Tackling Hardware/Software co-design from a database perspective

Gustavo Alonso, Timothy Roscoe, David Cock, Mohsen Ewaida,
Kaan Kara, Dario Korolija, David Sidler, Zeke Wang
Systems Group, Dept. of Computer Science, ETH Zurich, Switzerland
first_name.last_name@inf.ethz.ch

ABSTRACT

Hardware is evolving at a very fast pace due to diverse trends in the IT industry. In the area of data processing, it is fair to say that software often just reacts to these changes, trying to accommodate developments that are not always an immediate step forward in terms of either performance or functionality. In this paper we report on two ongoing, long-term projects: Enzian, an experimental hardware platform to explore the design of software systems on future hardware, and doppioDB, a research database engine built to explore how to to co-design hardware and software from a data procesisng perspective. The paper focuses on the possibilities offered by the combination of Enzian+doppioDB in terms of enabling novel data processing systems.

1. INTRODUCTION

Cloud computing and the widespread and increasing use of highly demanding data processing applications are driving major changes on computer and software platforms. On the one hand, cloud computing offers economies of scale and a service model where the user no longer needs to maintain the infrastructure as this is provided and controlled in a highly centralized manner. On the other hand, the growing demand for more computing capacity as well as the increasing computational cost and complexity of data processing operations has shown how inefficient conventional computers can be. These inefficiencies arise from their focus on general purpose computation and lack of support for specialized operations (e.g., large scale floating point arithmetic or optimized data movement). When these two trends are combined, an intriguing scenario arises where specialization from the application all the way down to the underlying hardware becomes feasible as a way to efficiently address application demands.

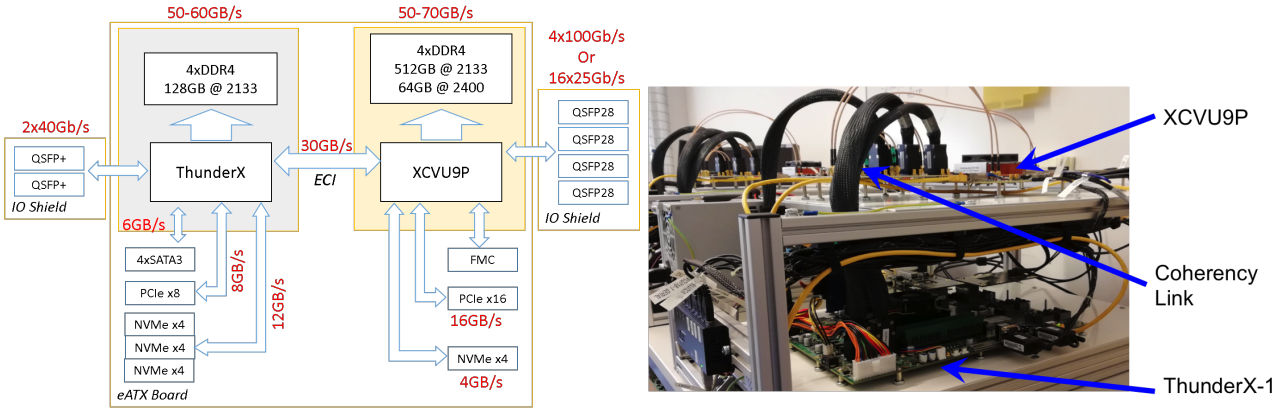
The technological developments in the last years have provided ample proof of the prevalence of such a trend. In almost all cases, the applications involved are related to data processing (for the purposes of this paper, we consider machine learning just another form of data processing). From Google's TPU[1] to Microsoft's Catapult [6] –both projects boosting the cloud infrastructure to provide more efficient services– to the growing availability of user programmable FPGAs in Amazon's[2] or Alibaba's[3] cloud services, there is no lack of examples demonstrating the trend towards specialization and hardware acceleration. Intel's announcements about the embedding of FPGAs into conventional CPUs[4] is an extreme form of the same phenomenon, where the CPU –the heart of the machine– becomes customizable and extensible through the inclusion of reconfigurable logic.

