High-throughput conversion of Apache Parquet files to Apache Arrow in-memory format using FPGA's

Lars T.J. van Leeuwen, Johan Peltenburg, Jian Fang, Zaid Al-Ars
Delft University of Technology
Delft, The Netherlands
{l.t.j.vanleeuwen@student., j.w.peltenburg@, j.fang-1@,
z.al-ars@}tudelft.nl

Peter Hofstee
IBM Austin & TU Delft
Austin, USA
h.p.hofstee@tudelft.nl

***Abstract*— Recently, persistent storage bandwidth has increased tremendously due to the use of flash technology. In the domain of big data analytics, the bottleneck of converting storage focused file formats to in-memory data structures has shifted from the storage technology to the software components that are tasked with decompression and organization of the data in memory. One commonly used file format is Apache Parquet, and a recently developed in-memory format is Apache Arrow. In order to improve the bandwidth at which such conversions take place, we propose a Parquet-to-Arrow converter implemented in an FPGA. The design is modular and extendable to support different Parquet formats. The resource utilization of the Xilinx XCVU9P device used for the prototype is 4.16% of CLBs and 1.78% BRAMs, leaving ample room to implement analytical kernels that operate in tandem with the file conversion. The prototype shows promising throughput for converting the basic structure of Parquet files with large page sizes, with the throughput being limited by the bandwidth of the connection to device memory.**

I. Introduction

With the arrival of NVMe SSD's, the bandwidth associated with reading data from persistent storage is increasing rapidly. If the bandwidth of persistent storage is no longer a bottleneck, conversion of storage focused formats to data structures usable in memory risks becoming a limiting factor in database systems. In order to improve the performance of database systems we propose performing this conversion on an FPGA. We present a framework that takes Apache Parquet [1] pages as input and creates Apache Arrow [2] format data structures in memory using the Fletcher [3] framework.

II. Background

Parquet is a columnar storage format that supports multiple compression and encoding schemes for stored data [1]. With Parquet's columns divided into individually compressed and encoded pages, analytics applications can benefit from columnar data while still allowing for smaller scale accesses without having to decompress and decode the whole file.

Arrow is a columnar format that is focused on efficient in-memory representation of data [2]. Like Parquet, its columnar format allows fast vectorized operations with the aim of preventing data copies or serialization between different language run-times through shared memory pools.

Fletcher is the framework that allows FPGA's access to Arrow data with a fast and easy to use interface [3]. Instead of byte addresses only column indices are required to read and write data in Arrow format. Fletcher hardware is generated based on a schema describing the data set stored in Arrow format.

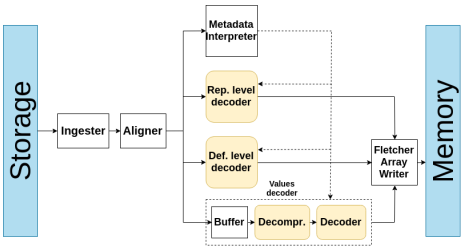

Fig. 1. High-level architecture of the hardware

III. Design

A. High-level architecture

With Parquet and Arrow both being columnar formats the hardware can read consecutive Parquet pages from memory, interpret the page headers, and perform decompression and decoding steps on the page data without having to do significant data reordering.

A Parquet page consists of four distinct, variable-length blocks of data. The header, the repetition levels, the definition levels, and the actual values. The header contains information on (among other things) the size of the following three blocks in the page and the number of entries in the page. The repetition levels and definition levels encode the structure (in case of nested data structures) and nulls in the page respectively according to the algorithm described in Google's Dremel paper [4]. These levels are not encoded in the case of non-nested or non-nullable data. Finally the compressed and encoded values complete the page.

There are three requirements for the design. First, the design should be modular and expandable to enable acceleration of the many different schemes Parquet supports. Second, the design should be area-efficient in order to fit many instances on an FPGA for parallel workloads, or leave room for analytical kernels. Third, the high-level architecture should maximize throughput so any decompressors and decoders can be fed data at a rate close to system bandwidth (e.g. PCIe bandwidth).

The proposed high-level architecture is seen in Figure 1. The aligner ensures that each of the modules responsible for reading one of the four main blocks of data in a Parquet page in turn receives their data correctly aligned. The

aligner requires these modules to report back the amount of bytes they used after they are finished. The rounded blocks are replaceable or omissible modules to enable support for different compression and encoding schemes.

Although Figure 1 shows the decoder directly streaming the decoded values into Fletcher for immediate writing to memory this does not have to be the case. Any hardware supporting Fletcher style streams can be inserted in between to allow for operations on the data (e.g. maps, filters, etc.) before it is written to memory, allowing for optimal use of the FPGA's resources in accelerating any data analytics application starting from a Parquet file.

B. Decompression

The first decompressor being integrated with the Parquet to Arrow converter is a Snappy decompressor implemented in previous work [5]. Preliminary benchmarks show a throughput of 3GB/s input and 5GB/s output for Snappy compressed files for a single instance of the decompressor. This is an order of magnitude higher than a single thread on a Core i7 processor.

