MXQuery With Hardware Acceleration

Peter M. Fischer [1,*], Jens Teubner [2]

[1]*Department of Computer Science, University of Freiburg, Germany*
peter.fischer@informatik.uni-freiburg.de

[2]*Systems Group, Department of Computer Science, ETH Zurich, Switzerland*
jens.teubner@inf.ethz.ch

***Abstract*—We demonstrate *MXQuery/H*, a modified version of MXQuery that uses *hardware acceleration* to speed up XML processing.**

The main goal of this demonstration is to give an interactive example of *hardware/software co-design* and show how system performance and energy efficiency can be improved by off-loading tasks to FPGA hardware. To this end, we equipped *MXQuery/H* with various hooks to inspect the different parts of the system.

Besides that, our system can finally really leverage the idea of *XML projection* [3]. Though the idea of projection had been around for a while, its effectiveness remained always limited because of the unavoidable and high *parsing overhead*. By performing the task in hardware, we relieve the software part from this overhead and achieve processing speed-ups of several factors.

I. Introduction

From a language perspective, XML-based data and query processing has been unprecedentedly successful, penetrating virtually all application areas that had been dominated by domain-specific solutions and languages previously. System makers, however, are scared by significant costs that are often associated with the versatile data model. Parsing and memory overheads often defeat the use of XML when the involved data volumes become large [7].

XML projection [3] is known as an effective tool to minimize data volumes in the query processor. But the idea cannot help with the parsing overhead, which dominates cost in many real-world applications. *Off-load engines for XML*, built in tailor-made hardware, have been suggested as another approach to speed up XML processing [8], [5]. But it remains unclear how such engines could be integrated with a software-based counterpart that would permit the full-fledged XML processing that is needed in real-world applications.

In this demonstration, we combine both strategies and showcase a particularly effective case of *hardware/software co-design*. We extend *MXQuery*, a mature XQuery processor optimized for streaming [1], with an FPGA-based implementation of XML projection to obtain *MXQuery/H*, a full-fledged XQuery processor with remarkable performance and energy efficiency characteristics. As such, our demonstration is equally relevant to people that work on high-volume XML processing and to those interested in systems and modern hardware aspects.

*This work was done while Peter Fischer was with the Systems Group at ETH Zurich.

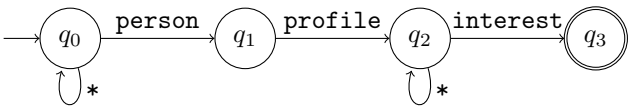

Fig. 1. Non-deterministic finite-state automaton to evaluate projection path `//person/profile//interest`. Each `descendant` step becomes a $\circlearrowleft *$ loop in the automaton.

Our demonstration setup is designed for interactivity. Visitors will be able to enter their own XQuery expressions and observe the improvements of FPGA-based co-processing on execution times as well as on memory and CPU consumption.

In the following, we sketch the main ingredients of *MXQuery/H*, XML projection and hardware acceleration (Section II); illustrate how they can be combined into a working system (Section III); describe the setup of this ICDE demo (Section IV); and then wrap up in Section V.

II. Background

A. Projecting XML

To understand the idea of XML projection [3], consider the following query (assuming an XMark schema):

```
for $p in doc('auction.xml')//person return
  <person> <name>{ $x/name }</name>
           <num-interests>                    (Q1)
            { count($x/profile//interest) }
           </num-interests> </person> .
```

During execution, this query will have to touch only very few nodes out of a potentially large XML instance. And, what is more, the relevant nodes can be described using a simple set of *projection paths*:

```
{  //person,
   //person/name #,
   //person/profile//interest  } .
```

Once derived from the input query (Marian and Siméon [3] detail the derivation procedure), the set of projection paths can be used to *pre-filter* the source document during load. The filter preserves only those nodes (and their root-to-node paths) in the document that match one of the projection paths. In addition, nodes can pass the filter if they are a descendant of a node matched by a `#`-marked projection path.[1]

[1]Observe how, *e.g.*, the full subtrees below `person` and `interest` elements are not needed to evaluate Q_1, whereas those below `name` elements are.

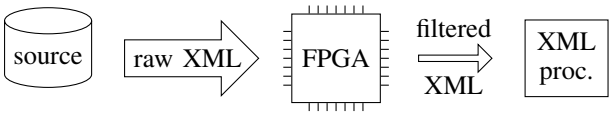

Fig. 2. XML projection in the system data path. Data is filtered as it is retrieved from the source and before it enters the software system.