In this paper, we focus our attention on hardware/software co-design from a database perspective and discuss our efforts around building a research hardware platform (Enzian) as well as the development of a database engine built on top of Enzian (doppioDB 3.0). The motivation for Enzian+doppioDB arises from the experience gathered over a decade of exploring hardware acceleration for data processing. An effort often hampered by (1) having to rely on hardware not designed for the purpose and that, consequently, quickly becomes a bottleneck; (2) not having open access to crucial parts of the system and hardware elements, which severely limits the possibilities for tailoring the design; and (3) lacking flexibility in the underlying platforms, making it very difficult to explore different architectural configurations. The main goal of Enzian+doppioDB is to serve as an open source platform (on both the hardware and the software side) to enable researchers to explore a very rich design space and obtain hard evidence of what are the best designs at all levels, from the hardware configuration, to the system stack, to the algorithms involved, and including the applications that can actually benefit from the new hardware.

For reasons of space, we cannot go into many of the details pertaining to design decisions and architecture of DoppioDB+Enzian (the paper will be significantly extended in the final version). Since at this stage of the development we are more interested in enabling functionality than in performance, we focus here on showing how Enzian+DoppioDB open up the possibility to explore designs that maybe have been speculated about in the past but for which no suitable hardware platform was available.

10th Annual Conference on Innovative Data Systems Research (CIDR '20) January 12-15, 2020, Amsterdam, Netherlands.

[1] https://cloud.google.com/tpu/
[2] https://aws.amazon.com/ec2/instance-types/f1/
[3] https://www.alibabacloud.com/help/doc-detail/25378.htm
[4] https://itpeernetwork.intel.com/intel-processors-fpga-better-together



(a) (b)

Figure 1: (a) Architectural schematic of Enzian and (b) photograph of the first operational prototypes

2. ENZIAN

Enzian[5] is a research computer built from standard components and intended to maximize capacity and configuration flexibility. Enzian has not been built explicitly for data processing but to support a wide range of research activities ranging from operating systems and system software to real time software verification and tuning. Nevertheless, Enzian happens to be –also by design- a particularly interesting platform for data processing because its flexibility in terms of how it can be used. To the extent possible, both the software stack on the CPU (Linux based) as well as the entire stack on the FPGA will be released as open source including the architectural blueprint.

The design of Enzian has been driven by the goal of building a platform where hardware bottlenecks are avoided so as to allow exploring future designs that are years out and not just what can be done today. There is no claim for the design to be commercially viable as it is heavily over-provisioned in several areas. Once a particular use case is chosen, there are probably more efficient designs that can be derived from Enzian by removing unnecessary components.

2.1 Hardware

At its core, Enzian is a heterogeneous architecture combining a multi/core CPU and an FPGA (Figure 1.a, which also indicates the bandwidth available between the different elements). The current CPU is an 48 ARM core (Cavium ThunderX v1, with ARMv8-A cores) with 128 GBytes of memory (DDR4 in 4 DIMMs, running at 2133 MHz) and two 40 Gbps network ports. The FPGA is a Xilinx UItrascale (XCVU9P) with four 100 Gbs network ports and 512 GBytes of DRAM (for 2133 MHz DDR4, or 64 GBytes if it is DDR4 2400MHz). The CPU and the FPGA are connected through a CCPI cache-coherent bus (the processor's native cache coherence protocol) providing 30GB/s of bandwidth. Both the CPU and the FPGA have their own PCIe interfaces for additional, optional modules (GPU, Non-Volatile Memory, High Bandwidth Memory, etc.) and the CPU has 4 SATA interfaces to connect storage devices.

Many of the design choices behind Enzian are motivated in part by data processing needs. The over-provisioning of network resources is intended to support distributed data processing and streaming using clusters of CPUs, FPGAs, or combinations of both without the networking becoming a bottleneck as it happens today where the FPGA has access to the network often only through the CPU. Similarly, the current version of Enzian allocates four times more memory to the FPGA than to the CPU, opening up the possibility of reversing the roles of CPU and FPGA where the CPU is treated as a specialized accelerator and the FPGA as the general purpose engine (similar to Systems-On-Chip designs for embedded systems (SoC) but at server scale). Such a design would be very useful in, e.g., data streaming engines.

