research approaches, ADAMANT at the Otto-von-Guericke-Universität Magdeburg (OvGU) and ReProVide at the Friedrich-Alexander Universität Erlangen-Nürnberg (FAU), are presented, where query partitioning and optimization plays a major role. On an abstract level, the common ground of both methodologies is to get a grip on the huge solution space with a distribution of optimization tasks among *cooperative optimizers*.

4.1 Cooperative Optimizers

The idea of cooperative optimizers is to split the optimization into a *global* and a *local optimization* according to the

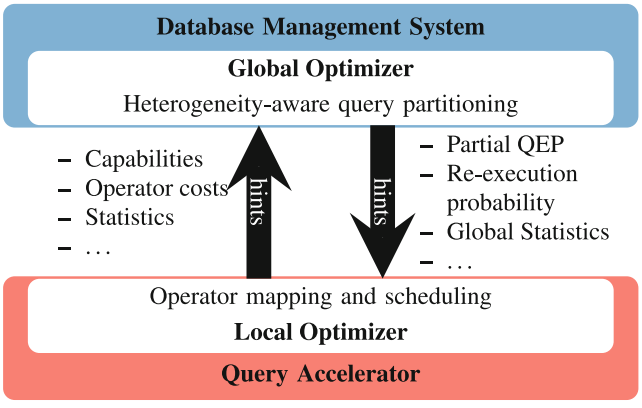

Fig. 2 Interplay of global and local optimizer

global and local views described in Sect. 3.2. Global optimization applies strategies at the level of the QEP, with the major goal of partitioning the QEP into sub-trees and assigning them to the heterogeneous processors. The local optimizer is responsible for finding the best implementation of a sub-tree on the specific target hardware. Fig. 1a already showed an example where (a) parts of the QEP are assigned to an FPGA (dashed line). The (b) target-specific optimization of the sub-tree involves selecting operator variants to be executed (e. g. sort-merge join) and the parallelism granularity (e. g. parallel sorting), as well as generating the hardware configurations of the FPGA and determining the schedule for reconfiguring between them.

Due to the huge solution space, distributing certain optimization decisions to the autonomous co-processors is reasonable to parallelize query optimization and lower the degree of detail and thus the load on the requesting CPU. The global optimizer decides based on the current load and also on co-processor properties according to cost models that are frequently refreshed at run-time with the help of information from the local optimizers. The capabilities of the heterogeneous co-processors such as device power, buffer pool, cost of operator evaluation, etc. play a key role in the degree of accuracy of query-cost estimation. They are only known to the resp. local optimizers and thus must be communicated to the global optimizer. Partitioning a query between heterogeneous processors as well as further operator placement and mapping of sub-queries are only effective after a communication between local and global optimizer. Our proposed approach of cooperative optimization involves exchanging *hints* between global and local optimizer. This is illustrated in Fig. 2. With the help of these hints, the global optimizer can make a decision on how much optimization is required per plan and can maximize the performance benefits. We expect this bidirectional hint interface to enable scalable optimization in both directions, i. e., globally and locally, which helps to subsequently reduce optimization overhead.

In the remainder of this section, we introduce two proposed solutions and demonstrate how to incorporate the interplay of local and global optimizers.

4.2 ADAMANT: Adaptive Data Management in Evolving Heterogeneous Hardware/Software Systems

ADAMANT integrates a DBMS with extensible, adaptable support for heterogeneous hardware. To support a multitude of devices, we employ the concept of Plug 'n' Play by abstracting their common functionalities. The main objectives of our project in order to support this highly volatile environment are as follows:

Abstraction of devices: Seamless integration of heterogeneous devices with different interfaces (e. g., PCIe, QPI, CAPI) on a one-by-one-basis is a difficult, time-consuming, and redundant task. Thus, it is necessary to find a useful abstraction level that considers the fundamental differences between accelerators.

Multi-layered optimization: The presence of multiple processing devices for query execution creates a new dimension of parallelism – cross-device parallelism. Additionally, FPGAs provide spatial parallelism, but run-time reconfiguration limits temporal availability of operators. Along with the dimensions of data and functional parallelism, these new dimensions require an improved optimization process in order to overcome the difficulties resulting from the explosion of search space.

