Data Processing on Modern Hardware

Jens Teubner, TU Dortmund, DBIS Group
jens.teubner@cs.tu-dortmund.de

Summer 2016

Part VII

FPGAs for Data Processing

Motivation

Modern hardware features a number of "speed-up tricks":

- caches,
- instruction scheduling (out-of-order exec., branch prediction, ...),
- parallelism (SIMD, multi-core),
- throughput-oriented designs (GPUs).

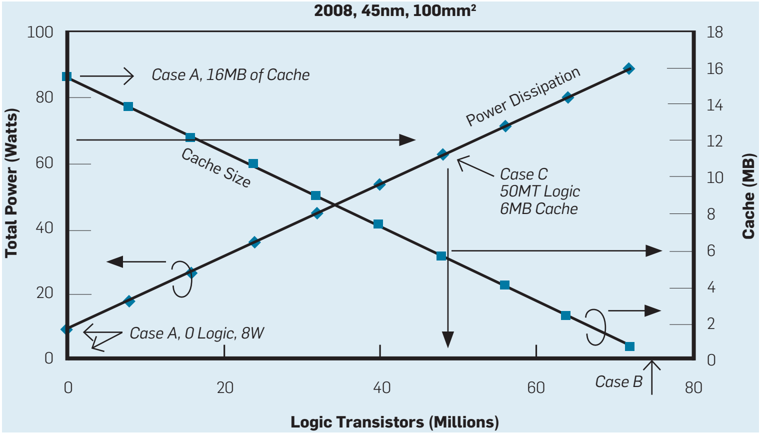
Combining these "tricks" is essentially an **economic choice**:

$\rightarrow$ chip space $\equiv$ €€€

$\rightarrow$ chip space $\leftrightarrow$ component selection $\leftrightarrow$ workload

Another Constraint: Power

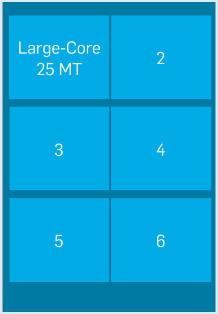
- Can use transistors for either logic or caches.



→ Power consumptions limits amount of logic that can be put on chip.

Source: Borkar and Chien. The Future of Microprocessors. *CACM 2011.*

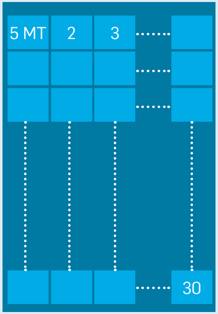
Heterogeneous Hardware



Large-Core Homogeneous

Large-core throughput	1
Small-core throughput	
Total throughput	6

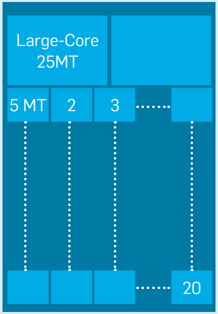
(a)



Small-Core Homogeneous

Large-core throughput	
Small-core throughput	Pollack's Rule $(5/25)^{0.5}=0.45$
Total throughput	13

(b)



Small-Core Homogeneous

Large-core throughput	1
Small-core throughput	Pollack's Rule $(5/25)^{0.5}=0.45$
Total throughput	11

(c)

Field-Programmable Gate Arrays

Field-Programmable Gate Arrays (FPGAs) are yet-another point in the design space.

- "Programmable hardware."
- Make (some) design decisions **after** chip fabrication.

Promises of FPGA technology:

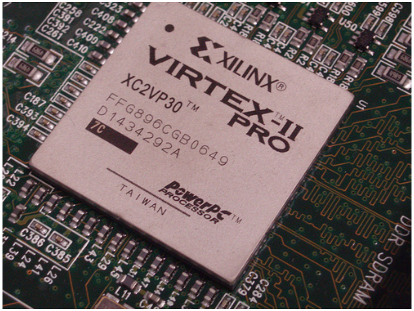
$\rightsquigarrow$ Build application-/workload-specific circuit.

$\rightsquigarrow$ Spend chip space only on functionality that you really need.

$\rightsquigarrow$ Tune for throughput, latency, energy consumption, . . .

$\rightsquigarrow$ Overcome limits of general-purpose hardware with regard to task at hand (*e.g.*, I/O limits).

Field-Programmable Gate Arrays



- An **array** of logic **gates**
- Functionality fully **programmable**
- Re-programmable after deployment ("in the **field**")

→ "programmable hardware"

- FPGAs can be configured to implement **any** logic circuit.
- Complexity bound by available **chip space**.
  - → Obviously, the effective chip space is less than in custom-fabricated chips (ASICs).

Field-Programmable Gate Arrays

FPGAs are **not** instruction set processors.

