FPGAs: A New Point in the Database Design Space

Rene Mueller Jens Teubner
Systems Group, Department of Computer Science, ETH Zurich
Haldeneggsteig 4, 8092 Zurich, Switzerland
{rene.mueller,jens.teubner}@inf.ethz.ch

ABSTRACT

In line with the insight that "one size" of databases will not fit all application needs [19], the database community is currently exploring various alternatives to commodity, CPU-based system designs. One particular candidate in this trend are *field-programmable gate arrays (FPGAs)*, programmable chips that allow tailor-made hardware designs optimized for specific systems, applications, or even user queries.

With a focus on database use, this tutorial introduces into FPGA technology, demonstrates its potential, but also pinpoints some challenges that need to be addressed before FPGA-accelerated database systems can go mainstream. The goal of this tutorial is to develop an intuition of an FPGA development cycle, receive guidelines for a "good" FPGA design, but also learn the limitations that hardware-implemented database processing faces. Our more high-level ambition is to spur a broader interest in database processing on novel hardware technology.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems; C.5 [**Computer System Implementation**]: VLSI Systems

General Terms

Design

Keywords

FPGA, hardware acceleration, data processing, VLSI

1. INTRODUCTION

Database applications built on top of general-purpose hard- and software systems satisfied actual industry demands for a remarkably long time. Only recently did the database community start to realize that "one size" will not fit all application needs [19].

Aside the emergence of new software architectures (such as column-store databases [16] or MapReduce-style engines [6]), the insight also led the community to explore alternatives on the hardware side. A number of database algorithms has been ported to modern processor architectures such as IBM's Cell [7, 8, 21], network processors [2, 9], graphics processors [3, 10, 11, 12], or the vector instruction sets of modern CPUs [4, 22].

In this tutorial, we look at *FPGAs* (*field-programmable gate arrays*) as another class of hardware technology that seems particularly interesting for high-volume data processing. Naïvely spoken, FPGA chips consist of a number of logic gates whose wiring can be programmed by software (more details below). The programmed chip can then be used, *e.g.*, as a hardware-accelerated implementation for specific compute or control tasks.

1.1 FPGAs for Database Co-Processing

Our interest here is in the application of FPGAs for *database co-processing*. The data-intensive nature of database tasks makes them a particularly good fit for FPGA-based processing. Streaming databases, *e.g.*, may benefit from the *low latencies* that FPGA implementations can provide even under high load. More traditional systems can use their existing set-oriented query formulations to exploit the high degree of *parallelism* inherent to programmable hardware.

Unfortunately, the potential of FPGAs is still not very widely known. One reason may be that the processing model of FPGAs—and hence the way they are controlled by software—is very different to the traditional von Neumann architecture that computer scientists are used to deal with.

1.2 Outline

Our tutorial is organized in three units whose contents we sketch in Sections 2–4. Roughly speaking, we give the necessary background about the inner workings of an FPGA in Unit 1. In Unit 2, we then illustrate some design techniques that can be used to create efficient FPGA circuits for database tasks. In Unit 3 we finally discuss how such FPGA circuits can be combined with commodity hardware to build heterogeneous FPGA/CPU co-designs.

We do not expect any particular background from tutorial attendees. Parts of the tutorial will also contain technical material, but attendees may easily skip these parts and still follow the remainder of the tutorial.

2. FPGA BASICS

A basic understanding of the hardware internals is important to judge the trade-offs that occur when building an FPGA circuit. In the first unit of this tutorial, we give a

EDBT 2010, March 22–26, 2010, Lausanne, Switzerland.

	Virtex-II Pro XC2VP30	Virtex-5 XC5VFX200T
Lookup Tables (LUTs)	27,392	122,880
Flip-Flops (registers)	27,392	122,880
Block RAM (kbit)	2,448	16,416
18-bit Multipliers	136	384
PowerPC Cores	2	2
maximum clock speed (MHz)	≈ 100	≈ 550
release year	2002	2006

Table 1: Selected characteristics of Xilinx FPGAs.

very brief overview about the inner workings of the underlying hardware and highlight some particular features which are most relevant in a database context.

2.1 FPGA Internals

In essence, FPGAs provide a large pool of *resources* that can be configured to implement a user circuit. Circuits are specified using a *hardware description language* (such as VHDL or Verilog) that is compiled into a *bitstream* using vendor-provided tools, then loaded into the FPGA chip.

Table 1 lists the most relevant resources to choose from: *lookup tables* are a configurable type of logic gates, *registers* and *Block RAM (BRAM)* provide different types of on-chip memory, specialized *hard cores* (such as multipliers or even full-fledged CPUs) contain implementations of often-required functionality directly in silicon. An *interconnect fabric* provides on-chip wiring between all available resources. The challenge in building good FPGA designs is to efficiently manage the provided resources.

