Accelerating Raw Data Analysis with the ACCORDA Software and Hardware Architecture

Yuanwei Fang
CS Department
University of Chicago
fywkevin@uchicago.edu

Chen Zou
CS Department
University of Chicago
chenzou@uchicago.edu

Andrew A. Chien
University of Chicago
Argonne National Laboratory
achien@uchicago.edu

ABSTRACT

The data science revolution and growing popularity of data lakes make efficient processing of raw data increasingly important. To address this, we propose the ACCelerated Operators for Raw Data Analysis (ACCORDA) architecture. By extending the operator interface (subtype with encoding) and employing a uniform runtime worker model, ACCORDA integrates data transformation acceleration seamlessly, enabling a new class of encoding optimizations and robust high-performance raw data processing. Together, these key features preserve the software system architecture, empowering state-of-art heuristic optimizations to drive flexible data encoding for performance. ACCORDA derives performance from its software architecture, but depends critically on the acceleration of the Unstructured Data Processor (UDP) that is integrated into the memory-hierarchy, and accelerates data transformation tasks by 16x-21x (parsing, decompression) to as much as 160x (deserialization) compared to an x86 core.

We evaluate ACCORDA using TPC-H queries on tabular data formats, exercising raw data properties such as parsing and data conversion. The ACCORDA system achieves 2.9x-13.2x speedups when compared to SparkSQL, reducing raw data processing overhead to a geomean of 1.2x (20%). In doing so, ACCORDA robustly matches or outperforms prior systems that depend on caching loaded data, while computing on raw, unloaded data. This performance benefit is robust across format complexity, query predicates, and selectivity (data statistics). ACCORDA's encoding-extended operator interface unlocks aggressive encoding-oriented optimizations that deliver 80% average performance increase over the 7 affected TPC-H queries.

PVLDB Reference Format:
Y. Fang, C. Zou, and A. A. Chien. Accelerating Raw Data Analysis with the ACCORDA Software and Hardware Architecture. *PVLDB*, 12(11): 1568-1582, 2019.
DOI: https://doi.org/10.14778/3342263.3342634

Proceedings of the VLDB Endowment, Vol. 12, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3342263.3342634

1. INTRODUCTION

Driven by a rapid increase in quantity, type, and number of sources of data, many data analytics approaches use the data lake model. Incoming data is pooled in large quantity – hence the name "lake" – and because of varied origin (e.g., purchased from other providers, internal employee data, web scraped, database dumps, etc.), it varies in size, encoding, age, quality, etc. Additionally, it often lacks consistency or any common schema. Analytics applications pull data from the lake as they see fit, digesting and processing it on-demand for a current analytics task. The applications of data lakes are vast, including machine learning, data mining, artificial intelligence, ad-hoc query, and data visualization. Fast direct processing on raw data is critical for these applications.

The data lake model has numerous advantages over traditional data warehouses. First, raw data is only processed when needed, the data lake model avoids the processing and IO work for unused data. Second, direct access to raw data can reduce the data latency for analysis, a key property for streaming data sources or critical fresh data. Third, data lake supports exploratory analytics where schemas and attributes may be updated rapidly. Fourth, data lakes can avoid information loss during loading by keeping the source information, so applications can interpret inconsistent data in customized fashion. Fifth, the data lake model avoids scaling limits such as maximum capacity of database systems and licensing limits or cost (price/loaded data licensing models). One consequence of the data lake model is that data remains in native complex structures and formats instead of in-machine SQL types with well-defined schemas.

From a system point of view, the essence of the data lake model is to shift ETL/loading work from offline into the query execution. Of course, the shifted work is much smaller - just the part needed for the query. As for each query, only the relevant data is pulled from the lake and processed (essentially load, then analyze) to complete the query.

Because data lakes put raw data interpretation on the critical path of query execution, they elevate raw data processing to a critical performance concern. Queries that require answers quickly, or those that require interactive and real-time analysis, perhaps on recently-arrived raw data, will be sensitive to the speed of raw data processing.

Prior research has made improvements for batch ETL (data loading). For example, several software-acceleration approaches exploit SIMD instructions (InstantLoad[46] – to find delimiters, and MISON [43] – to speculate parsing).

More closely relevant are Lazy ETL approaches that avoid unnecessary ETL by only processing the data actually needed.

Databases [12, 16, 36, 37] apply lazy loading, caching materialized results, and positional maps to avoid expensive data transformations. These approaches require query data reuse, significant memory, and storage resources to achieve high performance. When these properties do not hold, performance can become variable and poor. Another technique for raw data processing is pushing predicates down before the loading work, several systems [15, 37, 47] exploit query semantics for partial predicate pushdown. For these systems, performance is highly dependent on selectivity, and filters are limited to be very simple (literal-equal) and thus do not work for all external data representations (e.g., gzip).

Apache SparkSQL [15] is a widely-used analytics system for processing raw data built on top of Apache Spark [59], a popular in-memory map-reduce framework. SparkSQL offers a SQL-like interface for relational processing, and its popularity has produced a rich ecosystem for data analytics.

We propose ACCORDA, a novel software and hardware architecture that integrates data encoding into operator interfaces and query plans, and uses a uniform runtime worker model to seamlessly exploit the UDP accelerator for data transformation tasks. We implemented ACCORDA, modifying the SparkSQL code base and adding a set of accelerated operators and new optimization rules for data encoding into the Catalyst query optimizer.

In prior work, we designed and evaluated the ACCORDA accelerator (a.k.a the Unstructured Data Processor or UDP [23, 25]), and build on it here. The ACCORDA architecture integrates UDP acceleration in the memory-hierarchy with low overhead. The Unstructured Data Processor is a high-performance, low-power data transformation accelerator that achieves 20x geomean performance improvement and 1,900x geomean performance/watt improvement for a range of ETL tasks over an entire multi-core CPU [25].

We evaluate the ACCORDA architecture on raw data processing micro-benchmarks and TPC-H queries on tabular format to stress parsing flat data and data type conversion, and show that ACCORDA provides robust query performance on raw data that matches or exceeds loaded, cached data analysis. Specific contributions of the paper include:

- ACCORDA software architecture that integrates data transformation acceleration seamlessly by subtyping operator interfaces with data encoding and a uniform runtime worker model – all accelerated. Thus, ACCORDA enables flexible exploitation of encoding-based query transformation and optimization.
- ACCORDA provides robust, high-performance, raw data processing, delivering 2.9x-13.2x speedups on TPC-H queries on tabular data when compared to SparkSQL on CPU. ACCORDA narrows the penalty for raw data analysis from 5.6x to 1.2x, effectively eliminating the overhead relative to computing on loaded, cached data.
- To specifically quantify the benefits of ACCORDA's software architecture, we added encoding-based optimization in the middle of a query plan. These encoding optimizations produce additional 1.1x-11.8x speedups with 80% average performance increase over the 7 affected TPC-H queries.
- Finally, we show ACCORDA's in memory-hierarchy integration enables flexible use of hardware acceleration and improves efficiency in analytics systems (e.g., 1.6x faster and 14x less data movement in a typical select-filter-project query).

The remainder of the paper is organized as follows. Section 2 outlines background on data analytics systems and raw data performance challenges. In Section 3, we illustrate new analytics capabilities enabled by fast data transformation. In Section 4 and 5, we present the ACCORDA software and hardware architecture. We study ACCORDA's performance in Section 7 using the methodology described in Section 6. We discuss and compare our results to the research literature in Section 8, and summarize the results in Section 9.

2. BACKGROUND

2.1 Data Lake or Raw Data Processing

Raw data processing requires in-line data parsing, cleaning, and transformation in addition to analysis. The raw data exists in an external format, often serialized and compressed for portability, that must be parsed to be understood. Typically, only a small fraction of the data is required, so it can be imported to extract what is actually needed. In contrast, database computations on loaded data generally exploit an internal, optimized format for performance.

Furthermore, raw data lacks a clear schema, making extraction difficult. Parsing, decoding, deserialization, and more can be required to transform elements into CPU binary formats. To allow further read optimizations, row/column transposition can be desired. We term these operations collectively as data transformation, and all have a common underlying model derived from the finite automata (FA); these models underly all grammar-based encodings. In many cases, these raw formats are composed of a sequence of tag/value pairs (e.g., Snappy, CSV, JSON, and XML). The corresponding operation is determined by the specific tag value (e.g., delimiter in CSV, tag in Snappy, and key in JSON). These formats often employ sub-word encodings (bytes, symbols, etc.) and variable-length encodings (strings, Huffman, etc.). For each FA state transition, a little work is done. Software usually implements such FAs with costly switch statements or computed branches (multi-target branches) making this computation more expensive than IO (Section 3.2).

2.2 SparkSQL and Optimization

SparkSQL [15] is a relational processing module in Spark [59] using Dataset APIs. It leverages traditional RDBMS optimizations and a tight integration between relational and procedural processing in Spark code. SparkSQL uses a serialized internal format, rather than RDD, in the execution engine to reduce data SerDes cost in shuffling. Tungsten code generation is used to collapse multiple operators together to save interpretation cost and improve data locality. The Catalyst query optimizer uses heuristic rules to expand a set of candidate plans followed by cost-based plan selection. Modern query optimizers (e.g., Hive/Calcite [17], SQL Server [5], and Pig [27]) use rules for applying heuristics during query optimization to greatly reduce the plan search space.

3. TOWARDS FIRST-CLASS RAW DATA PROCESSING

We describe new analysis capabilities enabled by accelerated raw data processing, and then demonstrate CPU's poor performance on it, motivating the ACCORDA approach.

