SKYSHARK: A Benchmark with Real-world Data for Line-rate Stream Processing with FPGAs

Maximilian Langohr[1], Tim Vogler[1] and Klaus Meyer-Wegener[1]

[1]*Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Martensstraße 3, 91058 Erlangen, Chair of Computer Science 6 (Data Management)*

Abstract

To test and evaluate a heterogeneous stream-processing system consisting of an FPGA-based system-on-chip and a host, we develop a benchmark called SKYSHARK. It uses real-world data from air-traffic control that is publicly available. These data are enhanced for the purpose of the benchmark without changing their characteristics. They are further enriched with aircraft and airport data. We define 14 queries with respect to the particular requirements of our system. They should be useful for other hardware-accelerated platforms as well. A first evaluation has been done using Apache Flink. We envision a great potential because of the flexibility of the approach.

Keywords

benchmark, stream-processing-system, FPGA, hardware acceleration, real-world data, air-traffic control

1. Introduction

The ReProVide project [1], which is part of the DFG Priority Program SPP 2037[1] "Scalable Data Management for Future Hardware", engages an FPGA-based system-on-chip (SoC) to accelerate database queries that analyze large data sets. In order to test the system, queries from the well-known database benchmarks like TPC-DS have been used [2]. In the second phase of the project, however, the scope has been extended to include stream-processing queries as well. Here, the field of benchmarks is more diverse. We selected queries from the Yahoo Streaming Benchmark [3] and from RIoTBench [4] in some evaluations, but they did not match the specifics of our accelerator well enough. While the processing can be done at line-rate, some of the stream-query operators (e. g., sort, join) cannot be implemented easily on an FPGA-based system and thus must be executed on the host system that runs a full stream-processing system (SPS). On the other hand, some operators can be integrated into a single accelerator on the FPGA, e. g., parsing, projection, and filter. The need to transfer data between the SoC and the host calls for early filtering. So we need a number of queries on data streams that are suitable for such a combined, heterogeneous SPS. Modifying one of the existing benchmarks turned out to be more effort than to define a new and perfectly matching one from scratch.

LWDA'23: Lernen, Wissen, Daten, Analysen. October 09–11, 2023, Marburg, Germany

Maximilian.Langohr@fau.de (M. Langohr); Tim.Vogler@fau.de (T. Vogler); Klaus.Meyer-Wegener@fau.de (K. Meyer-Wegener)
https://www6.cs.fau.de/msl (M. Langohr); https://www6.cs.fau.de/kmw (K. Meyer-Wegener)
0000-0002-2520-7134 (M. Langohr); 0000-0002-8102-1019 (K. Meyer-Wegener)

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

This nourished the idea to use public flight data and utilize the queries that are already in use to analyze and visualize them. Hence, we designed a benchmark called SKYSHARK.

1.1. Real-Time Aircraft Tracking Data

Benchmarks face a significant challenge in obtaining sufficient and realistic data for their queries. Existing benchmarks often rely on synthetic data generated through specialized tools, which may not accurately represent real-world conditions. Access to authentic production data is limited due to concerns over data privacy and trade secrets. SKYSHARK offers a promising solution by leveraging publicly available real-world data from the aviation domain. This domain provides a wealth of data as nearly every aircraft continuously broadcasts its position, speed, altitude, and more. It allows for thorough data collection and evaluation. The data is not encrypted and can be received using software-defined radios or purpose-built receivers. Commercial providers collect this type of data from official sources like the Federal Aviation Administration (FAA), Eurocontrol and their own network of ADS-B receivers. Additionally, open-source projects like OpenSky Network (OSN) [5] rely on hobbyists to provide receivers, enabling the collection of tracking information. OSN has been developed by researchers for researchers, providing access to both historic and real-time tracking data. These data are primarily intended for research purposes in the aviation domain but are also publicly accessible for anyone to use.

1.2. Contribution

Recognizing the opportunity presented by this data-rich environment, we created a new benchmark called SKYSHARK to leverage the real-world data for SPSs. It aims to overcome the limitations of synthetic data and to provide a more accurate representation of real-world conditions. While the first implementation has been done in the context of a Master's Thesis [6], it has been improved substantially since then.