Enzian shares similarities with several existing systems. Like Intel's Xeon+FPGA designs or IBM's CAPI based systems[6], Enzian uses a cache coherent protocol to connect the FPGA to the CPU. Unlike these systems, Enzian opens up the FPGA side of the cache coherency protocol so that it can be tailored and extended as needed. Cache coherency in accelerators is a broad topic where hard data is often missing and Enzian will be an useful platform to explore the space, from the role and performance of cache coherency in accelerators to questions of scalability of cache coherency, including the potential of extending cache coherency across a pool of FPGAs. Unlike in Enzian, in Intel's design the FPGA has no local memory and no network access. Similarly to Microsoft's Catapult design [4, 6], in Enzian the FPGA is connected to the network. Unlike in Enzian, in Catapult the CPU has access to the network either directly or though the FPGA (by connecting a network port of the CPU to one of the network ports for the FPGA) in addition to the cache coherency protocol. The Microsoft's and Intel's designs are very different but highly complementary. Enzian covers both and can be configured to behave like either of them. By using the same underlying hardware, Enzian is the perfect vehicle to explore the differences and overheads implicit in the two complementary designs. In the case of FPGA instances available in cloud premises (Amazon, Alibaba), the FPGAs are configured as accelerators connected through PCI. There is no cache coherency with the CPU and no network access, making them a strict subset of the other systems in terms of architectural configuration.

[5]http://enzian.systems/

[6]https://developer.ibm.com/linuxonpower/capi/

2.2 Software

In addition to building the hardware platform, we are devoting considerable effort to develop the necessary software on both the CPU and the FPGA side. The most relevant development lines for this paper are the cache coherency protocol, a shell for the FPGA, **LynX**, and networking.

The cache coherency protocol on the FPGA side is an implementation of CCPI. We are aiming for a layered and open design that will enable applications on the FPGA side to dictate the level of cache coherency they need and will allow extending the protocol to tailor it to accommodate a variety of parallel data processing use cases.

LynX is a shell providing operating system like services on the FPGA including multi-threading, time sharing, multi-tenancy, dynamic reconfiguration, interrupts, etc. It also provides a unified memory space so that applications both on the CPU as well as the FPGA side see a single memory space regardless of where the actual physical memory is located. This unified memory space is based on virtual addresses, an innovation that allows Enzian code to deal with pointers and complex data structures in a seamless manner regardless of where the code accessing the data structure resides. In contrast, existing systems restrict how much memory can be accessed from the FPGA and, typically, the FPGA has to work on physical addresses rather than virtual ones, making the interaction between FPGA code and CPU code quite cumbersome. LynX provides a number of additional features such as the equivalent of *pipes* on the FPGA side to support connecting separate processes on the FPGAs as it is done on an operating system, a very efficient dynamic reconfiguration mechanism for different regions of the FPGA (to enable swapping applications in and out of the FPGA), and a number of interfaces for memory access that hide the complexities of a hybrid memory system.

Networking is a key component of Enzian as it can be readily seen from the amount of networking bandwidth and access points that are available. Given the lack of suitable open source alternatives, we have developed two networking stacks for FPGAs that are available in Enzian. One of the stacks is a TCP/IP off-load engine supporting conventional TCP/IP over 10, 40, or 100 Gbs links. Specially at the lower bandwidths, the stack is capable of supporting many thousands of concurrent flows, making it suitable for data center use and virtualization scenarios [14, 13]. The second stack is a RoCE (RDMA over Converged Ethernet) engine supporting RDMA and also working at either 10, 40, or 100 Gbs. In both cases, the stacks are open and provide interfaces for deep packet inspection, content analysis, and smart NIC features through kernels that can manipulate the flow and/or the contents of the packets.