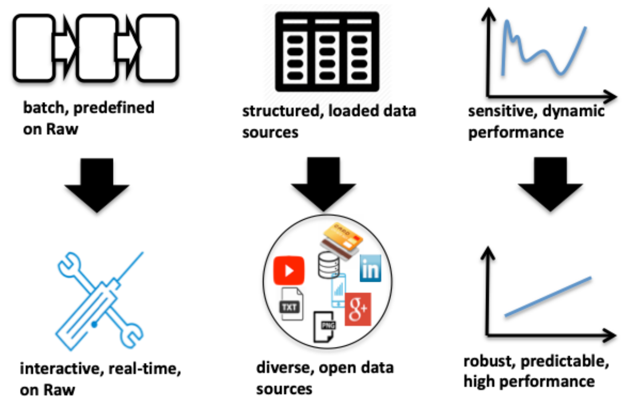

batch, predefined on Raw
structured, loaded data sources
sensitive, dynamic performance
interactive, real-time, on Raw
diverse, open data sources
robust, predictable, high performance

Figure 1: Ideal Raw Data Analysis.

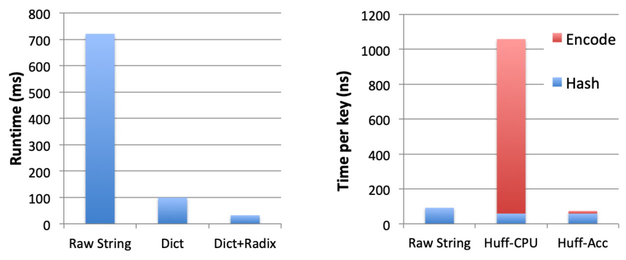
3.1 Fast Filtering, Extraction, and Transformation Unlock New Capabilities

A system with fast data filtering, extraction, and transformation unlocks new capabilities for data analysis. Robust, high-performance, raw data processing enables three dimensions of new analytics capabilities (see Figure 1). First, it enables interactive and real-time analysis on fresh data, avoiding data loading delays - due to scheduling or substantial export-transform-load (ETL). Second, it enables users to write ad-hoc queries that access arbitrary data, without regard for it having been loaded. With no fixed schema, users can compute with flexible, open, user-defined data types and operations on raw data. Uniform fast processing welcomes rich data sources – unstructured web logs, sensor data, social network activities, texts and images, and organized columnar data. This valuable analysis is restricted today by high data transformation cost and rigid schema structure. Third, prior research on acceleration (on-demand loading and in-memory buffering [12, 15, 16]) exploits caching efficiency, making performance sensitive to query history, and requiring large memory. Systems with robust, high-performance raw data processing can avoid these limitations.

High-performance data transformation also unlocks new opportunities in traditional query optimization. The encoding of both data source and intermediate data is a critical cost factor for analytical operations. Consider *sorting*. We sort *customerID* field in the TPC-H dataset [6] that is in the format "Customer#xxxxxxxx". The native string format implies a comparison-based implementation. Transforming to a partitioned dictionary encoding with a dictionary-encoded prefix delivers a 7x speedup (see Figure 2a).[1] Going further, a partitioned-dictionary format enables the use of radix sort (Dict+Radix), producing an overall 22x speedup.

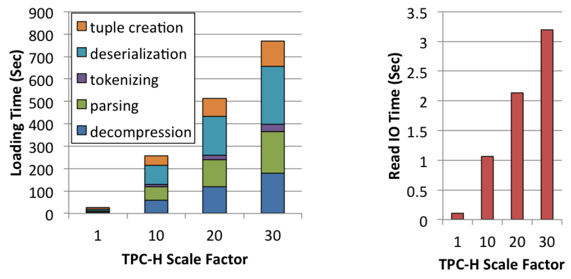
Consider *hashing*, used in hash-partition, hash-join, and hash-aggregate. Raw data is often unnormalized, using multiple long attributes to identify records. Their hashing cost can be significant; for example *murmurhash3* has cost proportional to key size. Compressing keys cheaply can reduce hashing cost. In Figure 2b, we compare applying Huffman encoding with 30% compression ratio (on ≈ 200 bytes per record). With ACCORDA's accelerator (Section 5), the encoding can be done cheaply, producing an overall 20% cost reduction. However, CPU's cannot realize the benefits of such optimizations due to poor encoding performance.

[1]We transform the *id* part as an integer and combine it with the encoded prefix for a 32-bit encoding. The key size reduction and the cheaper comparator increase the performance. Source data uses TPC-H scale factor 1.



(a) Sorting. (b) Hashing.

Figure 2: High-Performance Transformation Unlocks Performance from Encoding Choice.



(a) Cost Breakdown. (b) Disk IO.

Figure 3: High Load Costs: Compressed CSV into PostgreSQL.

3.2 Data Transformation is Expensive on CPU's

Transforming raw data to database formats is costly whether done in batch [46] or just-in-time [12]. For example, Figure 3a shows single-threaded costs to load the TPC-H [6] Gzip-compressed CSV files (scale factor from 1 to 30) from SSD into the PostgreSQL relational database [4] (Intel Core-i7 CPU with 250GB SATA 3.0 SSD). This common extract-transform-load (ETL) task includes decompression, parsing record delimiters, tokenizing attribute values, and deserialization. It requires nearly 800 seconds for scale factor 30 (about 30GB uncompressed), dominating time to initial analysis. Note that CPU execution time for loading is >200x larger than disk IO time (Figure 3b).

CPU's perform poorly on data transformation workloads due to 1) poor support for sub-word, variable-length data, and 2) unpredictable control flows in data transformation tasks. Modern CPU micro-architectures require fifty to hundreds of instructions in flight to achieve full performance, requiring correct branch prediction over a dozen or more branches. However, data transformation is heavily conditional and unpredictable, characterized by multi-target branches (switches) and short code blocks that have been shown to waste 86% of cycles due to misprediction [25]. Expensive data transformation limits the encoding optimizations for data analytics.

3.3 ACCORDA Enables Fast Raw Data Transformation and Processing

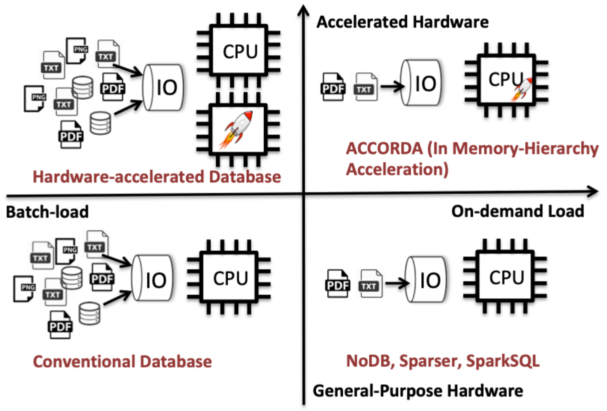


Figure 4: Approaches for Raw Data Analysis.

We present ACCORDA (ACCelerated Operators for Raw Data Analysis), a software and hardware architecture for data analytics systems on raw data. In Figure 4, we illustrate how ACCORDA fits into the landscape of analytics systems in two dimensions – batch/on-demand loading and accelerated/non-accelerated. Conventional databases (lower left) batch load raw data and run on CPU's. Data transformation is expensive on CPU's, consuming significant resources. Conventional accelerated database systems (upper left) use discrete GPU's or FPGA's, focusing on compute acceleration but suffering from high cost to access acceleration. Recently, several systems have focused on raw data processing, loading raw data on-demand and using CPU's (lower right). They include RAW/NoDB [12, 16, 36, 37] that loads on demand, and then caches loaded data in memory lazily to speedup query execution. Spark [15] executes on raw data sources but suffers from poor parsing and deserialization performance. Sparser [47] applies a narrow class of fast filters before data loading to reduce raw data processing cost.

The ACCORDA system (upper right) combines on-demand loading with seamless acceleration that focuses on data transformation, providing predictable, high performance on raw data processing. Key ideas include introducing data encoding types in a query plan, uniform runtime worker model with data transformation acceleration, in memory-hierarchy hardware acceleration integration, and novel query optimizations based on data encoding. In Section 4, we describe the ACCORDA software architecture, and in Section 5, we describe the UDP hardware acceleration that serves as the key enabler for ACCORDA's software architecture.

4. ACCORDA SOFTWARE ARCHITECTURE

We discuss the ACCORDA software architecture, describing the key design choices – extending operators with encoding subtypes and a uniform runtime worker model – that together enable seamless flexible acceleration and optimization. Next, we show how these elements preserve the structure and benefit of traditional analytics engines.

4.1 Encoding in the Operator Interface

ACCORDA extends the operator interface, conceptually adding data encodings as column properties.[2] This effectively subtypes the operators with data encoding, allowing expression of encoding-specialized operators (e.g., filter for RLE-encoded INTs). These extended types (see Figure 5) are used for query optimization and execution, so query plans can capture encodings and data transformation amongst encodings, and optimize them. A transformation can satisfy the encoding requirement of accelerated operator's implementation, or it can be fused to improve data locality and reduce transformation cost. ACCORDA's acceleration enables many new inexpensive data transformation operators.

[2]Blocking and cross-column group optimizations complicate this slightly, but we defer that complexity to later discussion.

```
Operator Interface = <Attribute1, Attribute2, …>
    Attribute = <DataType, Name, Metadata>
    DataType = String | Double | Date | Integer | …

ACCORDA Interface = <Encode-Attrib1, Encode-Attrib2,…>
    Encode-Attrib = <DataType, Name, Metadata, Encode >
    Encode = Native | Dictionary | Huffman | Part-Dictionary | …
```

Figure 5: ACCORDA's Encoding-extended Operator Interface.