2. Related Work

Many benchmarks have been proposed and implemented for stream processing. A very good overview is given in [7]. The purpose of such a benchmark can be different. Most common is the comparison of different systems. But some are also used to test a given system (which has been our original motivation) or to evaluate particular configuration settings. The early ones like Linear Road [8] and NexMark [9] are no longer used because they are too simple for today's SPSs. More recently, the *Yahoo Streaming Benchmark* (YSB) [10] has been engaged by many projects. The stream data consists of reactions on ads on the Web. There is only one stream query composed from filter, projection, static join, and a time-based window. The windows together with counts and a timestamp are stored in a database. The original paper reports results on Flink, Storm, and Spark. A follow-up paper [3] shows different results for Flink. The benchmark has just one query, with a precise definition only by code. The stream data are synthetic. Some of the specifics of our system are not addressed by this query, e. g., the option of using arithmetics. Karimov et al. [11] propose a benchmarking framework called *Rovio*. It is similar to the YSB, trying to improve upon its criticism.

The benchmark *RIoTBench* is much larger than the YSB [4, 12]. Queries (called applications) are defined as data-flow graphs, with tasks (operations) as nodes and edges for transporting messages between them. Tasks are given as the standard operations on streams. Unfortunately, the semantics are not precisely defined, one has to look at the implementation in GitHub. Streams are given with an input rate (messages per second), a distribution, which can be uniform, in bursts, sawtooth, normal, and bi-modal (e. g., day and night), and a message size. The benchmark first offers some micro-benchmarks, which are single tasks. It further defines IoT applications, which are a combination of tasks. All IoT input streams have elements with sensorId, timestamp, and n sensor values. The following four are provided with the benchmark: Sense Your City (CITY), NYC Taxi cab (TAXI), Energy dataset (GRID), and Mobile Health dataset (FIT). Real data are used here. However, the benchmark is very large. It does not provide queries, but rather whole applications. These cannot be implemented by SPSs alone.

StreamBench [13] provides a much broader set of evaluations in four workload suites: performance, multi-recipient, fault tolerance, and durability. It uses real data as seed and generates the workload by repetition. The 7 benchmarks (queries, or applications) are: identity, sample, projection, grep, wordcount, distinctcount, and statistics. Extensive measurements have been made. It has been planned to put the benchmark on GitHub, but the project there with the same name has different authors and goals. Only the simple queries are suitable for our environment, and they do not address the specific challenges of our system.

The approach that is closest to our ideas is *DSPBench* [7]. A low-level API is provided for unified development, so the workload can run anywhere (assuming an appropriate adapter that maps to the SPS-specific code). So-called probers inject timestamps in the tuples to track the latency, to count the number of received and sent tuples of an operator, to measure the time required for processing one tuple in an operator, and to calculate its throughput. Workload characterization is done by performance measurement instrumentation and source code analysis. 15 applications are defined as graphs. The benchmark uses real-world datasets. They are repeated if necessary, but with modifications. Some geographic calculations are also included. The benchmarked system runs on homogeneous nodes, while we have special operators running only on the FPGA. The semantics of the API is not given, but the code is available at GitHub. The timestamps are added to the tuples, which we tried to avoid. It is not clear which overhead is introduced by the measurements.

So while very good solutions are available already, we still felt a need to create another benchmark with the following characteristics: We insisted on using real data. Some extrapolations had to be done due to legal issues, but they do not change the characteristics. We have a larger set of queries, and they have been selected from existing applications w. r. t. line-rate processing on dedicated hardware. Queries are given in standard SQL (if possible) to define precise semantics. The benchmark is extensible: Other data can be downloaded, and new queries can be defined. (This may not be suitable for a benchmark, but definitely for testing specific systems.) Since the data are stored, a DMBS (Flight Schedule, Airports, Aircrafts) can also be evaluated. While all benchmarks so far are designed for homogeneous SPSs, either on a single node or distributed, our system is heterogeneous, consisting of one or more SoCs and a host.

It is our hope that other projects on FPGA-based stream processing can also benefit from our benchmark. There are a number of them; we just cite [14] and [15] as examples.

3. SKYSHARK Reference Data Set

As mentioned earlier, one of our goals is to use data as realistic as possible for our benchmark. Unfortunately, organizations are generally reluctant to share precise data due to concerns over potential data-privacy breaches or loss of trade secrets. In this regard, SKYSHARK emerges as a viable solution by leveraging publicly available real-world data, which can be collected by anyone. This section provides an overview of key concepts and terminology related to Air Traffic Control (ATC). ATC is vital for ensuring safe and efficient air travel.