On the software side, there are important differences when comparing Enzian with existing system. The LynX shell is unique in providing multi-threading on the FPGA through the use of **virtual FPGAs** and in supporting a unified memory space tying together the memory on the CPU and the FPGA under a common virtual address space. These two features are important in any system but crucial for a throughput oriented systems and involving large amounts of memory resident data such as databases engines. The open source networking stacks also offer many possibilities for processing packets and off-loading data processing functionality to the NIC, especially for streaming. Enzian has also been designed to be used in clusters for larger capacity.

3. DOPPIODB

DoppioDB is a quickly evolving database engine currently in its third generation (in this paper we will be referring to doppioDB 3.0 unless explicitly stated). Like Enzian, doppioDB is intended as a platform for research and exploration of hardware acceleration in the context of databases in particular and data processing in general, including machine learning. DoppioDB, in its current incarnation, combines MonetDB on the CPU side with an open source FPGA shell providing multi-threading and communication interfaces on the FPGA side. While not the latest database engine, MonetDB is a stable, well documented, easily extensible, full-fledged SQL engine with a reasonable performance. IN the future, though, we will consider other options, especially when addressing streaming and large scale distributed data processing.

DoppioDB has gone through three iterations, each one intended to explore a particular issue of hardware/software co-design in the context of databases. DoppioDB 1.0 implements hardware acceleration for operators such as hashing, skyline, or regular expressions [3], all things the database either does not do at all or does not do well. It is mainly intended as a basis for studying integration issues of the FPGA such as how to provide multi-threading on the FPGA side or data transformation functionality to convert a columnar representation into a row vector based representation suitable for machine learning. It also serves as a way to explore how to add machine learning operators and machine learning models as part of the engine and SQL as it incorporates FPGA modules to run stochastic gradient descent and decision tree ensembles [16]. DoppioDB 2.0 provides a more limited interface between the CPU and the accelerator as it is intended as a vehicle to study the use of custom processors on the FPGA rather than using specialized designs for each operator. It is also the basis for exploring different data representations more suitable for the hybrid architecture of the system than the conventional row/column representations typically used in database engines [17, 11, 12].

DoppioDB 3.0 is the version being built on top of Enzian and taking advantage of the additional hardware features (local memory on the FPGA, networking) and the more versatile LynX shell driving the FPGA. The architecture of DoppioDB 3.0 is still evolving given the many new features that running on top of Enzian enables (see below) and that allow to explore streaming, distributed execution, machine learning extensions, etc. At this stage, it is not clear that a single engine can be built (or that it makes sense to do so) covering all the use cases and possibilities that Enzian has to offer. Nevertheless, the initial intention is to try to unify the development as much as possible so as to allow comparisons across systems. For instance, we expect Enzian to change the performance equation for many operators and applications, enabling operations inside the database engine that are now done outside. Similarly, Enzian is an ideal platform for combining OLTP and OLAP under a single engine, with the added advantage of having the FPGA to implement hardware accelerated analytics and machine learning to boost the capacity of the OLAP side. Finally, streaming will be an important use case for Enzian. It is still open whether it makes more sense to build a separate streaming engine or to use the combined possibilities the hardware offers to build a hybrid relational-streaming engine. This is especially interesting with Enzian in a cluster configuration.

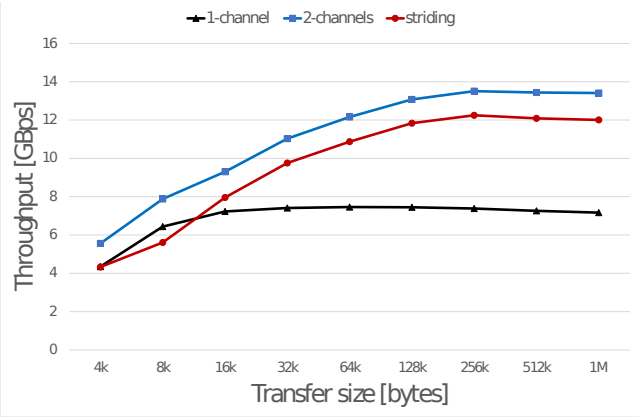


Figure 2: Customized memory controller on the FPGA accessing memory by striding using LynX interfaces