A traditional operator interface is a list of *attribute* types. ACCORDA operator interface is a list of <*attribute, encoding*> tuples as illustrated in Figure 5. ACCORDA's current implementation supports four encodings: Native (native format), Dictionary, Huffman, and Partitioned-Dictionary (a prefix of the data item is dictionary-encoded). NATIVE grandfathers legacy operators, enabling co-existence with new explicit encode operators as well as customized variants for improved performance.

Figure 7 gives a concrete example to demonstrate what happens in an encode operator. The encode operator first batches a set of records into a block and sends the block to the data transformation operator (accelerated as in Section 5) for a series of transformations (transpose for efficient dictionary encoding of attribute A, and then transpose back with each record streaming into downstream operators). In Section 5, we revisit this example, showing acceleration detail.

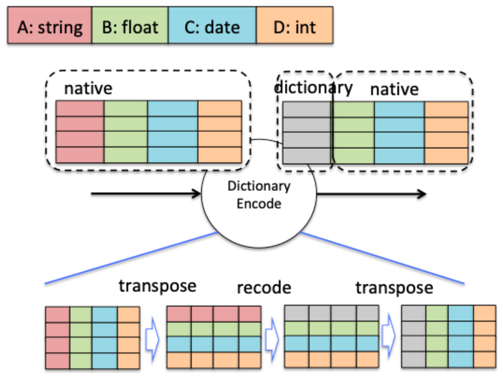


Figure 7: Dictionary Encode Operator.

Example To illustrate the power of encoding subtyping for operators, consider the full query optimization example in Figure 6. We add UDP-accelerated operators (left side), swapping in an accelerated JSON reader (blue) under the traditional operator interface. Extending the operator interface with data encoding (see "native" added to many encodings) enables the four query plan transformations (red).

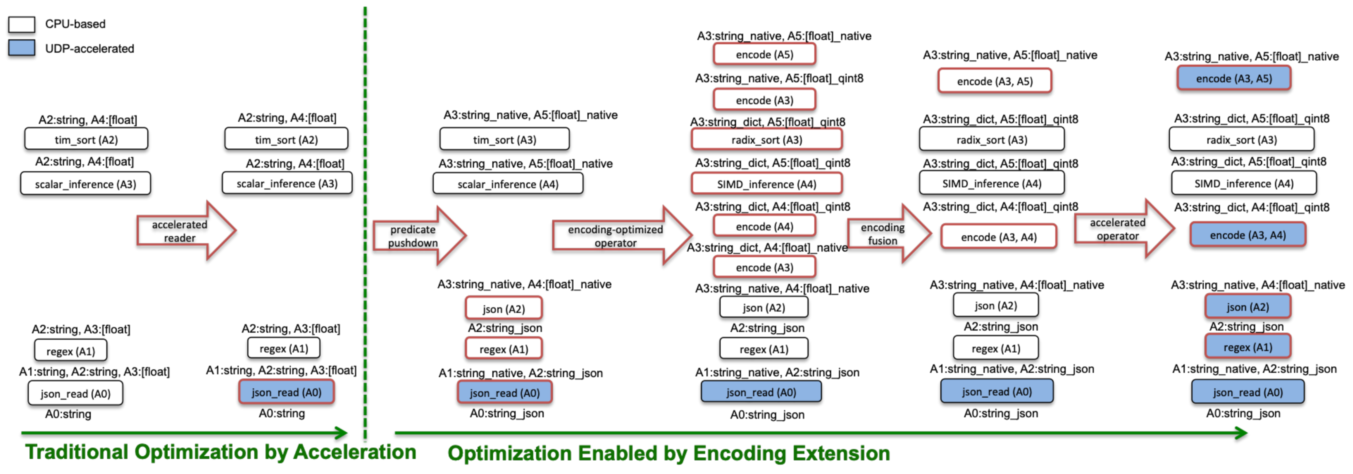


Figure 6: Encoding-Based Query Optimization Example (Encodings Indicated Between Operators).

The encoding for each data attribute is suffixed with "_". For example, in "string_json", *json* is the encoding and *string* is the data type. With the richer encoding-extended interface, the optimizer allows lazy parsing and deserialization by pushing predicates down. Next, it picks an efficient operator implementation with a specific encoding (e.g., inference, sort, and regex). Later steps fuse the introduced encode operators to improve execution efficiency and data locality. Finally, the remaining encode operators can be UDP-accelerated (blue). In summary, the encoding-extended operator interface captures data encodings for intermediate results in a query plan allowing flexible and cascading optimizations.

4.2 Uniform Runtime Worker Model

In ACCORDA, extended operator types enable accelerated and traditional operators to be treated uniformly. ACCORDA also implements a uniform runtime model; all runtime worker threads are accelerated (Figure 8). This uniformity simplifies scheduling of work, and allows queries to run from end-to-end on a single worker without switching, enhancing data locality.

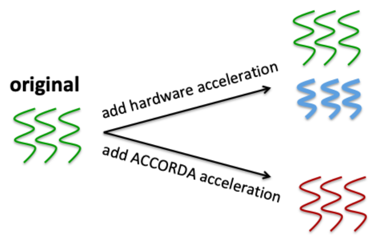


Figure 8: Most Accelerated Systems Create Different Worker Types (Upper). ACCORDA Has Uniform Accelerated Workers (Lower).

Likewise, ACCORDA integrates acceleration seamlessly within the memory-hierarchy, reducing access overhead, creating low-overhead data sharing across accelerated and unaccelerated operators. This is possible because the ACCORDA accelerator is fast, small, and low-power so that a single accelerator is sufficient to support across many CPU cores (see Section 5), and still delivers high speedups (evaluated in Section 7.3). Most other hardware acceleration approaches are forced into looser integration [18, 35, 45, 50] by power, and wind up with two worker types: accelerated and normal. Such an approach complicates scheduling, forcing query execution to switch between workers to exploit acceleration.

4.3 Preserving Software Architecture

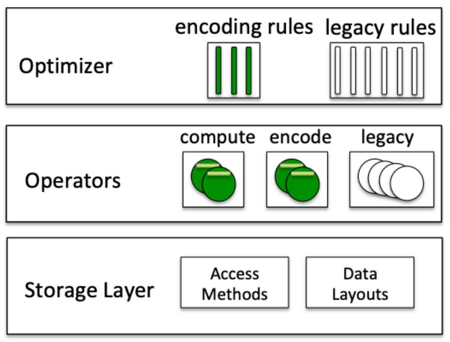


Figure 9: ACCORDA Query Engine Software Architecture (Added Components in Green).

The ACCORDA software system's design preserves conventional data analytics system architecture. For example, in our implementation built on SparkSQL, system components such as query frontend, storage system, memory management, and distributed manager are unchanged. As shown in Figure 9, newly added components include new rules and two classes of new operators (encoding-optimized compute and encode). New elements are depicted in green. The optimization layer gains new data-encoding rules from the rule-based optimizer (e.g., the Catalyst optimizer [15]) that transform the query plans to select good encodings.[3] ACCORDA encapsulates legacy operators using the extended interface, and adds encoding and compute operators, enabling exploitation of format-optimized compute operators.

Importantly, the ACCORDA software architecture avoids disrupting the optimizer. Figure 10 contrasts optimizer

[3]Our evaluation (Section 7) inspired addition of rules for data read, filter, hashing, regex matching and sorting to place data encoding operators in a query plan.

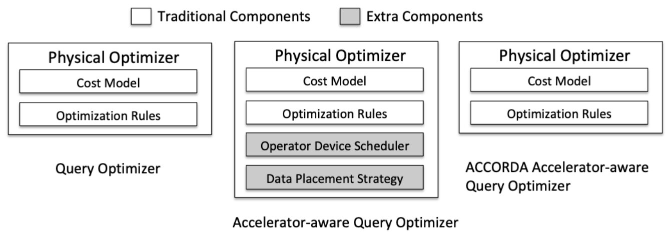


Figure 10: Comparing Optimizer Architectures for Acceleration.

architectures for traditional, accelerated, and ACCORDA-accelerated. The traditional non-accelerated query optimizer consists a cost model and dozens of optimization rules (left). GPU/FPGA-accelerated systems add layers to manage the cost of data transfer to/from PCIe, data transformation overheads, locality impacts, and device scheduling [18] (center). The ACCORDA query optimizer maintains the original optimizer architecture, adding rules, and using cost metrics alone to manage use of acceleration (right). The key to this is the ACCORDA's critical choice of in memory-hierarchy acceleration (Section 5) and uniform runtime worker model.

4.4 Preserving Flexible Query Optimization

4.4.1 *Coarse-grained vs. Fine-grained*

Iterator-based execution (tuple-at-a-time) has well-known virtues for avoiding unnecessary materialization – and corresponding computation and IO [15, 29]. ACCORDA's software architecture, combined with in memory-hierarchy accelerator integration (Section 5.1) allows ACCORDA to preserve the tuple-at-a-time processing model (or at least a block-oriented version).

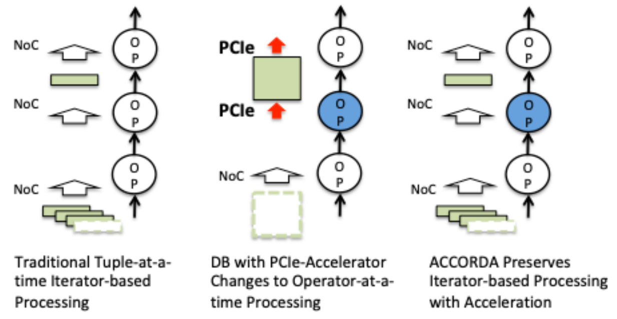

NoC
NoC
NoC
OP
OP
OP
Traditional Tuple-at-a-time Iterator-based Processing
PCIe
PCIe
NoC
OP
OP
OP
DB with PCIe-Accelerator Changes to Operator-at-a-time Processing
NoC
NoC
NoC
OP
OP
OP
ACCORDA Preserves Iterator-based Processing with Acceleration

Figure 11: Comparing Execution Models.