2.2 Potential and Limitations

The potential of the FPGA technology comes with limitations. For instance, the high degree of parallelism is contrasted with relatively small amounts of memory to hold state. In addition, as we show with short VHDL examples, releasing this parallelism is nontrivial. Building essentially a tailor-made piece of hardware takes the engineer beyond what he or she is used to in the software-only world.

3. USING FPGAS

Past research has developed a number of serviceable guidelines that help building efficient FPGA designs. In Unit 2 of the tutorial we focus on the most important techniques that are relevant for typical database tasks.

Inherent Parallelism. The most apparent feature that FPGAs have to offer is their intrinsic parallelism. Proper circuits can reach a degree of parallelism that is orders of magnitude higher than what can be achieved in general-purpose CPUs. In the tutorial we demonstrate different variants of parallelism—task, data, and pipeline parallelism—and how they can be implemented, combined, and mixed in FPGA circuits. Thereby, parallel circuits need not suffer from the synchronization overhead that typically leads to sub-optimal scaling in CPU-based systems.

Array-Based Designs. The most critical—and hardest to manage—resource constraint often turns out to be the interconnect fabric. Large designs tend to have long signal paths that lead to slow on-chip communication and inferior performance. The VLSI community has developed *systolic arrays* as an effective design technique to avoid this effect [15]. Based on recently proposed examples from the database domain (an implementation of the a-priori algorithm by Baker and Prasanna [1] and our own solution to the frequent item problem [20]), we demonstrate how systolic arrays can improve performance.

Circuit Speed. A design technique that can lead to very low latency and high work density inside the chip is the use of *asynchronous designs*. The runtime of an asynchronous component is solely dependent on the signal propagation times in the logic circuit and *not* bound to any external clock (as it is the case in CPU-based setups). By example of a *sorting network*, we show how even low-cost FPGA chips can outperform highly tuned software implementations on high-performance CPUs [17]. On the flip side, asynchronous circuits are generally harder to construct and cannot be pipelined. We illustrate, that, by splitting logic into stages, a computation pipeline can be built to trade throughput for latency.

3.1 Good FPGA Designs

We show several examples of how an FPGA circuit can be inferred from a high-level problem description. This way, our tutorial gives meaningful guidelines that show how a "good" FPGA design should look like and how to judge FPGA circuits for their quality, or to develop circuits for specific problems.

4. SYSTEM INTEGRATION

To make a hardware-accelerated operator implementation accessible to a database system, it has to be wired up to conventional components and connected to, *e.g.*, a general-purpose CPU. This *system integration* is the topic for Unit 3 of our tutorial.

Existing research work and industrial products indicate that two approaches to system integration are most promising:

FPGA in the Data Path. In systems like Netezza's TwinFin [5] or the Avalanche system that we presented in [18], the FPGA is inserted in to the system's *data path*:

The task of the FPGA here is to act as an early *filter* or *aggregator*. Located close to the data source, the FPGA often significantly reduces the volume of the data before a general-purpose CPU performs more complex high-level operations.

FPGA as a Co-Processor. Alternatively, the FPGA can function as a co-processor in a heterogeneous multi-core setup. Kickfire's Analytic Appliance [14] and XtremeData's dbX [13] use FPGAs in such a mode and off-load portions of a database query plan to the FPGA co-processor. A challenge is to ensure a sufficiently high communication bandwidth between general-purpose CPUs and the FPGA co-processor. As such, co-processor-based setups face similar integration challenges like graphics or network processors,

which have been proposed in recent database research papers [2, 3, 9, 10, 11, 12].

4.1 Hardware-Software Co-Design

This part of the tutorial is meant to give an intuition of what FPGAs can offer in a hard- and software co-design and what they cannot. We also relate FPGAs to other types of specialized hardware, such as graphics or network processors, or vector-processing features of general-purpose CPUs (SIMD). It turns out that problems and their solutions have a lot in common across the different technologies.

5. ABOUT THE AUTHORS

Both authors are actively working on FPGA-accelerated database processing in the context of the *Avalanche* project. The Systems Group at ETH is involved in a larger industry collaboration, where we currently build a stream processing engine with low latency at substantial throughput rates.

René Müller. After an undergraduate degree in electrical engineering, René Müller obtained a MSc in computer science from ETH Zurich. Since 2006, he is a PhD student at ETH Zurich, working on embedded data processing and wireless sensor networks. In his previous work, he developed *SwissQM*, a virtual machine-based stream processing platform for sensor networks.