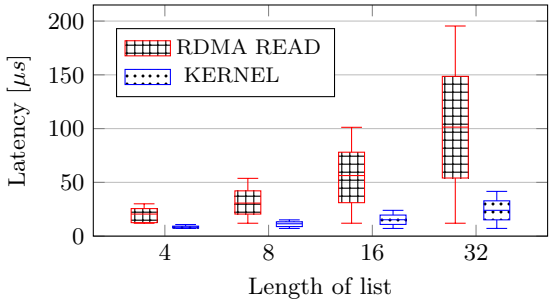


Figure 3: Traversing a remote linked list using conventional RDMA READ versus using the remote FPGA as memory controller imnplementing an RPC interface. Whiskers indicate the 1st and 99th percentile. Value size 64 B

4. EXPLORING A RICH SPACE

4.1 Near Memory Processing

A first intended use of Enzian + DoppioDB is to explore the possibilities resulting from having processing capabilities between the DRAM and the processor. The idea is similar to that behind Oracle's M7 Data Analytics Accelerator (DAX[7]) that inserts a dedicated accelerator along the memory bus near the memory controller units. These accelerators intercept the data as it comes from memory and can filter the data through Bloom filters, perform predicate evaluation, or filter rows by bit vectors. They also contain facilities to compress and decompress data as well as to pack and unpack the data. Similar systems and ideas focused on different use cases, are starting to appear in the literature [5, 1].

In our case, while the general principle would be similar to that of Oracle's, there are important differences. First, the accelerator is implemented on the FPGA as in [5] and, thus, is not limited to a set of predefined functions but can be used to run arbitrary code. Second, the interaction with the CPU happens through a cache coherent interface unlike in the DAX or [5] where the accelerator is inserted in the data bus. An extreme version of the system will let the CPU use only the memory behind the FPGA as its main memory, with the FPGA acting as a fully customizable memory controller that can tailor memory access and on-the-fly data manipulation to the task at hand. The DRAM available to the CPU can be used as scratch space for storing intermediate results.

The design is made even more interesting thanks to the network access available to the FPGA and the RDMA networking stack, which can be used to implement a form of remote or far memory [1]. The same customizable memory management functions provided to the local CPU can be provided to remote processors (CPU or FPGAs) who can access the memory through RDMA. This results in a huge pool of *active* memory across a cluster of machines with the possibility of using customizable memory controllers tailoring the memory access to the particular application needs. If the CPU is not involved at all, the memory on the FPGAs can be used to emulate a pool of disaggregated memory supported by a smart NIC/memory controller and directly accessible through the network. Many other designs are possible thanks to the ability to insert arbitrary code on the access path to/from memory and the ability to do this remotely through RDMA.

One example demonstrating the potential for using the FPGA as a customizable memory controller is to provide to the CPU striding access to the memory on the FPGA. The DRAM on the FPGA is typically accessed through several channels (4 in Enzian). To improve memory bandwdith, these channels can be used independently but the application needs to take care of the coordination and sharding of the data. LynX provides a simple interface where the data sharding and the strided access to all memory modules happens automatically and transparently. A reader familiar with traditional databases will immediately detect the similarities of the approach with that of RAID or parallel disks used in the past to increase the I/O bandwidth in database engines. An initial implementation of this idea on LynX and Enzian allows us to compare automated striding with hand coded single channel and two channel accesses (Figure 2). The results show that the performance of memory striding is very close to that of hand coded access to two channels but with the advantage of being fully automatic. The impact of such functionality on, e.g., a database scan over large amounts of data is obvious, especially when combined with processing or filtering done directly at the FPGA.