Exploiting device features: Exploiting the different features available in heterogeneous co-processors results in more efficient implementations. Device-specific functions are improved by fine-tuning device-dependent implementation parameters such as loop-unrolling depth, vector width, based on feedback from the environment. This goes beyond possible optimizations of OpenCL by also allowing for data-sensitive performance portability w.r.t. workload selectivity, for example [40]. Here, FPGAs allow for operator implementations using a wider set of possible architectures than fixed (co-)processors.

4.2.1 Architecture

Since the performance of database operators, especially domain-specific ones, depends on optimized implementations, a direct implementation can easily become inflexible, which hinders adoption of new processing devices and custom operators. In order to decouple domain-specific operators and hardware processing units, we base our system on a lay-

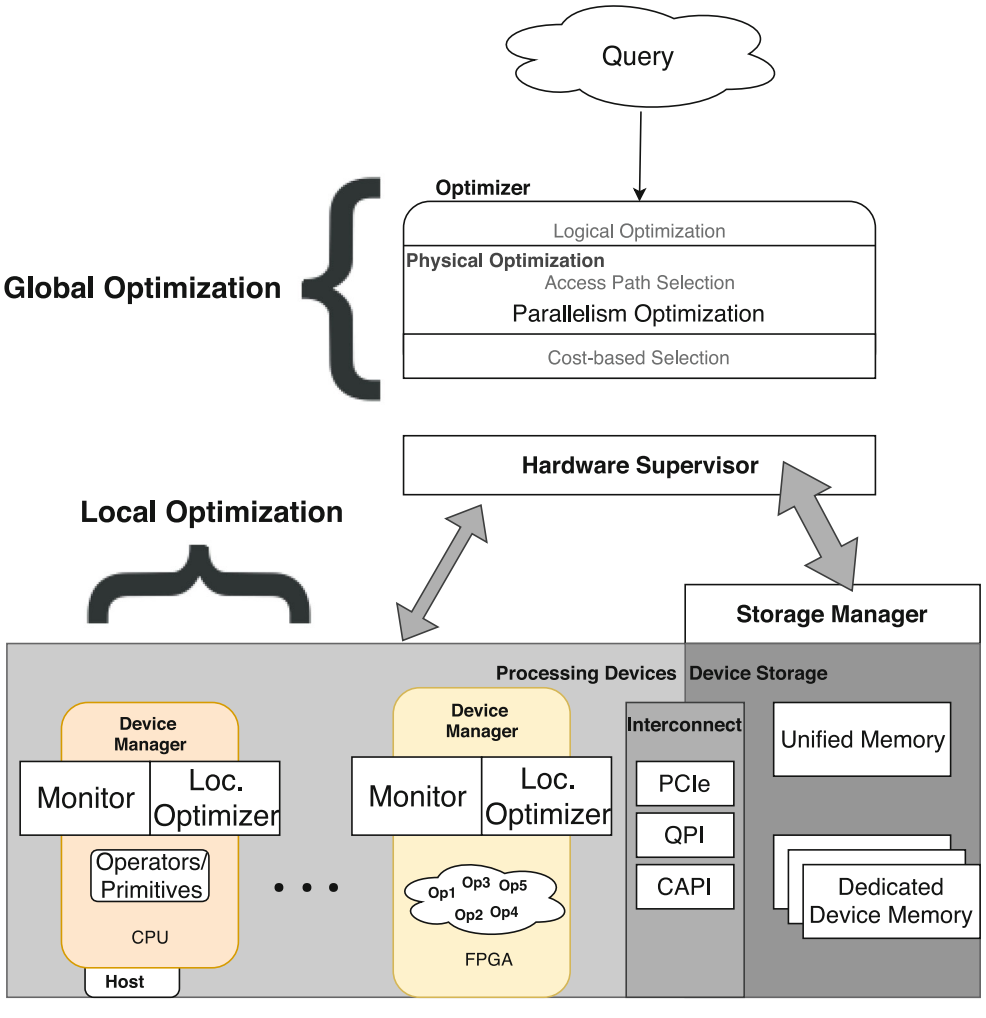

Fig. 3 ADAMANT: Adaptive DBMS architecture

ered architecture. This enables an efficient incorporation of FPGAs with their vastly different intrinsic features and requirements.

The architecture of the ADAMANT system is shown in Fig. 3. The global optimizer shown at the top performs the optimization steps found in classical DBMSs. It also delegates parts of the optimization process to local optimizers that are assigned to the co-processors. Thus, a layered approach to optimization enables efficient support for heterogeneous hardware. In addition to the local device-specific optimizer, the device manager of each co-processor, such as GPUs or FPGAs as well as the host CPU, contains run-time monitoring facilities to support the higher-level optimization components. At runtime, the device manager aims to improve operator performance by going beyond static compiler optimizations for the underlying device using techniques such as micro-adaptivity [39].