The achieved filtering effect can be significant. For instance, between 73 % and 99 % of all input data can be discarded in case of the 20 XMark benchmark queries (average: 97 %). At the same time, projection is easy to realize. All projection paths are composed of `child` and `descendant` steps only and can thus be evaluated efficiently with help of *finite-state automata*. For instance, Figure 1 visualizes the state automaton for the projection path `//person/profile//interest`.

On the flip side, projecting an XML document in software cannot avoid the cost of *parsing* the entire input file or data stream. Unfortunately, this cost in practice dominates the execution time of any decent file- or stream-based XML processor [7]. Typical parsing speeds for a modern XML processor range between 10 and 30 MB/s.

B. Filtering in the Data Path

The high parsing cost can be avoided if XML projection is implemented *in hardware* instead and applied *before* the (projected) document enters the software system. The problem then becomes an instance of filtering task *in the data path*, for which *field-programmable gate arrays (FPGAs)* have been found a good fit in the past [6].

Figure 2 illustrates this idea. As the data is retrieved from its source, it is pre-filtered on the FPGA. The software-based XML processor "sees" (and parses) only a small and relevant subset of the original document.

C. Hardware-Based Projection

As mentioned before, projection paths can be matched with help of finite-state automata (analyzing the root-to-node path of each tree node v). It is known that FPGAs can run finite-state automata very efficiently. The available FPGA chip resources *memory* (flip-flop registers); *logic gates* (lookup tables); and *signal wires* map very naturally to the three ingredients of any state machine: states; conditions; and transitions. By re-configuring these resources, the FPGA can be tailored to run any FSM directly in hardware. Several prototypes and systems have demonstrated the effectiveness of this idea (*e.g.*, [5], [9], [10]).

The idea has its drawback, however. Re-compiling and re-loading the FPGA circuit for every workload change is a very time-consuming process, ranging from a few minutes to several hours. Obviously, this is not an option for an interactive system like *MXQuery*. Therefore, in *MXQuery/H* we use a new evaluation mechanism (using so-called "skeleton automata") where workloads can be changed instantly and at any time.

To this end, we separate FSM characteristics into a *static part* that is query-independent and a *dynamic part* that covers all specifics of a particular query. We implement the static part through a set of *skeleton segments* that we instantiate in the FPGA hardware. Once they are instantiated, the FPGA circuit is fixed and need not be re-loaded even when workloads change.

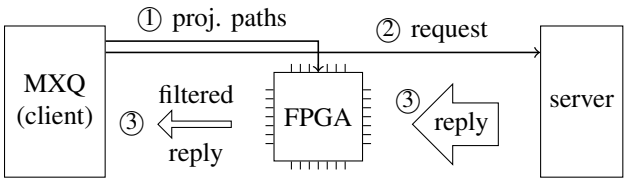

Fig. 4. *MXQuery/H* system design. For each query, *MXQuery/H* sends projection path information to the FPGA ① and a data request to the server ②. Data is sent back and filtered on the FPGA ③. All communication is through an Ethernet network.

Query-specific aspects are covered by the dynamic part. It is realized through *runtime parameters* available in each skeleton segment. They are held in on-chip storage units (flip-flop registers and FPGA Block RAM) and can be modified arbitrarily at runtime. Workload changed can thus be accommodated instantly.

Figure 3 visualizes the idea. The figure shows four skeleton segments (solid boxes) that are wired together to form a chain. As indicated with dashed lines, each segment matcher implements a small piece of a state automaton, with one input and a back-loop transition. The conditions on the input transition and the back-loop can both be defined through runtime parameters.

The four segments in Figure 3 are parameterized to implement the state automaton that we saw earlier in Figure 1. To do so, we set the incoming transition condition to the respective tag name (or to `true` for the initial state), and we enabled the back-loop where needed with a Boolean parameter.

Our "skeleton automaton" design runs at the same speed as existing automata with off-line compilation (*e.g.*, [5], [9], [10]). The price we pay is a small space overhead. Still, the design is compact enough to host several hundred XPath location steps on low-end FPGA hardware. This is more than enough for common uses of XML projection.