Another example of customized memory controller is the traversal of remote data structures using RDMA for remote access to memory and going through the FPGA to provide additional functionality (what we call a *kernel*). We have implemented, for instance, a simple kernel with a RPC interface that, upon a read request, triggers the FPGA to traverse a linked list looking for the corresponding item and returning it when found. We have similar examples with B-trees for indexes or hash tables for key-value stores. This approach allows to implement one-sided RDMA calls that behave like two-sided RDMA calls. Figure 3 shows the difference in access latency between using our FPGA kernel approach and using conventional one sided RDMA reads. The example mirrors recent work in accessing remote indexes using RDMA but using the FPGA as a smart NIC [18], which has considerable advantages in terms of performance. The same approach can be used to implement scans or indexed access over tables, all happening at the remote memory and accessible through a simple **get** interface.

[7]https://community.oracle.com/docs/DOC-994842

From a database perspective, the possibilities opened up by near memory processing and smart remote memory are endless. It is easy to imagine implementing a customized pre-fetcher that, knowing the operator that is being executed, e.g., a table scan, takes advantage of the known access pattern to pre-fetch the data accordingly. The pre-fetcher can be combined with near-memory processing capabilities such as pre-sorting the data, partial aggregation, filtering and transformation (compression/decompression, encryption/decryption). Operators accessing memory through an index can complete the index traversal entirely on the memory controller, freeing up the CPU, not to mention the avoidance of cache pollution on the CPU as the index does not reach the CPU cache during the traversal operation. Similarly, the memory controller could perform data transformations such as turning rows into columns or vice-versa as we have implemented in doppioDB 1.0 [3], typecasting, checking constraints as data is read or written, or even give the impression of the pages in memory have different physical representations. More ambitious efforts could combine these features to, e.g., make graphs look like tables by dynamically navigating the graph and listing the nodes visited, make tables look like graphs by constantly performing recursive queries that determine the next row to explore, etc. In essence, the memory becomes just one form of physical representation and the customized memory controller implemented on the FPGA provides arbitrary, consistent views over such physical representation.

4.2 Near Storage

The notion of customizable memory controller can be extended to any form of storage, be it Flash, NVM, HBM, conventional disks, or even disaggregated memory. The same ideas discussed above apply in this context as well, including the fact that the ability to access the system through RDMA potentially turns every Enzian node in the system into a smart storage server.

Enzian+doppioDB can be configured to implement a system such as Caribou [8] which provided a key value store on FPGAs directly supporting data processing and consistency. The advantage of Enzian+doppioDB over Caribou is the access to a full-fledged database on the CPU side. From the database side, such a system can be used to delegate to the FPGA the task of coordinating the replication of updates to the database, thereby providing a way to implement consistent replication without having to involve the CPU. The extraction of the actual changes to propagate can be done by analyzing the memory traffic, with no CPU overhead and minimal latency.

Enzain+doppioDB can be also used to implement smart, local disks such similar to those proposed by Samsung [9]. This is orthogonal to whether the compute node has only local storage or has access to a larger pool of storage servers as it commonly happens in cloud settings. The same functionality and advantages apply regardless of whether the local storage is a shared-nothing architecture or is used as a local cache on top of a remote storage system. Beyond such on-the-fly data processing tasks, we also envision the possibility of using the FPGA based memory controller as a way to perform important background tasks such as repairing memory errors, cleaning up data, obtaining statistics on the data itself, automatic index maintenance, replication, check-pointing, etc.

4.3 Extending Database Functionality

It can be, controversially, argued that additional throughput or reduced latency is no longer the main problem of database engines. Existing systems, whether commercial or open source, seem to be good enough for the vast majority of workloads and situations. Although hardware acceleration has become an important topic, CPUs are often very good at what they do. For instance, in DoppioDB 1.0 we have not been able to accelerate complete joins when compared to the best multi-core implementations (this might change when FPGAs have High Bandwdith Memory). The approach we have taken in doppioDB is that it is more promising to extend the functionality of databases than improving existing features. An example from our previous work is string processing through the SQL *LIKE* clause. Existing engines support it, but only in limited settings and performance is often dismal. An FPGA based accelerator boosts the performance of queries using *LIKE* by orders of magnitude [15], thereby enabling much more efficient processing within the database of an important data type. The same can be said about performing machine learning directly on the database [3] or finally integrating well known operators, e.g., Skylines, that few systems actually support. Currently we have a number of machine learning operators from stochastic gradient descent in various forms to decision trees already implemented and tested on the FPGA that will be integrating soon in Enzian to take advantage of the improved hardware [17, 11].