The storage manager on the right side of Fig. 3 is responsible for requesting data transfers between dedicated device memories and main memory. It also passes estimates to higher optimization layers, for example using cost models similar to [14]. Finally, edge cases such as devices that share a unified memory view are handled in this component. The data gathered by all managers is aggregated by a single global hardware supervisor, providing a unified abstract view of the complete system to the global optimizer.

Supporting the software part of the ADAMANT architecture with FPGAs requires more than custom implementations of the usual database operators. Both the architecture of the base design and the hardware interfaces between static infrastructure and dynamically configured operators have to be engineered for high throughput and flexibility. This is also part of the ADAMANT project.

4.2.2 Query Optimization

As mentioned above, a global optimizer determines the best execution plan for a given domain-specific query. As multiple devices can be used for processing that query, we add a new layer: parallelism optimization, to exploit concurrent processing on available devices. However, adding this new level of parallelism to the existing solution space makes it

even harder to find the optimal path. Hence, an efficient solution for traversing this multi-dimensional parallelism space in a reasonable time is required. To overcome this, we follow three basic concepts:

Information propagation: Since device-related characteristics are a key factor for optimizing a query execution plan, information about the devices is propagated to the global optimizer. The characteristics of these devices, such as memory organization and memory footprint, are considered for optimization.

Delegation: To further reduce the search space, we split the optimization into smaller steps and delegate them to the local optimizer. As it is hard to keep track of all device-related characteristics and tune operators based on them, especially the device-dependent optimization steps are delegated to the local optimizer available on the device.

Parallelism optimization: In order to fully exploit the data and functional parallelism, we consider different levels of granularity for operators (cf. Fig. 1b) for each of the underlying devices. The granularity and device-specific variant of an operator affect the overall performance of the system. Hence, we have to carefully consider these trade-offs between the devices, which requires the optimizers to collaborate and decide on the granularity level of the operators.

Overall, the QEP given by the global optimizer is further optimized to fit the underlying heterogeneous system. Since these hardware characteristics are considered for an optimized evaluation plan, a multi-level optimization is performed by propagating information about the devices to the global optimizer and pushing the device-specific optimizations to the local. Finally, each device has its own optimal implementation points for its supported primitives, leading to collaboration among available devices for path selection.

4.3 ReProVide: Query Optimization and Near-Data Processing on Reconfigurable SoCs for Big Data Analysis

ReProVide (standing for Reconfigurable Data ProVision) follows the idea of near-data processing. ReProVide units implemented on programmable SoC as illustrated in Fig. 4 serve as data providers accessible via network and equipped with a reconfigurable SoC. The data is stored in non-volatile memory such as SSDs or volatile DDR-SDRAM, which is directly attached to and managed by the platform. A main assumption and advantage of ReProvide is that it abstracts from the actual table and storage schema. Rather, the DBMS conveys which data it requires in which format. ReProVide units come with processing capabilities (in form

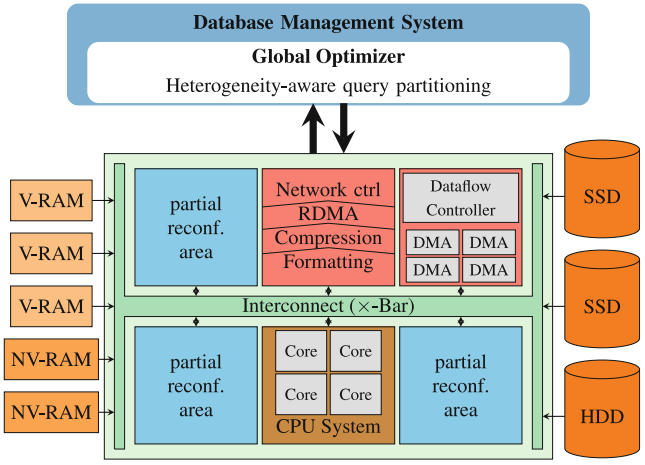

Fig. 4 A schematic overview of a ReProVide near-data query processing system implemented on a programmable SoC with a multi-core CPU and multiple partially reconfigurable FPGA regions

of hardware accelerators and closely-coupled multi-core processors) that can be used for data processing. Moreover, they contain special dataflow-control circuits (so-called ReOrder units [4]) to assemble tables and data as requested by the DBMS at I/O-rate (*schema-on-read*). This design enables ReProVide units to locally optimize the storage location of the data with regards to availability, access latency, power consumption, and the access patterns of the available hardware accelerators.