3.1. Technical Background

3.1.1. Radar

Primary Surveillance Radar (PSR) uses radio signals to detect objects by analyzing the delay and direction of the reflected signals [16]. However, it cannot identify individual aircrafts and only displays them as blips on the radar screen. To address this limitation, Secondary Surveillance Radar (SSR) has been developed. It transmits an interrogation signal to aircrafts, which then respond with key information such as their unique identification code (ICAO[2]), altitude, and squawk code (see 3.1.3). The SSR ground station combines this information with PSR data to provide a comprehensive radar display. Automatic Dependent Surveillance-Broadcast (ADS-B), a modern form of SSR, autonomously broadcasts aircraft information without being interrogated. It gathers data like position and velocity from internal systems such as GPS. ADS-B communication is not encrypted or authenticated, allowing even small general aviation aircrafts and others to receive these signals. This open communication protocol enhances safety and situational awareness for all aircrafts.

3.1.2. Aircraft Tracking Networks

Currently, there are several commercial flight-tracking services available, with Flightradar24[2] being the most popular of them. It offers real-time tracking, access to flight schedules, historical data, and detailed aircraft information. Other commercial networks like FlightAware[3], RadarBox[4], and ADS-B Exchange[5] provide similar functionalities. Alongside with the commercial networks, the OpenSky Network (OSN) [5] is a nonprofit research network. It plays a significant role in the benchmark. OSN has a network of over 1,800 receivers and has collected millions of MODE-S messages[6] from hundreds of thousands of aircrafts. While commercial networks often filter data, OSN provides unfiltered raw access to MODE-S/ADS-B messages received by its receivers. Initially covering the South of Germany and Switzerland, OSN has expanded to include the United States and Europe, but significant gaps still exist in other parts of the world.

[2]https://www.icao.int/
[2]https://www.flightradar24.com
[3]https://de.flightaware.com/
[4]https://www.radarbox.com/
[5]https://globe.adsbexchange.com/
[6]MODE-S is a transponder standard, where each aircraft is broadcasting its 24-bit ICAO address.

3.1.3. Aircraft Tracking Data

To facilitate easier access to ADS-B data for researchers, an abstraction called *state* has been introduced. A state represents an ADS-B message from an aircraft. The following attributes are particularly important for the benchmark:

- **icao24:** Unique 24-bit ICAO identifier. Identifies the aircraft.
- **time_position:** Unix timestamp (seconds) of the last position update.
- **latitude and longitude:** Geographic coordinates of the aircraft's position.
- **baro_altitude:** Barometric altitude in meters. Can be null.
- **velocity:** Speed of the aircraft, measured in meters per second.
- **squawk:** Special transponder code, used as ID for ATC or to indicate emergencies.

3.1.4. Additional Relational Meta-Data

To facilitate the integration of real-time tracking data with relational (batch) data, we focus on two additional sources of data: a database containing information about all aircrafts, and a database containing information about all airports. It is important to note that the data we utilize for this purpose are publicly available and sourced from open source communities.

Aircraft Database: The OSN maintains a comprehensive list of approx. 580,000 aircrafts belonging to 1,800 airlines, as of March 2023. It is important to note that due to the coverage limitations discussed, this list may not be complete. However, since our aircraft-tracking information is sourced from the same database, most states should have a corresponding aircraft entry in the database. The list is periodically exported as a CSV file, which is made available for download on the OSN dataset website.

Airport Database: Another valuable resource is OurAirports[7], a community-maintained open-data website that manages a database of over 5,200 airports as of March 2023. This extensive collection encompasses airports of various sizes and types, ranging from major international airports to smaller seaplane or helicopter bases. The airport database can be downloaded in CSV format. It is worth noting that community-maintained databases may not always be up-to-date. However, this is acceptable for our purposes as we do not need absolute accuracy.

3.2. Obtaining Real-Time Aircraft Tracking Data

The reference dataset that we have created by collecting and processing data from the OSN extends over a two-week period. OSN provides various data-retrieval options, including a REST API, Java and Python packages, and an Impala Shell. Access to the REST API is available both anonymously and as a registered user, with different credit-based mechanisms in place to prevent abuse. Anonymous users have a daily budget of 400 credits, registered users have a budget of 4,000 credits, and users with their own ADS-B receiver enjoy a larger budget of 8,000 credits. Within these limitations, we have been able to collect a comprehensive dataset using

[7]https://ourairports.com/

the SKYSHARK downloader. It allows users to gather a specified amount of data. Depending on the available credits, the downloader calculates the optimal interval between two API requests to achieve evenly spaced states. If users have an account and a daily budget of 4,000 credits, they can make 1,000 API requests per day. Distributing these 1,000 requests evenly over 24 hours would result in an average of approximately one request sent to OSN every 87 seconds. The states are stored as CSV files. The downloader can then convert the CSV file to a desired file format, e. g., JSON.