In contrast, accelerated DBMS (e.g., GPU database) generally employs operator-at-a-time processing [18]. Figure 11 shows these differences. This is a consequence of GPU's separate memory systems (copy-in/copy-out, data movement, and synchronization cost), as well as their non-uniform worker model (see Section 4.2). Consequences include large intermediate materialized results, extra computation, and memory demand – particularly for accelerators with limited memory (e.g., GPU's). So the accelerator totally disrupts the optimizer; a common response is to use a completely new strategy that splits a query plan into coarse-grained chunks and schedules each separately. Each chunk runs on the same device using a single processing model (e.g., operator-at-a-time).

ACCORDA enables full, fine-grained query optimization. We reuse the existing rules in SparkSQL. The optimizer treats accelerated operators as the first-class citizens as shown in Figure 12. Rule *a,b* are fired when the sub-tree in a query plan and the estimated statistics meet the transforming condition. Rule *c* replaces a data transformation operator with an accelerated counterpart. Rule *d* transforms a costly regex matching filter expression into a separate accelerated operator. While beneficial in ACCORDA, these fine-grained optimizations would damage performance in a GPU/FPGA-accelerated database systems if they are not constrained by the interaction overheads between CPU's and accelerators.

In Figure 13a, we illustrate the benefits of fine-grained optimization for two TPC-H queries using the platform, system and workload described in Section 6. In Q10, the ACCORDA optimizer replaces the data source operator with an accelerated one (rule c) and combines seven *group by* attributes together and compresses it using Huffman coding (rule b). The follow-on hash aggregation is computed based on the encoded group. Optimization with rule c along achieves a 2.6x speedup than the baseline, and an additional 18% performance improvement with rule b. Similarly in Q13, besides the accelerated data read (rule c), the expensive regex filtering expression is replaced by the hardware-accelerated regex operator (rule d). The combined effect (rule c+d) produces a 13.2x speedup compared to the baseline.

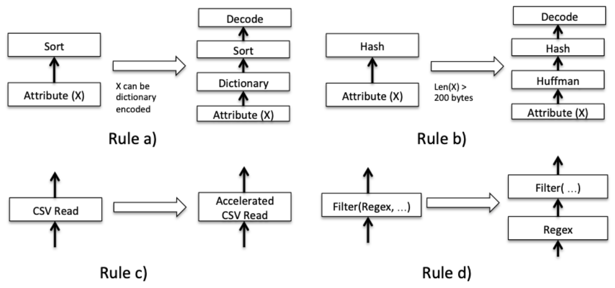


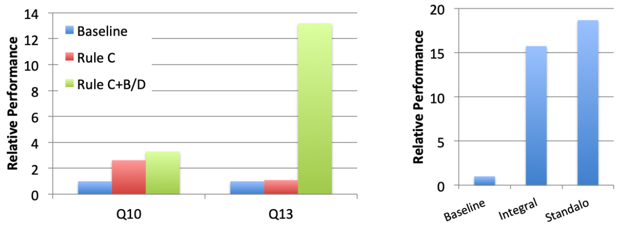
Figure 12: ACCORDA Rule Examples.

4.4.2 *Integral vs. Standalone Encoding*

ACCORDA's encoding-extended operator interface allows encode operators to be decoupled (standalone) from integral implementations (encode-compute-decode). In ACCORDA, attributes can take on varied encodings at each step, reducing operator implementation's encoding overhead. Thus the ACCORDA query optimizer can use rule-based optimizations to choose the best encodings while decode operators are inserted where necessary and delayed after filtering or aggregation to reduce cost. These standalone encoding transformations can later be fused to further improve execution efficiency.

We illustrate the potential gain in Figure 13b with sort-limit query on the *shipdate* attribute of the *lineitem* table on the ACCORDA system. The integral implementation for sorting with hardware acceleration gives a 15x speedup. On the other hand, standalone decode operators allow work to be shifted after the filter operations, producing another 20% performance improvement. Larger benefits are possible with more complicated, aggressive transformation rules.

5. ACCORDA HARDWARE ARCHITECTURE



(a) Q10 and Q13 (b) Sort-Limit

Figure 13: Benefits of ACCORDA Fine-grained Query Optimization.

The ACCORDA hardware system builds on our prior work on the UDP (a high-performance, low-power data transformation accelerator [23, 25]). The unique architectural features of the UDP include multi-way dispatch, variable-size symbol support, flexible-source dispatch, and flexible memory sharing. These features, combined with software programmability, a fast scratchpad memory, and 64 parallel lanes enable a single UDP to provide a 20x geomean speedup over an entire multi-core CPU across a broad variety of export-transform-load computations such as CSV parsing, Huffman encode/decode, regex pattern matching, dictionary encode/decode, run-length encoding, histogram, and snappy compression/decompression (see Figure 17). Furthermore, the UDP's implementation achieves geomean 1,900-fold increase in performance/watt (four orders of magnitude). At a tiny $3.82mm^2$ area, and 0.86 watt [25], the UDP can be integrated on a CPU with minimal cost. Together, these capabilities power ACCORDA's low-cost data transformation at full memory speeds. Next, we explore how to best integrate the UDP into the hardware system architecture and the analytics system software architecture.

5.1 In Memory-Hierarchy Integration

We consider two alternatives for integrating the ACCORDA accelerator into the hardware system: 1) on PCIe and 2) in memory-hierarchy. Historically, most accelerators (GPU's and FPGA's) use PCIe integration [18, 49] both for the convenience of modular "pluggability" as well as the realities of their large die sizes (both FPGA's and GPU's are typically as large as CPU's, requiring 100's of mm^2 for good acceleration performance) and high power (GPU's range from 50-300 watts, and 10-50 watts for FPGA's). As important, GPU's require a dedicated high bandwidth memory for performance, forcing data sharing with the CPU's via copying and expensive data transfers, to utilize the acceleration. The resulting data movement overhead prevents fine-grained interleaved execution between CPU's and accelerators. PCIe-attached acceleration is depicted in Figure 14 (blue).

In ACCORDA, we use "in memory-hierarchy" integration of the accelerator – UDP (Figure 14, green). The UDP is added to the CPU chip, reducing data transfer cost (traffic through the on-chip network). This is only possible with a high-performance, tiny (area), and low-power (0.86 watt) accelerator. The Unstructured Data Processor (UDP) meets all these requirements. A single UDP accelerator is <1% the area of a CPU chip, but delivers 20x data transformation performance in 0.86 watt [25].

We illustrate in memory-hierarchy integration in Figure 14 (green). The CPU core accesses the caches normally, but

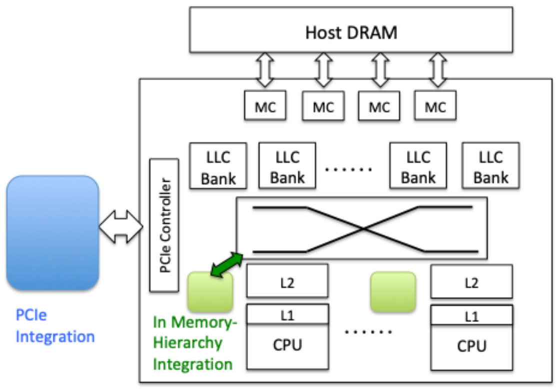


Figure 14: Two Approaches to Integrate Acceleration (In Memory-Hierarchy Integration in Green).

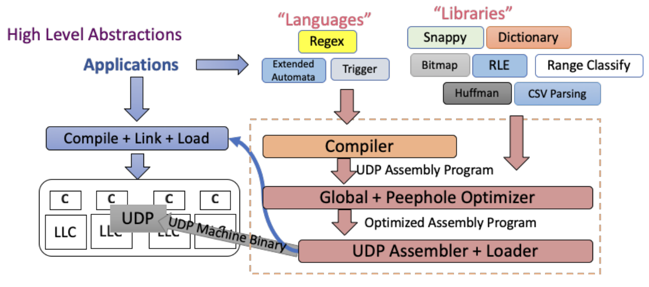


Figure 15: UDP's Software Stack Supports a Wide Range of Transformations. Traditional CPU and UDP Computation Can be Integrated Flexibly.

can also directly access the accelerator's scratchpad memory which is mapped as "uncacheable" (data doesn't appear in regular memory-hierarchy). Data is moved to/from scratchpad memory by libraries using lightweight DMA operations (like memcpy) [52] as in Figure 14 (green arrow). For example, the record block in Figure 7 is moved from CPU caches by the DMA engine into the UDP scratchpad. Next, the UDP program transposes columns with balanced distribution on the 64 UDP lanes. After that, each UDP lane performs encoding. Then, the result is transposed again and DMA'ed to the CPU caches. The key advantage of in memory-hierarchy integration is the low-overhead data sharing between CPU cores and accelerators; we leverage this as a foundation of the ACCORDA software architecture.

5.2 Programming the UDP Accelerator

A key challenge for specialized accelerators is their generality and software-programmability. We carefully designed the UDP instruction set [22] to achieve software-programmability with high-performance. The UDP programs are called as application-level dynamic libraries.

UDP programs can be composed by domain-specific translators, selected from UDP kernel libraries, or high-level languages from third-parties – all sharing a single backend (see Figure 15). The translators support a high-level abstraction and a domain-specific frontend to generate high-level assembly language. The backend takes UDP assembly and does intra-block and cross-block optimizations, but most importantly, it does the layout optimization to achieve high code density for multi-way dispatch [24]. Furthermore, it optimizes action block sharing to achieve small code size.