These schema-on-read capabilities provided by ReOrder units [4] bring multiple benefits:

(a) Flexibility to process data stored in row-oriented as well as column-oriented format.
(b) The data can be modified on the fly so that only columns which are actually needed are sent over the network to the requesting DBMS host, thus relaxing the I/O bottleneck.
(c) Additional columns can be inserted, e.g., while the data is fetched by the DBMS.

One example would be the on-the-fly calculation of intermediate results (e. g., *price* × *discount*) which is then inserted into the tuple stream relieving not only the DBMS of its calculation, but maybe also the network by transmitting fewer attributes. Moreover, the host can request an operator-optimized schema. Such a schema could be a hybrid between column- and row-store to, e. g., separate join keys and join data to allow for faster sorting/joining of the keys.

In addition, the processing capabilities of ReProVide itself can be exploited to reduce the number of rows sent over the network by pushing restrictions to the data.

4.3.1 Architecture

The ReProVide platform itself constitutes a heterogeneous hardware/software system, as shown in Fig. 4. While the CPU subsystem can process the full variety of operators and types, its performance may be limited. On the other hand, operators implemented in hardware can utilize parallel and pipelined implementations to achieve up to I/O-rate processing. ReProVide units therefore include partially reconfigurable areas (highlighted in blue in Fig. 4), into which query-/operator-specific accelerators can be dynamically loaded. The dynamic synthesis of a new operator (query compilation) can take from minutes up to multiple hours. We therefore make use of a library of pre-synthesized reconfigurable hardware accelerators.

As storage space for this library and thus the amount of accelerators is limited, these accelerators will most often implement multiple operators in order to gain a proper coverage of operators and queries. The challenge is to find a good trade-off between a high coverage of common operators, the resource limitations imposed by the partial areas, and the memory requirement for storing the library. Larger partial areas mean less resource restriction for implementing an accelerator, but increased reconfiguration time (start-up cost) and fewer concurrently available partial areas in total (reduced parallelism). We will therefore provide a design flow to automatically generate partially reconfigurable accelerators, i.e., pipelines of operators based on statistical data collected by the ReProVide unit itself (e. g., which operators in which combination and with which frequency were requested). By making use of model-based *multi-objective optimization*, it is thus possible to automatically obtain an optimized set of accelerator designs, which poses a optimized solution for frequently occuring operators and queries.

To improve resource efficiency, unused parts of loaded accelerators can be used to generate meta-data on data streamed through them, e. g., statistics and indices—if re-execution is likely. While indices can help a local optimizer, statistics are important for the global optimizer. Both optimizers need to work together closely to leverage the full potential of such a system.

4.3.2 Query Optimization

The interaction of the DBMS and ReProVide has already been sketched in Fig. 2. Required is the investigation of novel global query-optimization techniques based on concepts known from *multi-databases*. The analysis and preprocessing of a QEP consists of three phases:

In the first phase, the global optimizer decides which operations of a QEP are worthwhile to be assigned to the ReProVide SoC. The local optimizer provides information on its capabilities and statistics of its local data. In order to obtain a fast, energy-efficient, and cost-effective query execution, a novel cost model as well as a novel hierarchical query-optimization technique need to be investigated.

In the second phase, the QEP is passed from the global optimizer to a local optimizer, including some hints. The local optimizer then selects accelerators, maps query operations to accelerators, and schedules the reconfiguration of the accelerators (see Fig. 5). We envision that the local optimizer not only returns the cost for the complete QEP, but also the cost for specific sub-trees. In Fig. 5, each QEP partition is annotated with the expected end-to-end execution time for processing the partition's input data based on cost models and selectivity statistics of the data sources. The goal of this strategy is to reduce the amount of subsequent calls for cost estimations of other QEPs, as it gives the optimizer the possibility of anticipating the impact when reordering the QEP, or assigning only sub-trees of it to