3.3. Generating Flight Schedules

Obtaining airline flight plans without significant financial investment is not feasible due to the high cost associated with commercial services. To overcome this issue, we create our own flight schedules by querying individual flights using the REST API provided by OSN. We first retrieve the callsigns which consist of an airline abbreviation (e. g., DLH for Lufthansa) and a flight number (e. g., 101). Private or military aircrafts may not have callsigns or their meaning is unknown. We then use the OSN REST API and the callsigns to retrieve flight information such as origin and destination airports. Not all flights with flight numbers have flights recorded in OSN. We determine flight takeoff and landing by monitoring the *on_ground* field in each state. Changes from true to false indicate takeoff, and vice versa indicate landing. By storing and rounding the timestamps of these transitions, we obtain a rough flight schedule that aligns with the recorded flights. It is important to note that this flight schedule is incomplete and not entirely accurate but serves the purpose of the benchmark.

4. Benchmark Design and Implementation

In this chapter, we provide an overview of the general design of the SKYSHARK benchmark. To accomplish this, we will first examine the benchmarking tool. Following that, we will present the benchmark metrics and the 14 SKYSHARK queries. As examples, we will discuss our considerations and constraints in designing two of the queries.

4.1. Benchmarking Tool

The centerpiece of our benchmark is our custom-developed benchmarking tool. This tool allows to send the collected states to the SPS under test and to receive the resulting data. Additionally, it gathers various metrics that serve the purpose of comparing different systems and implementations with each other. In Fig. 1, a schematic representation of our benchmarking tool is shown, and how it can be integrated with an SPS. Our tool provides a set of well-defined interfaces for connecting with the SPS. These interfaces include TCP, UDP, as well as adapters for Apache Kafka[3] and ZeroMQ[4]. When designing the benchmark, we decided not to measure the individual operators of the SPS but rather focus on the system as a whole. This decision was motivated by our intention to create a benchmark for systems that utilize modern hardware, where such operator-level measurements may not be feasible. Considering

[3]https://kafka.apache.org/
[4]https://zeromq.org/

the expected data streams in the area of 10Gbps, measuring the individual operators would impose an additional burden on the system. Generating measurement data at the operator level, which would then need to be processed by our system, would introduce additional CPU, RAM, and potentially network overhead. This additional load would impact the performance of the SPS. The benchmarking tool loads the data from the drive and generates a data stream. Before