III. *MXQuery/H* = HARDWARE + SOFTWARE

To combine our hardware with the MXQuery XML processor, we realized the data path of Figure 2 as a physical Ethernet network. MXQuery—already designed to work from network-based sources in a streaming fashion [1]—requests its input data from a network server. Rather than replying directly to the XML processor, the server sends the raw XML stream to the FPGA pre-filter. There, the data is projected and forwarded to the MXQuery instance.

For effective filtering, the FPGA has to be configured with the right set of projection paths before the document is sent. Thus, we modified MXQuery to first infer this set (based on the inference procedure of [3]) and send it to the FPGA as a configuration instruction, before requesting the input data from the server.

Figure 4 illustrates how a query is processed in *MXQuery/H*. First, the software system sends projection path information to the FPGA ①, then requests the XML data from the server ②.

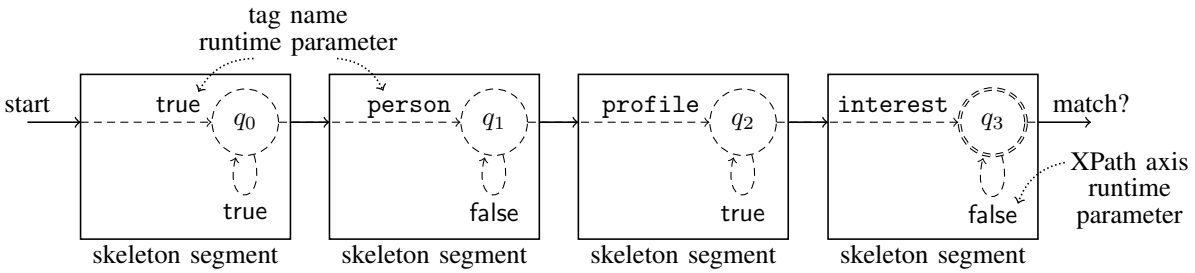

Fig. 3. Two-part "skeleton automaton" design for hardware-based XML projection in *MXQuery/H*. The FPGA configuration mechanisms are used to realize the *static part* of the automaton (drawn as solid lines), whereas all query-specific behavior is covered by the *dynamic part* (dashed lines) that can be parameterized at runtime.

The reply is sent via the FPGA ③, which filters the data "in the network."

Performing XML projection in the Ethernet network is inspired by common XML and XQuery use cases: 1) XML as the common data interchange format needs to be routed, filtered and transformed on the fly, ideally at wire speed. 2) Multi-tier or cloud-based XML databases distribute and integrate XML data sources over multiple hosts, and need to retrieve the relevant parts with maximum efficiency.

Apart from that, the design also allows us to make good use of available FPGA features: modern Xilinx FPGAs ship with a built-in Ethernet controller that allows for network data processing at exceptionally low latency and high throughput.

All communication channels use plain XML. This makes it particularly easy to mix and match components, potentially also from alternative XML solution providers (in fact, we successfully also paired our hardware implementation with the open-source version of Saxon [4]). In the context of an ICDE software demonstration, the feature comes in handy to analyze and demonstrate individual system components easily.

A. Runtime Characteristics

For most XQuery processors and workloads, the observable processing speed is determined by the capabilities of the system's XML parser. Unfortunately, XML parsing is notoriously hard to parallelize, such that even highly tuned XML processors rarely achieve parsing rates beyond 30 MB/s—far below the speed at which the data could be read from disk or network (≈ 100 MB/s).

The parsing speed of MXQuery is even lower. As can be seen in Figure 5, the off-the-shelf version of MXQuery (plotted in light gray ▭) requires at least 2 seconds to evaluate even very simple XMark queries over a 24 MB XML file (XMark scale factor 0.2). MXQuery is bottlenecked here by its parsing speed of around 12 MB/s.

XML projection in the network data path eliminates this bottleneck and we see significant performance improvements (shown in dark gray ■ in Figure 5). For the 20 XMark queries and our 24 MB input, the achieved speed-up ranges between 2.2x and 13.7x (note the logarithmic scale in Figure 5).

For XMark Query *Q15*, hardware-based projection even exposes the next bottleneck in client/server-oriented XML processing. Here the system became limited by the server's ability to send data fast enough via the physical 1 Gb/s Ethernet speed. The CPU demand on the client side is much lower. As indicated using ▨ in Figure 5, the actual client CPU time needed to evaluate the query is actually less than the 211 ms our server needs to send all data through the network.