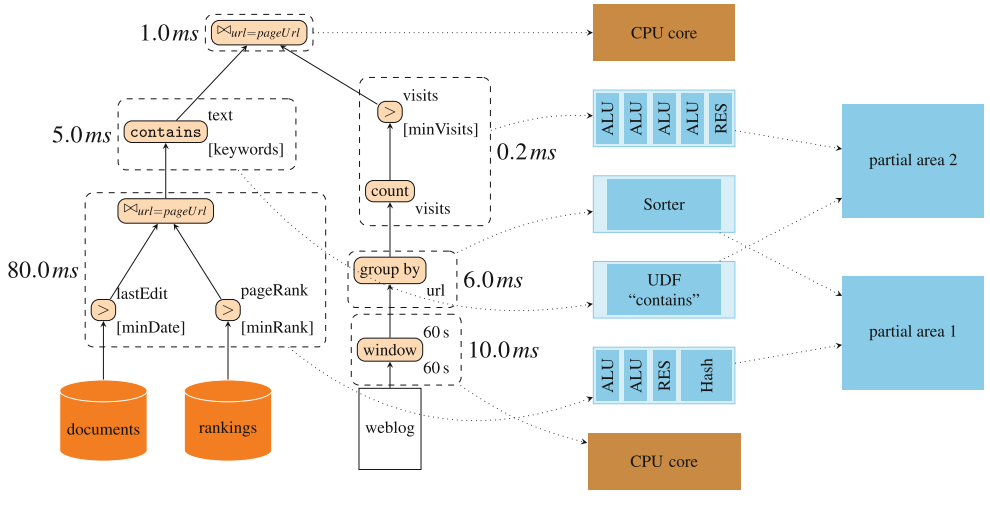

Fig. 5 Example of partitioning a QEP into sub-trees (*dashed boxes*), binding of partitions to accelerators, and scheduling of accelerators on allocated partial areas

the ReProVide platform. Moreover, ReProVide may also answer that it cannot execute some operator at all, for instance, because some expressions are too complex. Along with the QEP, the frequency of QEP execution and an upper bound for latency are also transferred as hints. So, the local optimizer can keep often used accelerations and also set a time budget for running the local optimizer.

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

5 Conclusion

FPGAs have a huge potential in query processing in DBMSs because of their special capabilities. In order to efficiently use them, we address three major challenges. First, efficient partitioning of data and DBMS operations among available processors is necessary to fully exploit all of their capabilities. Second, the availability of a multitude of heterogeneous devices requires improved optimization strategies that take into consideration the device-based characteristics for finding an efficient evaluation plan. Finally, FPGAs have high reconfiguration and design-iteration costs and hence, require concepts to adapt an accelerator circuitry to a given query. Since all of these challenges lie in the domain of query optimization, we propose a cooperative optimization approach. To overcome the issue of larger search space, we split the general optimizer into a global and a local optimizer so that device-related optimization is performed by the local optimizer.

Thus, the major contributions of this paper are: Identifying the challenges of integrating FPGAs into DBMSs, providing a comprehensive survey of the state-of-the-art in FPGA-accelerated DBMS processing, and proposing a two-layered co-operative optimizer design to efficiently traverse the enlarged search space.