Jens Teubner. Graduated with a PhD from TU München in 2006, Jens Teubner worked at the IBM T. J. Watson lab from 2007–2008. Since 2008, he is a postdoc at ETH Zurich, working on hardware-accelerated data processing. Most of his earlier work revolved around scalable XML processing. He was a co-founder of the *Pathfinder* XQuery compiler project.

6. REFERENCES

[1] Zachary K. Baker and Viktor K. Prasanna. Efficient Hardware Data Mining with the Apriori Algorithm on FPGAs. In *Proc. of the 13th Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, USA, April 2005.

[2] Nagender Bandi, Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Data Stream Algorithms using Associative Memories. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, Beijing, China, June 2007.

[3] Nagender Bandi, Chengyu Sun, Divyakant Agrawal, and Amr El Abbadi. Processing Spacial Data Using Graphics Processors. In *Proc. of the Int'l Conference on Very Large Databases (VLDB)*, Toronto, ON, Canada, 2004.

[4] Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, William Macy, Mostafa Hagog, Yen-Kuang Chen, Akram Baransi, Sanjeev Kumar, and Pradeep Dubey. Efficient Implementation of Sorting on Multi-Core SIMD CPU architecture. *Proc. of the VLDB Endowment*, 1(2), 2008.

[5] Netezza Corp. TwinFin^{TM}. `http://www.netezza.com/`.

[6] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *6th Symposium on Operating System Design and Implementation*, San Francisco, CA, USA, December 2004.

[7] Buğra Gedik, Rajesh R. Bordawekar, and Philip S. Yu. CellSort: High Performance Sorting on the Cell Processor. In *Int'l Conference on Very Large Databases (VLDB)*, Vienna, Austria, September 2007.

[8] Buğra Gedik, Philip S. Yu, and Rajesh Bordawekar. Executing Stream Joins on the Cell Processor. In *Proc. of the Int'l Conference on Very Large Databases (VLDB)*, Vienna, Austria, 2007.

[9] Brian T. Gold, Anastassia Ailamaki, Larry Huston, and Babak Falsafi. Accelerating Database Operations Using a Network Processor. In *Workshop on Data Management on New Hardware (DaMoN)*, Baltimore, MD, USA, June 2005.

[10] Naga Govindaraju, Jim Gray, Ritesh Kumar, and Dinesh Manocha. GPUTeraSort: High Performance Graphics Co-processor Sorting for Large Database Management. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, Chicago, IL, USA, June 2006.

[11] Naga K. Govindaraju, Brandon Lloyd, Wei Wang, Ming Lin, and Dinesh Manocha. Fast Computation of Database Operations using Graphics Processors. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, Paris, France, June 2004.

[12] Bingsheng He, Ke Yang, Rui Fang, Mian Lu, Naga K. Govindaraju, Qiong Luo, and Pedro V. Sander. Relational Joins on Graphics Processors. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, Vancouver, BC, Canada, June 2008.

[13] XtremeData Inc. dbX. `http://www.xtremedatainc.com/`.

[14] Kickfire. Analytic Appliance. `http://www.kickfire.com`.

[15] H. T. Kung and Charles E. Leiserson. Systolic Arrays (for VLSI). In *Sparse Matrix Proceedings*, Knoxville, TN, USA, November 1978.

[16] MonetDB. `http://monetdb.cwi.nl/`.

[17] Rene Mueller, Jens Teubner, and Gustavo Alonso. Data Processing on FPGAs. *Proc. of the VLDB Endowment*, 2(1), August 2009.

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

[19] Michael Stonebraker and Uğur Çetintemel. "One Size Fits All": An Idea Whose Time Has Come and Gone. In *Proc. of the 21st Int'l Conference on Data Engineering (ICDE)*, Tokyo, Japan, April 2005.

[20] Jens Teubner, Rene Mueller, and Gustavo Alonso. FPGA Acceleration for the Frequent Item Problem. In *Proc. of the 26th Int'l Conference on Data Engineering (ICDE)*, Long Beach, CA, USA, March 2010.

[21] Dina Thomas, Rajesh Bordawekar, Charu C. Aggarwal, and Philip S. Yu. On Efficient Query Processing of Stream Counts on the Cell Processor. In *Proc. of the 25th Int'l Conference on Data Engineering (ICDE)*, Shanghai, China, March 2009.

[22] Thomas Willhalm, Nicolae Popovici, Yazan Boshmaf, Hasso Plattner, Alexander Zeier, and Jan Schaffner. SIMD-Scan: Ultra Fast in-Memory Table Scan using on-Chip Vector Processing Units. *Proc. of the VLDB Endowment*, 2(1), August 2009.