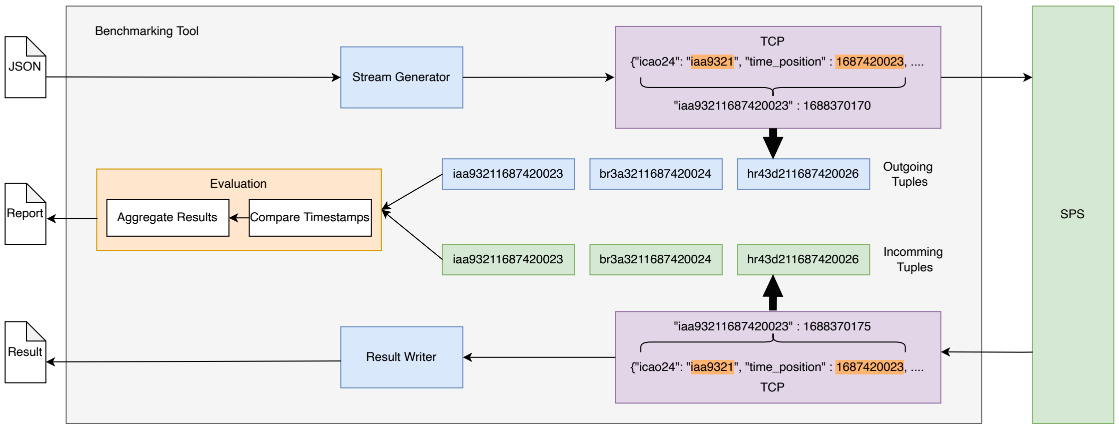

Figure 1: Architecture of the SKYSHARK Benchmarking Tool

sending to the SPS, a hash is computed using the *icao24* and the *time_position* of the state. This key, along with a timestamp, is stored internally. Once the tuple reaches the benchmarking tool again, another hash is computed and stored with a timestamp. The result tuple is then saved to disk. In a process running concurrently with sending and receiving, the latencies of each tuple are calculated and stored for further analysis. Since some queries involve filtering tuples based on certain conditions, we cannot calculate the latency for all tuples. The calculation of latencies is only possible for queries that do not involve blocking operations, as this would disrupt the relationship between input and output. For these queries, only the throughput can be measured.

4.2. SKYSHARK Metrics

As mentioned in Chapter 2, there is a variety of metrics that can be used for measurement purposes. For SKYSHARK, we specifically focus on two metrics.

End-to-End Latency: End-to-end latency plays a crucial role in the deployment of SPSs. Depending on the application the latency can have immense impact on the usability of the result. Therefore, we have identified this metric as an important measurement for SKYSHARK.

Throughput: Another significant metric in the field of stream processing is throughput. With our benchmarking tool, we can measure it with high precision, as we have full control over the incoming and outgoing tuples. Furthermore, using the aforementioned map where keys and timestamps are stored, we can measure the number of tuples that have been discarded during the process.

No.	Query	Query Focus	Latency
1	Identity	Test query for implementations	Yes
2	JSON Parsing	JSON decoding/encoding performance	Yes
3	Speed Conversion	Arith. expressions/renaming/rearranging	Yes
4	Aircraft Conversion	Filter with string equals	Yes
5	Emergency Squawk	Filter with Boolean expression (integer)	Yes
6	Emergency Descent	Complex filter with arithmetic expressions	Yes
7	Airline Filter	Filter with string wildcard	Yes
8	Airspace Check	Complex projection, floating-point comparison	Yes
9	Airport Proximity	Complex numeric calculations	Yes
10	Aircraft Information	Join	No
11	Airborn Aircraft State Count	Aggregation, Time windows	No
12	OpenSky Latency	Aggregation, tuple count windows	No
13	Diversion Airports	Join with complex calculations	No
14	Flight Schedule Updating	Complex join, updating RDBMS table	No

Figure 2: List of SKYSHARK Queries

4.3. SKYSHARK Queries

In addition to the data and the benchmarking tool, queries play a particularly important role in SKYSHARK. We have closely aligned the design of the queries with real-world aviation and airspace surveillance challenges. A total of 14 queries has been identified. These queries progressively increase in complexity and in demand from the system being measured. We have chosen to code the queries in standard SQL or, if that was impossible, in the extension of SQL developed by Apache Calcite. This extension allows for the definition of stream queries, where no standard is available yet. The goal was to provide a precise and formal description to eliminate misinterpretation. The actual code of the queries can be different in a particular SPS. The table in Fig. 2 presents a list of all queries, their intended focus, and an indication of whether latencies can be measured for these queries. A detailed listing of all queries, including their code and intention, can be found on our website[5]. To further illustrate our design decisions we present two exemplary queries.

4.3.1. Using Basic Arithmetic to Solve Complex Problems

Query 9 of the SKYSHARK benchmark (refer to Fig. 3) exemplifies the trade-offs we made in the query design process. One specific requirement from the ReProVide project is that only basic arithmetic operations can be computed on the FPGA. The purpose of this query is to filter out aircrafts within a certain radius around the airport. Ideally, the Euclidean distance would be calculated, which requires the use of trigonometric functions. However, due to the FPGA's limitations in performing these calculations, a decision was made to approximate the distance. The approximation method used assumes that the circumference of the Earth is the

[5]https://skyshark.org

```
--airport_lat = 50.036521
--airport_long = 8.561268
SELECT STREAM icao24 ,callsign ,time_position ,latitude ,longitude ,baro_altitude ,
 ((airport_lat -(state_lat))*(airport_lat -(state_lat)))
 +((airport_long -(state_long))*(airport_long -(state_long))) as distance_squared
FROM (SELECT icao24 ,callsign ,time_position ,latitude ,longitude ,baro_altitude ,
     (latitude*111.139) AS state_lat ,
     ((1001.879 / 9)*(0.0129281*longitude+1.2)*longitude) AS state_long ,
     (airport_latitude * 111.139) AS airport_lat ,
     ((1001.879 / 9)*(0.0129281*airport_long+1.2)*airport_long) AS airport_long
FROM states) as position
WHERE {copy distance_squared from SELECT}<625
```

Figure 3: Query 9 – Airport Proximity

same everywhere, allowing the cosine calculation to be replaced with a linear function. This approximation introduces a decrease in accuracy at the poles and near the equator. Nonetheless, for our specific purposes, this is acceptable as long as the selected airport is not located in these boundary areas. We believe that these types of calculations are better suited for offloading to FPGAs or other modern hardware, which are capable of handling more complex computations efficiently.