References

1. Barthels C, Alonso G, Hoefler T, Schneider T, Müller I (2017) Distributed join algorithms on thousands of cores. Proceedings VLDB Endowment 10(5):517–528
2. Becher A, Ziener D, Meyer-Wegener K, Teich J (2015) A co-design approach for accelerated SQL query processing via FPGA-based data filtering. In: FPT, pp 192–195
3. Becher A, Pirkl J, Herrmann A, Teich J, Wildermann S (2016a) Hybrid energy-aware reconfiguration management on Xilinx Zynq SoCs. In: ReConFig, pp 1–7
4. Becher A, Wildermann S, Mühlenthaler M, Teich J (2016b) Reorder: Runtime datapath generation for high-throughput multi-stream processing. In: ReConFig, pp 1–8
5. Becher A, Wildermann S, Teich J (2018) Optimistic regular expression matching on FPGAs for near-data processing. Proc. 14th Int. Workshop on Data Management on New Hardware, Houston, 11.6.2018, pp 1–4
6. Breß S (2013) Why it is time for a HyPE: a hybrid query processing engine for efficient GPU coprocessing in DBMS. Proceedings VLDB Endowment 6(12):1398–1403
7. Breß S (2014) The design and implementation of CoGaDB: a column-oriented GPU-accelerated DBMS. Datenbank Spektrum 14(3):199–209. https://doi.org/10.1007/s13222-014-0164-z
8. Breß S, Beier F, Rauhe H, Sattler KU, Schallehn E, Saake G (2013) Efficient co-processor utilization in database query processing. Inf Syst 38(8):1084–1096. https://doi.org/10.1016/j.is.2013.05.004
9. Breß S, Funke H, Teubner J (2016) Robust query processing in co-processor-accelerated databases. In: SIGMOD, pp 1891–1906 https://doi.org/10.1145/2882903.2882936
10. Breß S, Köcher B, Funke H, Rabl T, Markl V (2018) Generating custom code for efficient query execution on heterogeneous processors. VLDB J. https://doi.org/10.1007/s00778-018-0512-y
11. Casper J, Olukotun K (2014) Hardware acceleration of database operations. In: FPGA, pp 151–160
12. Funke H, Breß S, Noll S, Markl V, Teubner J (2018) Pipelined query processing in coprocessor environments. In: SIGMOD, pp 1603–1618 https://doi.org/10.1145/3183713.3183734
13. Halstead RJ, Sukhwani B, Min H, Thoennes M, Dube P, Asaad SW, Iyer B (2013) Accelerating join operation for relational databases with FPGAs. In: FCCM, pp 17–20
14. Hampel V, Pionteck T, Maehle E (2012) An approach for performance estimation of hybrid systems with FPGAs and GPUs as co-processors. In: ARCS, pp 160–171
15. He B, Lu M, Yang K, Fang R, Govindaraju NK, Luo Q, Sander PV (2009) Relational query coprocessing on graphics processors. ACM Trans Database Syst 34(4):Art. No 21. https://doi.org/10.1145/1620585.1620588
16. Heimel M, Saecker M, Pirk H, Manegold S, Markl V (2013) Hardware-oblivious parallelism for in-memory column-stores. Proceedings VLDB Endowment 6(9):709–720
17. Hemsoth N (2016) Baidu takes FPGA approach to accelerating SQL at scale. The next platform. https://www.nextplatform.com/2016/08/24/baidu-takes-fpga-approach-accelerating-big-sql/. Accessed 24 Aug 2018
18. István Z, Sidler D, Alonso G (2016) Runtime parameterizable regular expression operators for databases. In: FCCM, pp 204–211
19. Kaldewey T, Lohman GM, Müller R, Volk PB (2012) GPU join processing revisited. In: DaMoN, pp 55–62
20. Kara K, Giceva J, Alonso G (2017) FPGA-based data partitioning. In: SIGMOD, pp 433–445
21. Karnagel T, Habich D, Schlegel B, Lehner W (2014) Heterogeneity-aware operator placement in column-store DBMS. Datenbank Spektrum 14(3):211–221. https://doi.org/10.1007/s13222-014-0167-9
22. Karnagel T, Habich D, Lehner W (2015) Local vs. global optimization: operator placement strategies in heterogeneous environments. In: Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference, Brussels, Belgium, 27 March 2015, pp 48–55
23. Karnagel T, Habich D, Lehner W (2017) Adaptive work placement for query processing on heterogeneous computing resources. Proceedings VLDB Endowment 10(7):733–744
24. Kim C, Sedlar E, Chhugani J, Kaldewey T, Nguyen AD, Blas AD, Lee VW, Satish N, Dubey P (2009) Sort vs. hash revisited: fast join implementation on modern multi-core CPUs. Proceedings VLDB Endowment 2(2):1378–1389
25. Kim C, Chhugani J, Satish N, Sedlar E, Nguyen AD, Kaldewey T, Lee VW, Brandt SA, Dubey P (2010) FAST: fast architecture sensitive tree search on modern CPUs and GPUs. In: SIGMOD, pp 339–350
26. Koch D, Tørresen J (2011) FPGASort: a high performance sorting architecture exploiting run-time reconfiguration on FPGAs for large problem sorting. In: FPGA, pp 45–54