6. METHODOLOGY

6.1 System Modeling

We compare ACCORDA performance to a baseline of SparkSQL running on a 14-core, 28-thread CPU (Intel Xeon E5-2680) with 64GB DDR3. We use default Spark/SparkSQL configuration and a single worker with 10GB memory.

ACCORDA's performance is based on a software prototype derived from Apache SparkSQL (version 2.2) with the query engine and optimizer extensions described in Section 4. This includes the cost of using Java Native Interface (JNI) to make data accessible to the accelerator. ACCORDA system performance is determined by a timed hybrid-simulation modeling methodology. We utilize the Intel HARP system (two-socket system with the same Intel Xeon E5-2680 and an Altera Arria10 FPGA connected by QPI) to achieve high-speed execution for large workloads [7]. We capture event logs with timing (e.g., timestamps) and execution labels.

Because the FPGA emulation of the UDP as well as the QPI emulation of in memory-hierarchy integration are both slower than a real implementation, we re-time the traces according to Figure 16 and the data in Table 1. ACCORDA workers run across the CPU and UDP with no overlap. UDP execution is scaled up to 1Ghz ASIC speed [25], and data movement of in memory-hierarchy transfers is scaled by the ratio of DMA bandwidth over QPI. The trace log is post-processed to implement the hybrid-simulation model and reflect the updated task time. This hybrid performance modeling is conservative since it assumes no overlap between the CPU core and acceleration, and performance is penalized by cache pollution due to simulation and logging.

Table 1: Hardware Platform Statistics.

	FPGA Emulation	Actual
DMA Bandwidth	3 GB/s	30 GB/s
Clock Frequency	40 MHz	1 GHz
Resources	295K ALM, 10Mb BRAM	8.69mm^2

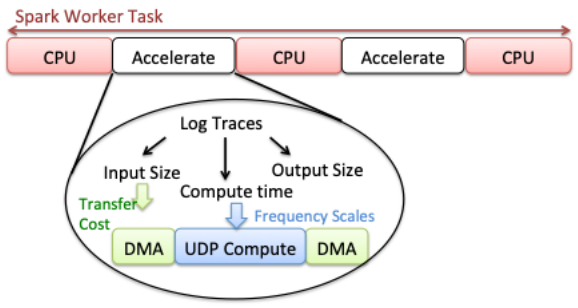


Figure 16: Timed Hybrid-Simulation Model.

6.2 Workload

We use the TPC-H dataset [6] and the entire 22 TPC-H queries for evaluation, spanning both simple select-project-join structures to more complicated ones with expensive expressions and large intermediate attributes. We use source raw data in tabular format (TBL) with different fields separated by delimiter '|'. Each field is in text format rather than binary, exercising the flat parsing and data type conversion dimensions of raw data processing.

A more aggressive raw data study would include deep nested structures and unnormalized relations, so our estimates of ACCORDA benefit are conservative. Raw data workloads containing these richer raw properties will further tax conventional systems, causing them to spend more time on complex data parsing and conversion. In contrast, ACCORDA will find more opportunities to apply hardware acceleration and to employ aggressive encoding optimizations on nested structures, unnormalized relations, and so on.

7. EVALUATION

Evaluation of ACCORDA system includes micro-benchmark for the hardware accelerator (see [25] for a more complete study), its integration, query performance versus alternate raw data approaches, and performance sensitivity studies.

7.1 ACCORDA Accelerator Micro-benchmark

7.1.1 Regex Matching

UDP accelerates regular expression matching dramatically. Regex is important for data filtering SQL operators (e.g., LIKE and RLIKE). Figure 17a compares performance on a single regex pattern from TPC-H Q13, showing the performance of ACCORDA accelerator (UDP), an 8-thread CPU (Intel Hyperscan [3]), an FPGA [50] (using the best performance number), and a commercial ASIC implementation (Titan [53]). The 8-thread CPU achieves 12.8 GB/s, and the FPGA nearly doubles that. The Titan accelerator performs similar to the 8-thread CPU but at much less power. UDP achieves 64 GB/s, 4.9x faster than the 8-thread CPU.

7.1.2 Decompression

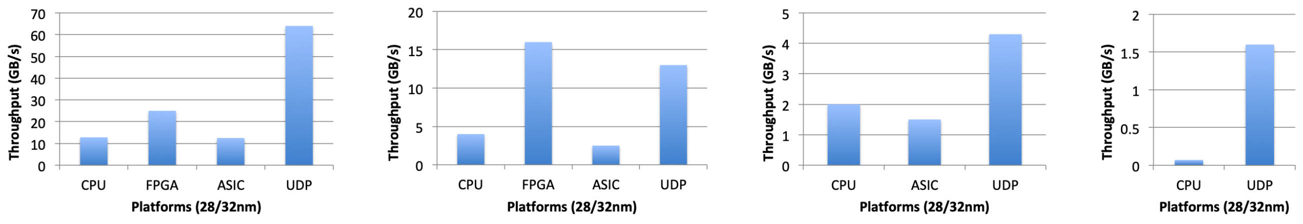
UDP accelerates data compression, which is widely-used to reduce raw data storage cost. For decompression, we compare the UDP to the 8-thread CPU-based Snappy [1] library, a commercial FPGA implementation [8] using Virtex 7 (28nm) and 28Mb BRAMs, and a commercial ASIC (Intel [2]). Our results (see Figure 17b) show UDP matches best performance (13 GB/s) but at a much smaller fraction of power and area cost required for the FPGA implementation. ASIC delivers less performance with significantly lower power. UDP is 2.6x faster than the 8-thread CPU.

7.1.3 Parsing

UDP accelerates parsing, which is critical to extract relevant fields from unstructured raw data. The parsing involves finding and validating delimiters, and extracting interesting fields. In Figure 17c, we compare the UDP performance to an 8-thread CPU (SIMD-optimized parser [46]) and an ASIC design for parsing CSV/JSON/XML [14]. The UDP achieves 4.3 GB/s throughput, 2x the performance of the 8-thread CPU (2 GB/s), and 3x greater than the ASIC design.

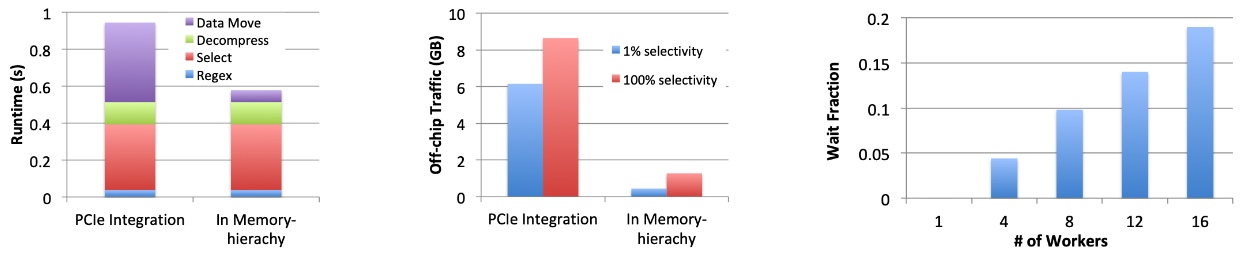
7.1.4 Deserialization

UDP can accelerate data transformation from external format to native machine format (deserialization). Deserialization happens when raw data is in machine-neutral interchangeable format. In Figure 17d, we use the UDP to



(a) Regular Expression Matching. (b) LZ77 Decompression. (c) Parsing. (d) Deserialization.

Figure 17: ACCORDA Accelerator Micro-benchmark Performance (UDP vs. 8 CPU Threads).



(a) Query Runtime. (b) Off-chip Data Movement. (c) Scalability (Sharing Overhead).

Figure 18: Evaluation of In Memory-Hierarchy Integration and Acceleration Scalability.

accelerate ASCII to DATE type conversion. It is one of the most expensive data transformation tasks in raw data processing [12, 37]. We compare the Java deserialization CPU library to our UDP-based implementation. UDP achieves 1.6 GB/s throughput, 20x faster than the 8-thread CPU.

7.2 Benefits of In Memory-Hierarchy

In memory-hierarchy accelerator integration lays the foundation for the ACCORDA software architecture, enabling data transformation to improve performance. We compare two approaches in terms of accelerator integration: UDP accelerator in memory-hierarchy vs. on PCIe. The total performance and data movement are modeled using the LogCA model [13]. In this experiment, we use the TPC-H *order* table in the compressed tabular format. The query is simple: **SELECT** *date, comment* **FROM** *order* **WHERE** *comment* **RLIKE** *".*special.*requests.*"*.

Figure 18a shows the runtime breakdown for three phases: decompression, parsing-select, and regex filtering. Decompression and regex filtering are offloaded completely, but select requires CPU-accelerator collaboration. This data sharing across the PCIe bus increases overhead by 66%. Figure 18b shows that varying the *comment* attribute from 1%-100% selectivity, resulting PCIe-integration to cause 6.8x-14x higher data movement. In memory-hierarchy accelerator integration preserves the iterator-based execution model, allowing ACCORDA to preserve fine-grained data sharing and interleaved execution between CPU cores and the accelerator.

7.3 Acceleration Scalability

Acceleration performance must be sufficient to support both multiple cores and accelerated worker uniformity. We study the sharing of one UDP accelerator (64 lanes) across multiple cores by increasing the number of workers. Results (see Figure 18c) show that with 16 workers, the sharing increases accelerated runtime by less than 20% on TPC-H Q1 (data transform, filter, hash aggregate, float computation, and sort) with "weak-scaling". When combined with the accelerator's 304x performance advantage on these computations, after data movement and sharing overhead, each of the 16 workers still experiences an overall 3.6x speedup, producing large benefit for the entire query.