4.3.2. Joining Stream Data and Relational Data

The second query we take a closer look at is Query 13 (see Fig. 4, “Diversion Airports”). In the event of an emergency, the pilot needs to know which airports are closest to the aircraft at all times. These airports can be used for an emergency landing if necessary. In this query, we not only have the complexity of distance calculation, but also involve joining each incoming tuple with the airport relation. Additionally, the join condition poses a particularly complex aspect. The topic of joining data on FPGAs is gaining popularity [17, 18], making such a query a great candidate for exploration in this area.

5. Evaluation

To test the functionality of our benchmarking tool, we have implemented several queries from the benchmark in Apache Flink. The measurements have been performed on a PC with an AMD Ryzen 9 5900X 12-core processor and 32GB DDR4 RAM. Throughout the testing, all queries were able to run at a throughput of 10Gbps. For Query 2, the average latency measured was 100ms, with occasional spikes reaching 180ms or more. Fig. 5 provides a visual representation of the latencies caused by Apache Flink. It is important to emphasize that these measured numbers are preliminary and require further refinement. A thorough evaluation of the feasibility of our benchmark is ongoing work. The hardware with the FPGA should be available within a few weeks.

```
SELECT STREAM icao24 ,callsign ,time_position ,ident as airport ,
((airports_lat -(state_lat))*(airports_lat -(state_lat)))
+((airports_long -(states_long))*(airports_long -(states_long)))
 as distance_to_airport_squared
 FROM ( SELECT states.icao24 ,states.callsign ,states.time_position ,
     (states.latitude*111.139) AS state_latitude ,
     ((1001.879/9)*(0.0129281*states.longitude+1.2)*states.longitude)
     AS states_long_km
     FROM states WHERE NOT states.on_ground ) as state_position
JOIN ( SELECT airports.ident ,airports.type ,
     (airports.latitude * 111.139) as airports_lat ,
     ((1001.879/9)*(0.0129281 * airports_long + 1.2)*airports.longitude) AS airports_long
        FROM airports WHERE airports.type like 'large_airport') as airport_position
 ON {copy distance_to_airport_squared from SELECT }<62500
```

Figure 4: Query 13 – Diversion Airports

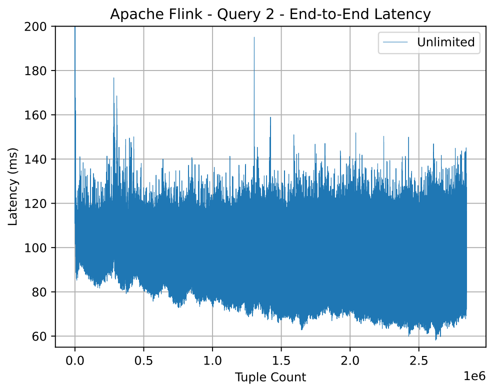

Figure 5: Latencies of Query 2 with Apache Flink

6. Future Work

With SKYSHARK, we have taken the first step in using real-time aircraft-tracking data as a source for benchmarks. Focusing on SPSs, we were mainly inspired by our project ReProVide and the FPGA used in it. Our next step is to extensively measure the prototype we developed as part of ReProVide using our new benchmark. Additionally, we want to complete and measure our reference implementation in Apache Flink as well, to compare our system with a homogeneous SPS. Additional queries could be identified, e. g., using match-recognize as proposed in [19].

Acknowledgments

This work has been supported by the German Science Foundation (Deutsche Forschungsgemeinschaft, DFG) as a part of SPP 2037 with the grant no. ME 943/9-2. We thank the reviewers for their valuable hints!

References

[1] A. Becher, A. Herrmann, S. Wildermann, J. Teich, ReProVide: Towards utilizing heterogeneous partially reconfigurable architectures for near-memory data processing, in: Proc. BTW – Workshopband, Gesellschaft für Informatik, Bonn, 2019, pp. 51–70.