27. Lang W, Kandhan R, Patel JM (2011) Rethinking query processing for energy efficiency: slowing down to win the race. IEEE Data Eng Bull 34(1):12–23
28. Manegold S, Boncz P, Kersten ML (2002) Generic database cost models for hierarchical memory systems. In: Proceedings VLDB Endowment pp 191–202
29. Müller R, Teubner J (2009) FPGA: what's in it for a database? In: SIGMOD, pp 999–1004
30. Müller R, Teubner J, Alonso G (2009) Streams on wires – A query compiler for FPGAs. Proceedings VLDB Endowment 2(1):229–240
31. Müller R, Teubner J, Alonso G (2010) Glacier: a query-to-hardware compiler. In: SIGMOD, pp 1159–1162
32. Müller R, Teubner J, Alonso G (2012) Sorting networks on FPGAs. VLDB J 21(1):1–23. http://dx.doi.org/10.1007/s00778-011-0232-z
33. Neumann T, Leis V (2014) Compiling database queries into machine code. IEEE Data Eng Bull 37(1):3–11
34. Owaida M, Sidler D, Kara K, Alonso G (2017) Centaur: a framework for hybrid CPU-FPGA databases. In: FCCM, pp 211–218
35. Pirk H, Moll O, Zaharia M, Madden S (2016) Voodoo – a vector algebra for portable database performance on modern hardware. Proceedings VLDB Endowment 9(14):1707–1718
36. Pohl C (2017) Exploiting manycore architectures for parallel data stream processing. In: GvDB, pp 66–71 (http://ceur-ws.org/Vol-1858/paper13.pdf)
37. Polychroniou O, Raghavan A, Ross KA (2015) Rethinking SIMD vectorization for in-memory databases. In: SIGMOD, pp 1493–1508
38. Putnam A, Caulfield A, Chung E, Chiou D, Constantinides K, Demme J, Esmaeilzadeh H, Fowers J, Gray J, Haselman M, Hauck S, Heil S, Hormati A, Kim JY, Lanka S, Peterson E, Smith A, Thong J, Xiao PY, Burger D, Larus J, Gopal GP, Pope S (2014) A reconfigurable fabric for accelerating large-scale datacenter services. In: ISCA, pp 13–24
39. Raducanu B, Boncz PA, Zukowski M (2013) Micro adaptivity in vectorwise. In: SIGMOD, pp 1231–1242 https://doi.org/10.1145/2463676.2465292
40. Rosenfeld V, Heimel M, Viebig C, Markl V (2015) The operator variant selection problem on heterogeneous hardware. In: ADMS, pp 1–12
41. Sidhu RPS, Prasanna VK (2001) Fast regular expression matching using FPGAs. In: FCCM, pp 227–238
42. Sidler D, István Z, Owaida M, Alonso G (2017) Accelerating pattern matching queries in hybrid CPU-FPGA architectures. In: SIGMOD, pp 403–415
43. Sitaridi E, Ross K (2013) Optimizing select conditions on GPUs. In: DaMoN, ACM. pp, vol 4, pp 1–4
44. Sukhwani B, Thoennes M, Min H, Dube P, Brezzo B, Asaad SW, Dillenberger D (2013) Large payload streaming database sort and projection on FPGAs. In: SBAC-PAD, pp 25–32
45. Sukhwani B, Thoennes M, Min H, Dube P, Brezzo B, Asaad SW, Dillenberger D (2015) A hardware/software approach for database query acceleration with FPGAs. Int J Parallel Program 43(6):1129–1159
46. Ueda T, Ito M, Ohara M (2015) A dynamically reconfigurable equi-joiner on FPGA. IBM Technical Report RT0969
47. Ungethüm A, Habich D, Karnagel T, Lehner W, Asmussen N, Völp M, Noethen B, Fettweis GP (2015) Query processing on low-energy many-core processors. In: ICDE, pp 155–160
48. Wang Z, Paul J, Cheah HY, He B, Zhang W (2016) Relational query processing on OpenCL-based FPGAs. In: FPL, pp 1–10
49. Xu Z, Tu Y, Wang X (2012) PET: reducing database energy cost via query optimization. Proceedings VLDB Endowment 5(12):1954–1957
50. Yuan Y, Lee R, Zhang X (2013) The yin and yang of processing data warehousing queries on GPU devices. Proceedings VLDB Endowment 6(10):817–828
51. Zhang S, He J, He B, Lu M (2013) OmniDB: Towards portable and efficient query processing on parallel CPU/GPU architectures. Proceedings VLDB Endowment 6(12):1374–1377
52. Ziener D, Bauer F, Becher A, Dennl C, Meyer-Wegener K, Schürfeld U, Teich J, Vogt J, Weber H (2016) FPGA-based dynamically reconfigurable SQL query processing. ACM Trans Reconfigurable Technol Syst 9(4):Article No. 25