7.4 Benefits of Software-Programmability

The ACCORDA accelerator is software-programmable. FPGA's are popular alternative: hardware-programmability. We use a set of data transformation and filtering tasks to compare performance achieved per unit silicon area (performance density). FPGA's use LUTs (look up table) and interconnection configuration to achieve programmability, paying silicon area overhead and lower clock rate for flexibility. In contrast, the UDP hardwires an instruction set architecture (basic primitives) and runs software written to that ISA. This enables the UDP implementation to use silicon area efficiently and achieve a high clock rate.

Figure 20 compares performance density for the decompression and the regular expression matching (1 and 500 patterns). For FPGA's, we report a Xilinx's commercial design [8] for decompression, a database-oriented design [50] for single-pattern regex, and a network monitoring based design [58] for 500-pattern regex. Area is calibrated LUT's used [54], excluding BRAMs. For the UDP, we present two metrics. The first is program area (program size in the SRAM), and the second adds UDP logic area. In all cases, UDP is more area-efficient (see Figure 20), with 19x better performance density for decompression and 160x for single-pattern and 90x for 500-pattern regex. Here, software-programmability provides significantly higher performance density than hardware-programmability.

7.5 ACCORDA Raw Data Performance

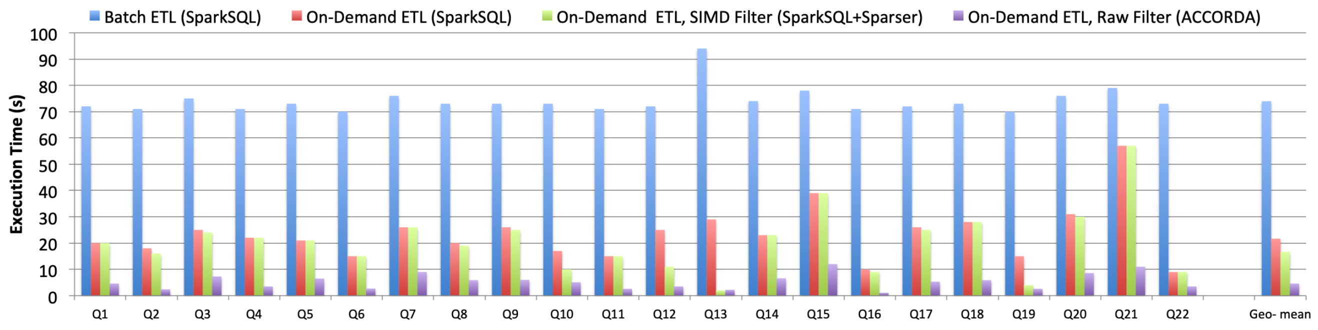


Figure 19: TPC-H Performance (External, Raw Data).

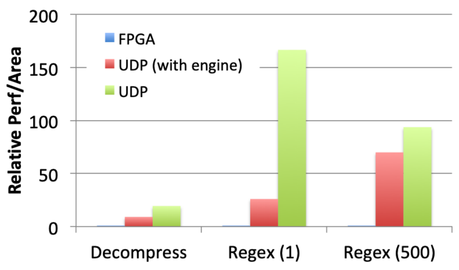


Figure 20: UDP Software Programmability vs. FPGA Hardware Programmability.

Comparing Raw Data Approaches We compare ACCORDA system performance to several alternate approaches: 1) SparkSQL [15] with batch loading of the TPC-H tables into memory beforehand, 2) SparkSQL with on-demand ETL without caching any transformed data (the worst case for RAW/NoDB approach [12]), and 3) Sparser [47] on SparkSQL with on-demand ETL (SIMD-accelerated partial pre-filtering) with data and queries detailed in Section 6.2.

Our results (see Figure 19) show that ACCORDA improves query performance for all 22 queries, and not only compared to batch ETL (16x geomean), but also relative to on-demand ETL (4.6x geomean), and optimized on-demand SIMD filtered ETL (3.6x geomean). The reasons for ACCORDA's benefits compared to batch ETL (only load what's needed) and on-demand ETL (UDP acceleration of data parsing, conversion, and encoding optimization) are straightforward, but the comparison to Sparser bears further discussion.

Table 2: Sparser and ACCORDA Predicate Support (TPC-H 22 Queries). S = Sparser, A = ACCORDA, O = TPC-H Occurrence.

Predicate Type	Example	S	A	O
Multi-Column	col1 = col2	0	15	15
Range Compare	10 < col < 20	0	40	40
IN Operator	col IN (val1, val2, ...)	0	5	5
Literal Equal	col = "abcd"	32	32	32
Regex (partial)	col RLIKE "ab.*[0-9]{2}"	(6)	6	6
	Total Supported	32	98	98

Sparser tries to avoid parsing and deserialization by using literal comparisons to raw bytes. This limits the types of predicates and raw data encodings that can be covered, and the effectiveness of the filtering. For example, gzipped data cannot be filtered. Table 2 categorizes the predicates used in TPC-H queries, comparing Sparser and ACCORDA's ability to support. Sparser's SIMD approach supports 32 (and 6 regex filters partially) out of 98 predicates, limited to *Literal Equal* predicates and sub-string part of *Regex* predicates. ACCORDA's flexible hardware acceleration enables it to support a broad range of predicates, including *Regex* with complex semantics (e.g., range and repetition), multiple columns, range comparison, or IN operator. Thus, ACCORDA supports all 98 predicates in the TPC-H queries – general acceleration without constraints on predicate semantics.

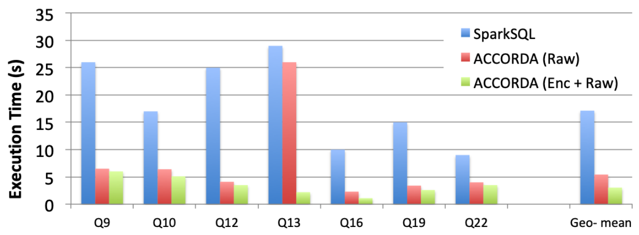


Figure 21: Software Architecture Enabled Performance Benefit (Selected TPC-H Queries).

ACCORDA Software Architecture Benefits Next, we explore the benefits of ACCORDA's software architecture that enables aggressive encoding-based optimizations. Figure 21 compares ACCORDA (Raw) and ACCORDA (Enc+Raw) on selected queries Q9, Q10, Q12, Q13, Q16, Q19 and Q22. ACCORDA (Raw) shows acceleration of data loading, but ACCORDA (Enc+Raw) shows how further aggressive encoding-based query optimization in the middle of a query plan can further increase performance. Specifically, Q10 benefits from attribute group compression, Q12 from dictionary encoding for fast filtering, and Q9, Q13, Q16, Q19, Q22 from accelerated operators for multi-pattern matching and regular expression filtering. Overall, the ACCORDA software architecture enables encoding-based query optimizations that yield 1.1x to 11.8x speedups beyond raw data acceleration, a software architecture benefit of 1.8x geomean (80%!). This workload contains only flat parsing (see Section 6.2); we expect larger benefits for raw data with nested structures and unnormalized relations with long attributes.

ACCORDA Raw vs. Loaded, Cached Analysis To see how much raw data processing overhead remains, we compare ACCORDA raw data analysis to loaded data analysis

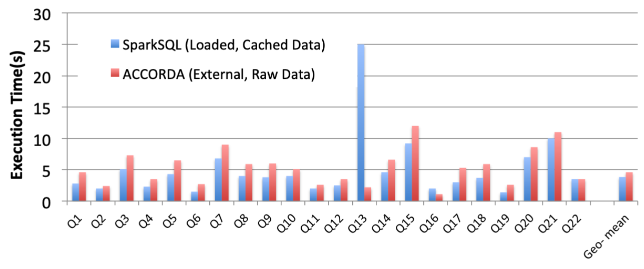


Figure 22: ACCORDA Raw Data Processing vs. Loaded, Cached Data Analysis.

(SparkSQL Loaded, Cached Data). In SparkSQL (Loaded, Cached Data), data has been parsed, transformed, transposed, loaded into its internal columnar-format, and cached into main memory for fast access. Thus it approximates both traditional database (with loaded data) and the best case for RAW/NoDB [12] when all raw data accesses hit in-memory buffer with only selected columns fetched. Our results, in Figure 22, show raw data processing overhead between ACCORDA (External, Raw Data) and SparkSQL (Loaded, Cached Data) is reduced to 1.2x overall (geomean) from 5.6x. Q13 is an outlier for loaded data, suffering from poor regex performance. The raw data overhead is 1.3x geomean excluding Q13. The key reasons for this improvement are ACCORDA's encoding-based optimization and hardware acceleration. The remaining overheads are due to ACCORDA's lack of projection pushdown, floating number conversion, and memory caching – tailored for loaded data, and not so useful for raw data. Overall, ACCORDA reduces raw data processing overhead to only 20%.

In summary, ACCORDA nearly eliminates transformation cost for raw data, enabling it to outperform other raw data approaches and achieve competitive performance against loaded, cached data analysis. Encoding optimization is critical, enabled by ACCORDA's software architecture, contributing 1.1x-11.8x speedups beyond data loading acceleration. Overall, ACCORDA gives speedups of 16x vs. batch loading, 4.6x vs. on-demand ETL, and 3.6x vs. SIMD filtering (Sparser).

7.6 Sensitivity to Data Statistics

In raw data processing, early partial filtering with inexpensive predicates [15, 47] can avoid the cost of data transformation (e.g., parsing, deserialization) and subsequent query computation. We study ACCORDA's benefit on supported predicate functionality, filter selectivity, cost of predicates, and volume of data. As in Table 2, ACCORDA's flexible hardware acceleration provides excellent coverage.