Another approach to extend databases is to use a machine learning model to, on the fly, infer the correct value for missing values on tuples being written to or inserted into the database. This would be a very useful functionality, enforcing integrity constraints and even performing ETL operations on the fly. Doing so using a multi-core machine is probably not efficient, as running the inference would interfere with normal operations. In Enzain+doppioDB, the FPGA can be used for inference as it can not only perform the inference without using CPU resources but can also perform the insertion directly once the missing values are completed. The parallelism intrinsic to an FPGA and the possibility of implementing deep pipelines give the FPGA a significant edge over the CPU for such tasks. The inference process can be combined with other operations such as performing complex data cleaning upon insertion.

The high bandwidth cache coherency protocol between the CPU and the FPGA allows to forward a significant amount of data from the CPU to the FPGA. We can take advantage of this to send hardware and software instrumentation data form the CPU to the FPGA to run ML models and heuristics geared to optimize the performance of the system by, e.g., reassigning priorities, allocating more memory, increase or reduce parallelism, etc. This is a basic system chore that becomes most interesting in the context of a query optimizer reacting in real time to the instrumentation data obtained from the CPU. Seeing the FPGA as a co-processor in charge of auxiliary system tasks is a powerful scenario. For instance, these days there is a lot interest in using learning to optimize different database aspects. Most of this work does not discuss the logistics of learning and how to use it in a dynamic environment. Enzian+doppioDB provide a platform for processing an exhaustive stream of instrumentation data on the fly (see below) to learn from as a first step towards making automatic database tuning and operation a reality.

4.4 Streaming at Wire Speed

One of the key advantages of FPGAs is the ability to process data in a streaming fashion. This has been, in fact, one of the main use cases of FPGAs in applications such as deep network packet inspection or algorithmic trading, where latency is of utmost importance and where the overhead of receiving data through a network card, transfer the data to memory through PCI, and then from there to the CPU for processing is simply not an option. The architecture of Enzian+DoppioDB opens up the possibility of combining a regular relational engine with a streaming engine running on the FPGA that can process the streams at wire speed, without any latency overhead, and using pipelined designs to significantly increase throughput. The fact that the FPGA has access to unified memory space implies the FPGA can read directly tables and use them to, e.g., join them with arriving streams without involvement of the CPU. Just from the nature of the data path through the architecture, such a setting will immediately be faster than any CPU based streaming system. Currently we are exploring the use of Enzian+doppioDB in conjunction with our own streaming engine [7, 10] to identify the most promising venues for acceleration.

5. PROJECT STATUS AND TIMELINE

At the time of writing, we have two operational prototypes connecting a Cavium ThunderX board to a Xilinx Ultrascale board (Figure 1.b). These prototypes have been used for instrumentation, to obtain traces of the system's components, to developed LynX, the memory controllers, test the networking stacks, as well as to develop the FPGA side of the cache coherency protocol.

An integrated board following the basic design shown in Figure 1.a is expected to be available at the end of 2019. A full release of the hardware and software is intended for the second half of 2020. Simultaneously, we are already exploring the next versions of Enzian considering different processors, different accelerators, and changing key aspects of the system such as the cache coherence protocol.

We plan to have a basic version of doppioDB on top of Enzian at the time the system is publicly released. Currently, we are obtaining experimental data on different configuration and basic functionality, exploring the potential of many of the ideas described in the paper: database extensions for ML, stream data processing, views over memory representations, and uses of RDMA and smart NICs in databases [2]. The next steps will be integrating each one of those ideas in doppioDB and evaluate their performance.

Enzian+doppioDB is an open research platform. We hope the project turns into a collaborative effort where many groups contribute to creating potential leverage to influence hardware evolution through prototype systems backed by data obtained on actual hardware.