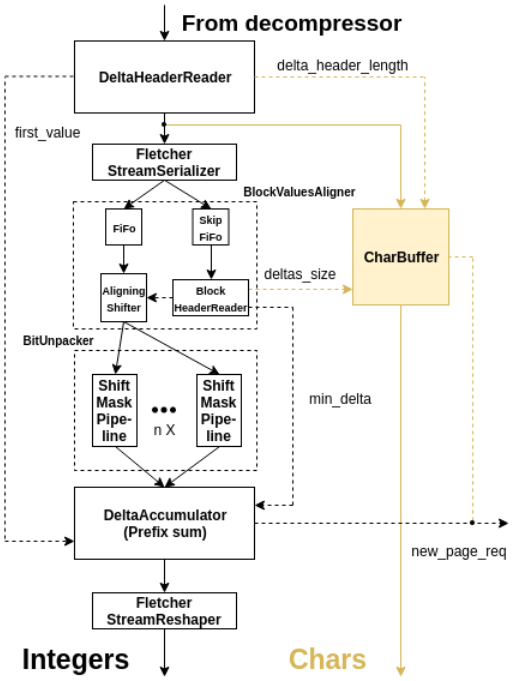

Fig. 2. Delta decoder

C. Decoding

In order to make the Parquet to Arrow converter usable out of the box, decoders will be included that can decode at least one encoding for floats, doubles, 32 and 64 bit integers, and strings. Because of the prevalence of delta encoding in Parquet (for integers and string lengths) a delta decoder as seen in fig. 2 is proposed. In delta decoding the values are encoded as variable-width bit-packed deltas with respect to the previous value. After narrowing the bit-width of the stream from the decompressor for easier manipulation using a serializer, the bit-packed values will be correctly aligned based on the bit-packing width and block header length data. Hereafter, the values can be unpacked in a separate shift and mask pipeline for each value in the aligned data. These values will be added to the minimum delta received from the BlockHeaderReader to create the final deltas that can be added to the previously decoded integers in the DeltaAccumulator.

A buffer for the character data can be added to make this decoder work for strings as the raw character data directly follows the encoded string lengths.

IV. Preliminary Results

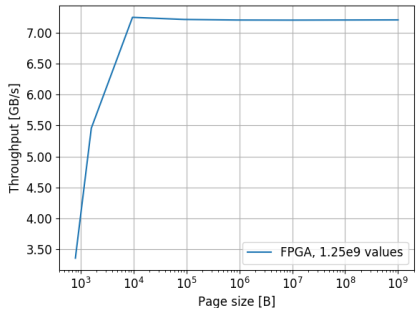

Fig. 3. Throughput for different page sizes

To establish the overhead of processing the Parquet page headers, an implementation was made to convert simple (uncompressed, non-nullable, plainly encoded) Parquet files containing only a column of 64 bit integers on an AWS f1.2xlarge instance with a Xilinx Ultrascale+ FPGA. The resulting throughput is seen in fig. 3. This implementation required only 4.16% of CLB's and 1.78% of BRAM while timing at 250MHz. For files containing small Parquet pages the conversion rate is limited by the latency of reading the page headers. This effect stops being significant at 100kB pages, after which it starts being limited by the bandwidth of the connection to device memory. This suggests decompression and decoding modules can make full use of the read/write bandwidth of the FPGA if large enough pages are used in the Parquet file.

V. Conclusion

Preliminary results show that the converter can process the basic structure of a Parquet file at a high throughput. Continuing work is focused on implementing decompression and decoding modules that make full use of the strengths of an FPGA to create Arrow format data from a Parquet file significantly faster than a general processor. Through the use of the proposed converter, big data analytics applications may alleviate the software bottlenecks resulting from decompression and conversion overhead of the Parquet storage format.

References

[1] Apache. Apache Parquet. [Online]. Available: https://parquet.apache.org/ (Accessed 2019-04-29).

[2] Apache. Apache Arrow. [Online]. Available: https://arrow.apache.org/ (Accessed 2019-04-29).

[3] J. Peltenburg, J. van Straten, M. Brobbel, H. P. Hofstee, and Z. Al-Ars, "Supporting columnar in-memory formats on fpga: The hardware design of fletcher for apache arrow," in *Applied Reconfigurable Computing*, C. Hochberger, B. Nelson, A. Koch, R. Woods, and P. Diniz, Eds. Cham: Springer International Publishing, 2019, pp. 32–47.

[4] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," in *Proc. of the 36th Int'l Conf on Very Large Data Bases*, 2010, pp. 330–339. [Online]. Available: http://www.vldb2010.org/accept.htm

[5] J. Fang, J. Chen, Z. Al-Ars, P. Hofstee, and J. Hidders, "Work-in-progress: A high-bandwidth snappy decompressor in reconfigurable logic," in *2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Sep. 2018, pp. 1–2.