[2] L. Beena Gopalakrishnan Nair, A. Becher, K. Meyer-Wegener, S. Wildermann, J. Teich, SQL query processing using an integrated FPGA-based near-data accelerator in ReProVide (demo paper), in: Proc. EDBT, 2020, pp. 639–642.

[3] J. Grier, Extending the Yahoo! Streaming Benchmark, 2016. URL: https://www.ververica.com/blog/extending-the-yahoo-streaming-benchmark, accessed on June 14, 2023.

[4] A. Shukla, S. Chaturvedi, Y. Simmhan, RIoTBench: A real-time IoT benchmark for distributed stream processing platforms, CoRR abs/1701.08530 (2017).

[5] M. Schäfer, M. Strohmeier, V. Lenders, I. Martinovic, M. Wilhelm, Bringing up OpenSky: A large-scale ADS-B sensor network for research, in: Proc. 13th Int. Symp. on Information Processing in Sensor Networks (IPSN), 2014, pp. 83–94.

[6] T. Vogler, Development and Implementation of a Database-Benchmark Using Real-Time Flight Data (ADS-B) and Flight Schedules, Master's thesis, FAU, 2023.

[7] M. V. Bordin, D. Griebler, G. Mencagli, C. F. R. Geyer, L. G. L. Fernandes, DSPBench: A suite of benchmark applications for distributed data stream processing systems, IEEE Access 8 (2020) 222900–222917.

[8] A. Arasu, M. Cherniack, E. F. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker, R. Tibbetts, Linear road: A stream data management benchmark, in: Proc. VLDB, 2004, pp. 480–491. URL: http://www.vldb.org/conf/2004/RS12P1.PDF.

[9] NEXMark, Nexmark benchmark, URL: http://datalab.cs.pdx.edu/niagara/NEXMark/, 2002. Last visited: June 15, 2023.

[10] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, P. Poulosky, Benchmarking streaming computation engines at Yahoo!, Online, yahoo! developer blogs, 2015. URL: https://developer.yahoo.com/blogs/135370591481/.

[11] J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen, V. Markl, Benchmarking distributed stream data processing systems, in: Proc. ICDE, 2018, pp. 1507–1518.

[12] A. Shukla, S. Chaturvedi, Y. Simmhan, RIoTBench: An IoT benchmark for distributed stream processing systems, Concurrency and Computation: Practice and Experience 29 (2017) e4257.

[13] R. Lu, G. Wu, B. Xie, J. Hu, StreamBench: Towards benchmarking modern distributed stream computing frameworks, in: Proc. 7th IEEE/ACM Int. Conf. on Utility and Cloud Computing (UCC, London, United Kingdom, Dec. 8-11), IEEE Computer Society, 2014, pp. 69–78.

[14] S. Wu, D. Hu, S. Ibrahim, H. Jin, J. Xiao, F. Chen, H. Liu, When FPGA-Accelerator meets stream data processing in the Edge, in: Proc. ICDCS, 2019, pp. 1818–1829.

[15] M. Najafi, Hardware Accelerated Stream Processing, Ph.D. thesis, TU München, Fakultät für Informatik, 2019.

[16] M. Stevens, Secondary Surveillance Radar, Artech House radar library, Artech House, 1988. URL: https://books.google.de/books?id=qH9TAAAAMAAJ.

[17] M.-T. Xue, Q.-J. Xing, C. Feng, F. Yu, Z.-G. Ma, FPGA-accelerated hash join operation for relational databases, IEEE Transactions on Circuits and Systems II: Express Briefs 67 (2020) 1919–1923.

[18] B. Salami, O. Arcas-Abella, N. Sonmez, O. Unsal, A. C. Kestelman, Accelerating hash-based query processing operations on FPGAs by a hash table caching technique, in: Communications in Computer and Information Science, Springer International Publishing, 2017, pp. 131–145.

[19] C. Beilschmidt, J. Drönner, N. Glombiewski, C. Heigele, J. Holznigenkemper, A. Isenberg, M. Körber, M. Mattig, A. Morgen, B. Seeger, Pretty fly for a VAT GUI: visualizing event patterns for flight data, in: Proc. DEBS, 2019, pp. 224–227.

A. Online Resources

A description of the benchmark is available at https://skyshark.org.