We compare ACCORDA with different raw data processing approaches using three predicates – *Range Compare, Literal Equal, Regex Match* – and varying their selectivity (see Figure 23). As shown in Figure 23, On-Demand ETL (SparkSQL) suffers from CPU's slow data transformation thus delivers poor performance in all cases. Sparser leverages SIMD-accelerated filters to have less data to parse, deserialize, and filter. However, Sparser can't support range-based predicates existed in Q1. In Q12 and Q13, Sparser pushes substrings in the predicates on raw data, thus achieves speedups. Sparser's performance is sensitive to data statistics (selectivity) in these cases. On the other hand, ACCORDA's hardware

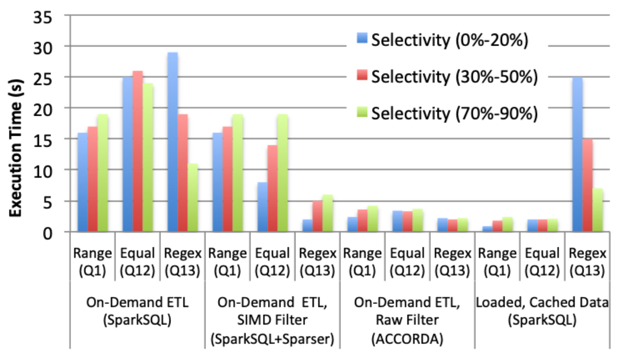


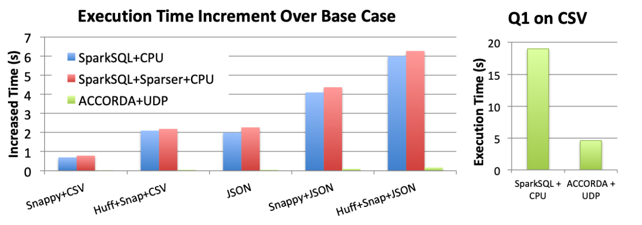
Figure 23: Selectivity Impact on Execution Time.

acceleration enables robust support for all TPC-H predicates, reducing data loading cost. For a wide range of data statistics, it is approaching loaded, cached data analysis.

7.7 Format Type and Complexity Sensitivity

Open data formats are pervasive in data lakes, enabling size optimization, readability, execution performance, and portability. The resulting format complexity incurs raw data processing cost [12, 37, 43, 47] that grows with data complexity. We study its impact on ACCORDA's benefits.

Figure 24a compares TPC-H Q1 execution time on raw data with a succession of more complex formats (Snappy + CSV, Huff+Snappy+CSV, JSON, Snappy+JSON, and Huff+Snappy +JSON). CPU processing cost increases with format complexity – *lineitem* in CSV format increases by 32% when data is in Huffman+Snappy+JSON format. Sparser sees a similar increase. The ACCORDA system, with capable hardware acceleration, sees minimal increase, only 3.7%. As shown in Figure 24b, ACCORDA with its flexibly-programmable acceleration (UDP) delivers a robust 4x speedup vs. SparkSQL on a variety of source format complexity.



(a) Varied Format Complexity. (b) Base Case.

Figure 24: Increased Format Complexity: Performance Penalty.

8. DISCUSSION AND RELATED WORK

Optimizing Raw Data Processing Several SIMD acceleration techniques target parsing in ETL. InstantLoad [46] exploits SIMD parallelism to find delimiters, reducing branch misses by 50%, to achieve single-thread performance over 250 MB/s (x86 CPU). Our UDP delivers 40x better single thread performance and easily saturates a DDR memory channel. MISON [43] exploits speculative parsing and SIMD acceleration for JSON. A list of pattern trees is built from a training data set, optimizing a speculative parsing pattern. However, effectiveness depends heavily on template variety and good sample data. ACCORDA accelerates JSON parsing robustly and supports more structures (e.g., XML).

SCANRAW [20] is a pipelined architecture for loading raw data, integrated as a meta-operator. Speculative loading attempts to mitigate high ETL cost by loading whenever there is spare disk IO bandwidth. In contrast, ACCORDA doesn't load data but accelerates raw data processing directly.

RAW/NoDB [12, 16, 36, 37] applies lazy loading, caching materialized results, and positional maps to raw data processing to avoid ETL/data transformation. It also caches transformed records if there is reuse. RAW/NoDB can trade memory for performance. Data Vault [33] takes a similar approach, using query-driven memory caching of multidimensional arrays from scientific raw files. These results show several limitations. First, query performance on first-time touched columns suffers from slow data loading. Second, because caching depends on reuse, query performance is sensitive to pattern and history – queries that share little in a large data lake or few columns will have poor performance. Third, effective caching requires significant memory capacity (e.g., 32GB [16]); NoDB's positional maps also consume significant space. ACCORDA uses hardware to accelerate raw data processing, achieving robust, good performance.

Other systems [15, 37, 47] exploit query semantics for predicate pushdown. Inexpensive partial predicates are used to filter raw data, avoiding downstream transformation and processing cost. As in Table 2, predicate types and hardware support limit their applicability. ACCORDA has good predicate coverage, achieving inexpensive filtering broadly.

Data Representations for Fast Query Execution Database representation strongly affects query performance. C-store [9, 51] demonstrates that column-oriented storage layouts with rich encodings (e.g., bitmap, dictionary, and run-length-encodings) can optimize query performance on OLAP workloads. Recently, more advanced data representations [26, 39, 42, 44] have been proposed to exploit SIMD accelerations for fast query scan while preserving storage size reduction. Compressed formats are designed to allow direct processing without decompression [10, 60] under a limited set of query semantics. In ACCORDA, we consider dynamic data transformation acceleration during query execution on raw data without any constraints on query semantics.

Such compressed data representations can accelerate iterative machine learning algorithms [21, 40]. They are designed to allow direct computation during training. Zukowski et al. [61] explored dynamic transposition between row and column formats during query execution. The results suggest distinct layouts benefit different operations (e.g., scan for column and aggregation for row). On the other hand, many database operators can be accelerated by applying SIMD, GPU's or FPGA's [35, 48, 50, 57]. However, many accelerated operators require a special data layouts, and thus could benefit from ACCORDA's fast data transformation, enabling more general exploitation of customized formats.

Database Hardware Accelerators Data centers are now incorporate varied accelerators (e.g., ASIC's, FPGA's, and GPU's) [30, 34, 49]. One thread is accelerators for data analytics. Accelerator designs have been proposed for query procssing bottlenecks [11, 28, 35, 38, 41, 50, 55, 56]. Researchers have explored customized accelerators for common operators in SQL such as join [56], partition [11, 55, 56], sort [56], aggregation [56], regex search [28, 50, 56], and index traversal [38]. Deploying an accelerator for each task is challenging – design and deployment cost, and obsolescence when new algorithms emerge. Closest is the Oracle DAX [41], accelerating column scan on compressed formats. DAX only supports fixed formats; the UDP accelerator is flexibly-programmable, matching DAX performance and efficiency, and is extensible to broader existing and future encodings.

Active research explores the use of FPGA's to accelerate data analytics [19, 31, 32, 35, 45, 49, 50]. Computation kernels are implemented on FPGA's as co-processors. DAnA [45] automatically generates FPGA implementations from UDFs for iterative machine learning training jobs in RDBMS. Other work explores FPGA's for regex matching [50], data partitioning [35] and histogram generation [32]. However, for interactive queries, building one circuit for each algorithm is impractical due to the latency and resource requirement. FPGA acceleration on multiple operations in a query requires runtime reconfiguration with additional performance penalty. Furthermore, the reconfigurable LUTs in FPGA's suffer from low energy efficiency [54]. Our UDP accelerator provides data transformation flexibility with high-efficiency and flexible software-programmability. Results in Section 7.4 show its advantages over FPGA's in performance and area efficiency.

9. SUMMARY AND FUTURE WORK

ACCelerated Operators for Raw Data Analysis (ACC ORDA) is a software and hardware architecture that extends the operator interface with encoding and uses a uniform runtime worker model to seamlessy integrate acceleration. ACCORDA improves data lake / raw data processing performance by 2.9x-13.2x on TPC-H benchmarks, reducing raw data processing overhead to 20%, and its extended operator interface unlocks aggressive encoding-oriented optimizations that deliver an 80% average speedup on the affected TPC-H queries. These experiments exercise flat parsing and data type conversion; larger benefits will accrue for nested structures and unnormalized relations that require complex parsing and expose more encoding optimization opportunities.

ACCORDA's flexible query optimization across data encodings and good performance expose new opportunities. Interesting directions include: novel data formats (customized to application or data), aggressive query optimizations for open data type encodings, and new computation kernels / operators tuned to specific data encodings. Further, ACCORDA integrates acceleration deeply into both hardware and software architectures, preserving and extending traditional query optimization structures and maximizing the breadth of acceleration applicability. This creates the opportunity to extend data analytics to a growing diversity of user-defined data types, addressing the expressiveness gap between programming and query languages.

10. ACKNOWLEDGMENTS

We thank our colleagues at UChicago and the VLDB reviewers for their detailed feedback. This research was supported in part by Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of the DOE Office of Science and the National Nuclear Security Administration. Support from the CERES Center for Unstoppable Computing and the Texas Advanced Computing Center (TACC) is also gratefully acknowledged.

11. REFERENCES

[1] Google snappy compression library. `https://github.com/google/snappy`.

[2] Intel chipset 89xx series. `http://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/scaling-acceleration-capacity-brief.pdf`.

[3] Intel hyperscan. `https://github.com/01org/hyperscan`.

[4] Postgresql database. `https://www.postgresql.org/`.

[5] Query processing architecture guide. `https://docs.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-2017`.

[6] Tpc-h benchmark. `http://www.tpc.org/tpch/`.