Acknowledgements

The following doctoral students and post-docs have contributed to the development of Enzian and DoppioDB in a variety of capacities: Zsolt Istvan, Adam Turowski, Tobias Grosser, Amit Kulkarni, Reto Achermann, Abishek Ramdas. We would like to thank Cavium and Xilinx for the generous donations of hardware that have made the construction of the first prototypes possible.

6. REFERENCES

[1] M. K. Aguilera, K. Keeton, S. Novakovic, and S. Singhal. Designing far memory data structures: Think outside the box. In *Hot OS*, 2019.

[2] G. Alonso, C. Binnig, I. Pandis, K. Salem, J. Skrzypczak, and et al. DPI: the data processing interface for modern networks. In *CIDR*, 2019.

[3] G. Alonso, Z. Istvan, K. Kara, M. Owaida, and D. Sidler. DoppioDB 1.0: Machine Learning inside a Relational Engine. *IEEE Data Engineering Bulletin*, 42(2), 2019.

[4] A. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, and et al. A Cloud-Scale Acceleration Architecture. In *MICRO*, 2016.

[5] Y. Fang, C. Zou, and A. Chien. Accelerating Raw Data Analysis with the ACCORDA Software and Hardware Architecture. In *PVLDB*, 2019.

[6] D. Firestone, A. Putnam, H. Angepat, D. Chiou, A. Caulfield, C. Chung, M. Humphrey, and et. al. Azure Accelerated Networking: SmartNICs in the Public Cloud . In *NSDI*, 2018.

[7] M. Hoffmann, A. Lattuada, F. McSherry, V. Kalavri, J. Liagouris, and T. Roscoe. Megaphone: Latency-conscious state migration for distributed streaming dataflows. *PVLDB*, 12(9), 2019.

[8] Z. István, D. Sidler, and G. Alonso. Caribou: Intelligent distributed storage. *PVLDB*, 10(11), 2017.

[9] I. Jo, D.-H. Bae, A. S. Yoon, and J.-U. Kang. YourSQL: A HighPerformance Database System Leveraging In Storage Computing. *PVLDB*, 9(12), 2016.

[10] V. Kalavri, J. Liagouris, M. Hoffmann, D. C. Dimitrova, M. Forshaw, and T. Roscoe. Three steps is all you need: fast, accurate, automatic scaling decisions for distributed streaming dataflows. In *OSDI*.

[11] K. Kara, K. Eguro, C. Zhang, and G. Alonso. ColumnML: Column-store Machine Learning with On-the-fly Data Transformation. *PVLDB*, 12(4), 2018.

[12] K. Kara, Z. Wang, C. Zhang, and G. Alonso. DoppioDB 2.0: Hardware Techniques for Improved Integration of Machine Learning into Databases. In *PVLDB*, 2019.

[13] M. Ruiz, D. Sidler, G. Sutter, G. Alonso, and S. Lopez-Buedo. Limago: an FPGA-based Open-source 100 GbE TCP/IP Stack. In *FPL*, 2019.

[14] D. Sidler, Z. István, and G. Alonso. Low-latency TCP/IP stack for data center applications. In *FPL*, 2016.

[15] D. Sidler, Z. István, M. Owaida, and G. Alonso. Accelerating Pattern Matching Queries in Hybrid CPU-FPGA Architectures. In *SIGMOD*, 2017.

[16] D. Sidler, M. Owaida, Z. István, K. Kara, and G. Alonso. DoppioDB: A Hardware Accelerated Database. In *SIGMOD*, 2017.

[17] Z. Wang, K. Kara, H. Zhang, G. Alonso, O. Mutlu, and C. Zhang. Accelerating Generalized Linear Models with MLWeaving: A One-Size-Fits-All System for Any-Precision Learning. *PVLDB*, 12(7), 2019.

[18] T. Ziegler, S. T. Vani, C. Binnig, R. Fonseca, and T. Kraska. designing distributed tree-based index structures for fast rdma-capable networks.