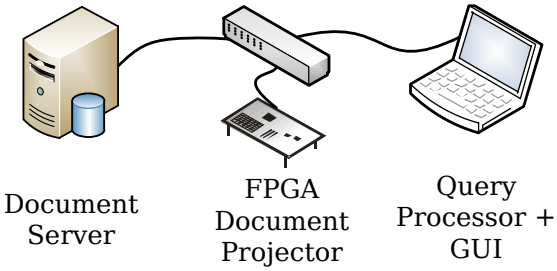

Fig. 6. *MXQuery/H* demonstration setup.

A second major bottleneck in XML processing, high memory consumption, is more difficult to measure reliably for a Java-based XQuery processor like MXQuery. Roughly, we found the maximum amount of memory needed during query execution to be proportional to the amount of XML data loaded into the software processor. By pre-filtering the XML stream, we can thus run the XMark benchmark at much higher scale factors than the off-the-shelf version of MXQuery could.

IV. Demonstration Setup

To demonstrate *MXQuery/H* at ICDE 2012, we will bring two laptops that represent the server and client machines, as shown in Figure 6. Most interaction of the demonstration visitors will be with the client side, where a GUI interface lets users state their own queries; watch and modify the runtime projection paths; and observe the effect of in-network projection. To this end, we extended the MXQuery runtime infrastructure to provide information on execution performance, memory consumption, and CPU utilization for both parsing and query processing.

We will show hardware-based projection based on an XUPV5 FPGA development board, a widely-used and low-priced (US$ 750) FPGA platform. It includes a low-end (but easily sufficient for our purposes) Virtex-5 LX110T FPGA and a 1 Gb/s Ethernet interface, which we use to connect the client and server laptops (through a gigabit switch; see Figure 6).

The XUPV5 board also includes an LCD display, which we

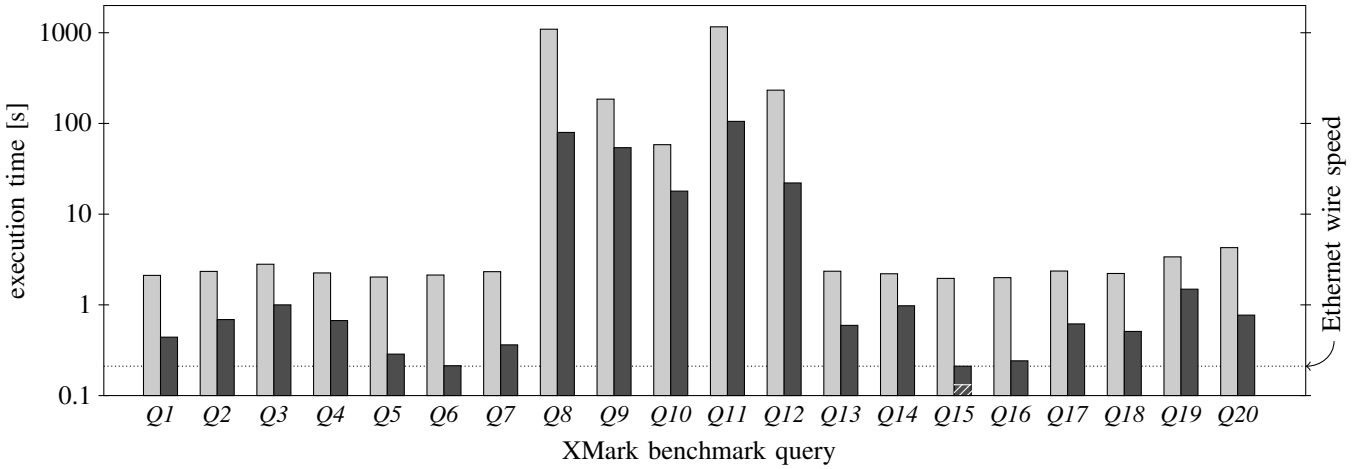

Fig. 5. MXQuery execution times for the 20 XMark queries (scale factor 0.2; ≈ 24 MB) before (▭) and after (■) input projection. With projection, Query *Q15* becomes bound by the physical network speed. Without this bound, *Q15* could run significantly faster, as indicated with ▨.

will configure to show key performance characteristics, such as throughput and network packet rates. With help of FPGA design tools, demonstration visitors will also have the opportunity to inspect runtime internals of our "skeleton automaton" system and get an impression of a typical development cycle for hardware/software co-designed systems.