[7] Xeon+fpga platform for the data center. `https://www.ece.cmu.edu/~calcm/carl/lib/exe/fetch.php?media=carl15-gupta.pdf`.

[8] Zipaccel-d gunzip/zlib/inflate data decompression core. `http://www.cast-inc.com/ip-cores/data/zipaccel-d/cast-zipaccel-d-x.pdf`.

[9] D. Abadi, S. Madden, and M. Ferreira. Integrating compression and execution in column-oriented database systems. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, June 2006.

[10] R. Agarwal, A. Khandelwal, and I. Stoica. Succinct: Enabling queries on compressed data. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015.

[11] S. R. Agrawal, S. Idicula, A. Raghavan, E. Vlachos, V. Govindaraju, V. Varadarajan, C. Balkesen, G. Giannikis, C. Roth, N. Agarwal, et al. A many-core architecture for in-memory data processing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 245–258. ACM, 2017.

[12] I. Alagiannis, R. Borovica, M. Branco, S. Idreos, and A. Ailamaki. Nodb: efficient query execution on raw data files. In *SIGMOD*, pages 241–252. ACM, 2012.

[13] M. S. B. Altaf and D. A. Wood. Logca: a performance model for hardware accelerators. *IEEE Computer Architecture Letters*, 14(2):132–135, 2015.

[14] K. Angstadt, A. Subramaniyan, E. Sadredini, R. Rahimi, K. Skadron, W. Weimer, and R. Das. Aspen: A scalable in-sram architecture for pushdown automata. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 921–932. IEEE, 2018.

[15] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia. Spark sql: Relational data processing in spark. In *SIGMOD*. ACM, 2015.

[16] T. Azim, M. Karpathiotakis, and A. Ailamaki. Recache: reactive caching for fast analytics over heterogeneous data. *PVLDB*, 11(3):324–337, 2017.

[17] E. Begoli, J. Camacho-Rodríguez, J. Hyde, M. J. Mior, and D. Lemire. Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In *Proceedings of the 2018 International Conference on Management of Data*, pages 221–230. ACM, 2018.

[18] S. Breß, M. Heimel, N. Siegmund, L. Bellatreche, and G. Saake. Gpu-accelerated database systems: Survey and open challenges. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XV*, pages 1–35. Springer, 2014.

[19] A. Caulfield, E. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, H. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger. A cloud-scale acceleration architecture. In *MICRO'16*. ACM/IEEE, 2016.

[20] Y. Cheng and F. Rusu. Scanraw: A database meta-operator for parallel in-situ processing and loading. *ACM Transactions on Database Systems (TODS)*, 40(3):19, 2015.

[21] A. Elgohary, M. Boehm, P. J. Haas, F. R. Reiss, and B. Reinwald. Compressed linear algebra for large-scale machine learning. *PVLDB*, 9(12):960–971, 2016.

[22] Y. Fang and A. A. Chien. Udp system interface and lane isa definition. *The University of Chicago Technical Report, TR-2017-05*, May 2017.

[23] Y. Fang, T. T. Hoang, M. Becchi, and A. A. Chien. Fast support for unstructured data processing: The unified automata processor. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, Dec. 2015.

[24] Y. Fang, A. Lehane, and A. A. Chien. Effclip: Efficient coupled-linear packing for finite automata. *The University of Chicago Technical Report, TR-2015-05*, May 2015.

[25] Y. Fang, C. Zou, A. J. Elmore, and A. A. Chien. Udp: a programmable accelerator for extract-transform-load workloads and more. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 55–68. ACM, 2017.

[26] Z. Feng, E. Lo, B. Kao, and W. Xu. Byteslice: Pushing the envelop of main memory data processing with a new storage layout. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 31–46. ACM, 2015.

[27] A. Gates, J. Dai, and T. Nair. Apache pig's optimizer. *IEEE Data Eng. Bull.*, 36(1):34–45, 2013.

[28] V. Gogte, A. Kolli, M. J. Cafarella, L. D. Antoni, and T. F. Wenisch. Hare: Hardware accelerator for regular expressions. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2016.

[29] G. Graefe. Volcano/spl minus/an extensible and parallel query evaluation system. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):120–135, 1994.

[30] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, et al. Applied machine learning at facebook: a datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 620–629. IEEE, 2018.

[31] Z. István, G. Alonso, M. Blott, and K. Vissers. A flexible hash table design for 10gbps key-value stores on fpgas. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–8. IEEE, 2013.

[32] Z. Istvan, L. Woods, and G. Alonso. Histograms as a side effect of data movement for big data. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, June 2014.

[33] M. Ivanova, M. Kersten, and S. Manegold. Data vaults: a symbiosis between database technology and scientific file repositories. In *International Conference on Scientific and Statistical Database Management*, pages 485–494. Springer, 2012.

[34] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12. IEEE, 2017.

[35] K. Kara, J. Giceva, and G. Alonso. Fpga-based data partitioning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 433–445. ACM, 2017.

[36] M. Karpathiotakis, I. Alagiannis, and A. Ailamaki. Fast queries over heterogeneous data through engine customization. *PVLDB*, 9(12):972–983, 2016.

[37] M. Karpathiotakis, M. Branco, I. Alagiannis, and A. Ailamaki. Adaptive query processing on raw data. *PVLDB*, 7(12):1119–1130, 2014.

[38] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan. Meet the walkers: Accelerating index traversals for in-memory databases. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 468–479. ACM, 2013.

[39] H. Lang, T. Mühlbauer, F. Funke, P. A. Boncz, T. Neumann, and A. Kemper. Data blocks: hybrid oltp and olap on compressed storage using both vectorization and compilation. In *Proceedings of the 2016 International Conference on Management of Data*, pages 311–326. ACM, 2016.

[40] F. Li, L. Chen, A. Kumar, J. F. Naughton, J. M. Patel, and X. Wu. When lempel-ziv-welch meets machine learning: A case study of accelerating machine learning using coding. *arXiv:1702.06943*, 2017.

[41] P. Li, J. L. Shin, G. Konstadinidis, F. Schumacher, V. Krishnaswamy, H. Cho, S. Dash, R. Masleid, C. Zheng, Y. D. Lin, et al. 4.2 a 20nm 32-core 64mb l3 cache sparc m7 processor. In *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*, pages 1–3. IEEE, 2015.

[42] Y. Li, C. Chasseur, and J. M. Patel. A padded encoding scheme to accelerate scans by leveraging skew. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1509–1524. ACM, 2015.

[43] Y. Li, N. R. Katsipoulakis, B. Chandramouli, J. Goldstein, and D. Kossmann. Mison: a fast json parser for data analytics. *PVLDB*, 10(10):1118–1129, 2017.

[44] Y. Li and J. M. Patel. Bitweaving: fast scans for main memory data processing. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 289–300. ACM, 2013.

[45] D. Mahajan, J. K. Kim, J. Sacks, A. Ardalan, A. Kumar, and H. Esmaeilzadeh. In-rdbms hardware acceleration of advanced analytics. *PVLDB*, 11(11):1317–1331, 2018.

[46] T. Mühlbauer, W. Rödiger, R. Seilbeck, A. Reiser, A. Kemper, and T. Neumann. Instant loading for main memory databases. *PVLDB*, 6(14):1702–1713, 2013.

[47] S. Palkar, F. Abuzaid, P. Bailis, and M. Zaharia. Filter before you parse: faster analytics on raw data with sparser. *PVLDB*, 11(11):1576–1589, 2018.

[48] O. Polychroniou, A. Raghavan, and K. A. Ross. Rethinking simd vectorization for in-memory databases. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1493–1508. ACM, 2015.

[49] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, et al. A reconfigurable fabric for accelerating large-scale datacenter services. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pages 13–24. IEEE, 2014.

[50] D. Sidler, Z. István, M. Owaida, and G. Alonso. Accelerating pattern matching queries in hybrid cpu-fpga architectures. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 403–415. ACM, 2017.

[51] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, et al. C-store: a column-oriented dbms. *PVLDB*, pages 553–564, 2005.

[52] T. Thanh-Hoang, A. Shambayati, and A. A. Chien. A data layout transformation (dlt) accelerator: Architectural support for data movement optimization in accelerated-centric heterogeneous systems. In *2016 Design, Automation, and Test in Europe Conference and Exhibition (DATE)*. IEEE, Mar. 2016.

[53] B. Wheeler. Titan ic floats 100gbps reg-ex engine. `https://www.linleygroup.com/newsletters/newsletter_detail.php?num=5924&year=2018&tag=3`.

[54] H. Wong, V. Betz, and J. Rose. Comparing fpga vs. custom cmos and the impact on processor microarchitecture. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pages 5–14. ACM, 2011.

[55] L. Wu, R. J. Barker, M. A. Kim, and K. A. Ross. Navigating big data with high-throughput, energy-efficient data partitioning. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, pages 249–260. ACM, 2013.

[56] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross. Q100: The architecture and design of a database processing unit. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, Mar. 2014.

[57] W. Xu, Z. Feng, and E. Lo. Fast multi-column sorting in main-memory column-stores. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1263–1278. ACM, 2016.

[58] Y.-H. Yang and V. Prasanna. High-performance and compact architecture for regular expression matching on fpga. *IEEE Trans. Comput.*, 61(7), July 2012.

[59] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma,

M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[60] F. Zhang, J. Zhai, X. Shen, O. Mutlu, and W. Chen. Efficient document analytics on compressed data: Method, challenges, algorithms, insights. *PVLDB*, 11(11):1522–1535, 2018.

[61] M. Zukowski, N. Nes, and P. Boncz. Dsm vs. nsm: Cpu performance tradeoffs in block-oriented query processing. In *Proceedings of the 4th international workshop on Data management on new hardware*, pages 47–54. ACM, 2008.