V. Summary

There is a general consensus that hardware/software co-design is the key to escape from architectural limitations of current computer systems. In recent years, this has triggered a large number of research and development efforts on the hard- as well as on the software side.

Our work brings both worlds together and provides a working implementation of the complete stack. We off-load *XML projection* to a tailor-made hardware solution and thus leverage one of the particular strengths of FPGA hardware, regular expression matching. At the same time, XML projection has known-good properties that can significantly reduce the cost in answering XQuery expressions. Our implementation confirms this cost reduction with speed-ups in the range of 2.2x to 13.7x on XMark benchmark data.

The motivation for this demonstration is two-fold: *(a)* we illustrate the potential of hardware/software co-designed systems to increase efficiency and performance; *(b)* by giving a concrete example, we show how applications can be designed to benefit best from the characteristics of hard- and software.

MXQuery/H is work in progress. While XML projection in hardware significantly reduces the amount of data that has to go through software-based parsing, parts of the document now have to be parsed twice: first in hardware then in software. We are looking into techniques to avoid also (parts of) this re-parsing cost, which could be achieved by using an alternative "binary" format when data is forwarded by the FPGA.

The effectiveness of hardware support for XML parsing has also been demonstrated previously in the context of the *wire-speed processor (WSP)* [2] effort at IBM. WSP combines 4–16 PowerPC cores and various application accelerators—among them support for XML parsing—within a system-on-a-chip design. WSP does not, however, offer hardware support for the processing of queries over XML data.

In terms of functionality, we also look at additional filter criteria that could be used to further increase filter selectivity. In particular, with support for *value-based predicates* (*e.g.*, on attribute values) some meaningful query types could even be processed entirely in hardware.

Acknowledgements

This work is part of the *Avalanche* project at ETH Zurich, funded by the Swiss National Science Foundation (SNSF) via an *Ambizione* grant for Jens Teubner.

References

[1] Irina Botan, Peter M. Fischer, Daniela Florescu, Donald Kossmann, Tim Kraska, and Rokas Tamosevicius. Extending XQuery with Window Functions. In *Proc. of the 33rd VLDB Conference*, Vienna, Austria, September 2007.

[2] Hubertus Franke, J. Xenidis, Claude Basso, Brian M. Bass, Sandra S. Woodward, Jeffrey D. Brown, and Charles L. Johnson. Introduction to the Wire-Speed Processor and Architecture. *IBM Journal of Research and Development*, 54(1):3:1–3:11, 2010.

[3] Amélie Marian and Jérôme Siméon. Projecting XML Documents. In *Proc. of the 29th VLDB Conference*, Berlin, Germany, September 2003.

[4] Michael Kay, Saxonica Inc. Saxon-HE 9.3.0.4J.

[5] Roger Moussalli, Mariam Salloum, Walid A. Najjar, and Vassilis J. Tsotras. Massively Parallel XML Twig Filtering Using Dynamic Programming on FPGAs. In *Proc. of the 27th ICDE Conference*, Hannover, Germany, April 2011.

[6] Rene Mueller, Jens Teubner, and Gustavo Alonso. Streams on Wires—A Query Compiler for FPGAs. *Proc. of the VLDB Endowment (PVLDB)*, 2(1), August 2009.

[7] Matthias Nicola and Jasmi John. XML Parsing: A Threat to Database Performance. In *Proc. of the 12th CIKM Conference*, New Orleans, LA, USA, November 2003.

[8] Jan van Lunteren, Ton Engbersen, Joe Bostian, Bill Carey, and Chris Larsson. XML Accelerator Engine. In *Proc. of the 1st Int'l Workshop on High-Performance XML Processing*, New York, NY, USA, May 2004.

[9] Louis Woods, Jens Teubner, and Gustavo Alonso. Complex Event Detection at Wire Speed with FPGAs. *Proc. of the VLDB Endowment (PVLDB)*, 3(1), 2010.

[10] Yi-Hua E. Yang, Weirong Jiang, and Viktor K. Prasanna. Compact Architecture for High-Throughput Regular Expression Matching on FPGA. In *Proc. of the 2008 ANCS Conference*, San Jose, CA, USA, 2008.