$\rightarrow$ Cannot run (sequential) programs.

One **could** build an instruction set processor using an FPGA.

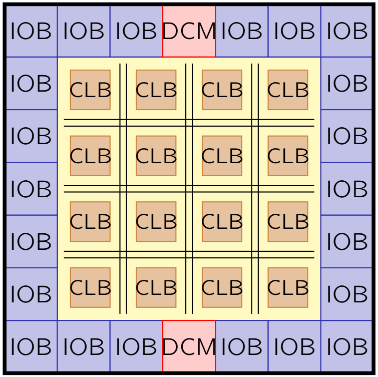
$\rightarrow$ Bad idea. FPGA $\approx 14\times$ slower than equivalent ASIC.

$\rightarrow$ If you want an instruction set processor, buy an instruction set processor.

Instead:

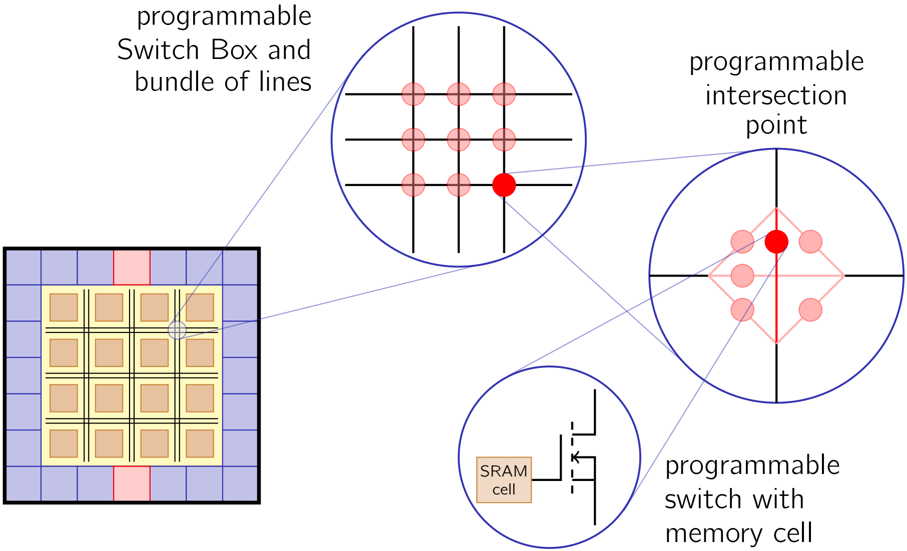
- Create arbitrary logic circuits.
- Hardware description language (HDL).

Basic FPGA Architecture

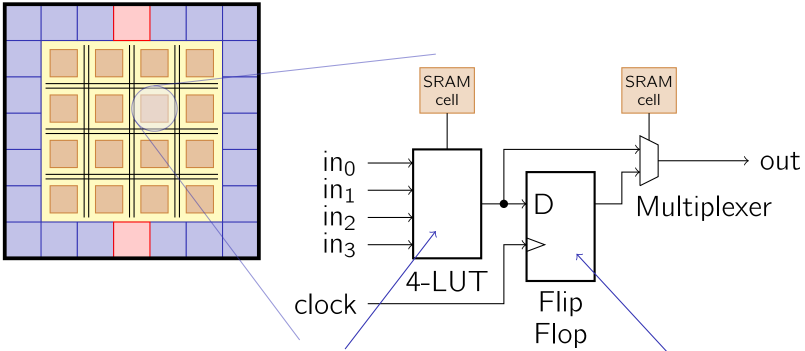


- chip layout: 2D array
- Components
  - CLB: Configurable Logic Block ("logic gates")
  - IOB: Input/Output Block
  - DCM: Digital Clock Manager
- Interconnect Network
  - signal lines
  - configurable switch boxes

Signal Routing


programmable Switch Box and bundle of lines
programmable intersection point
SRAM cell
programmable switch with memory cell

Configurable Logic Block (CLB)



implements
$\{0,1\}^4 \rightarrow \{0,1\}$
function

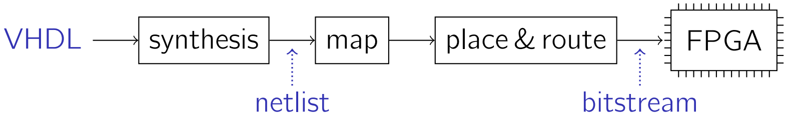
stores a
single bit

Programming FPGAs

Programming is usually done using a **hardware description language**.

- *E.g.*, **VHDL**[12], Verilog
- High-level circuit description

Circuit description is compiled into a **bitstream**, then loaded into SRAM cells on the FPGA:



[12] VHSIC Hardware Description language

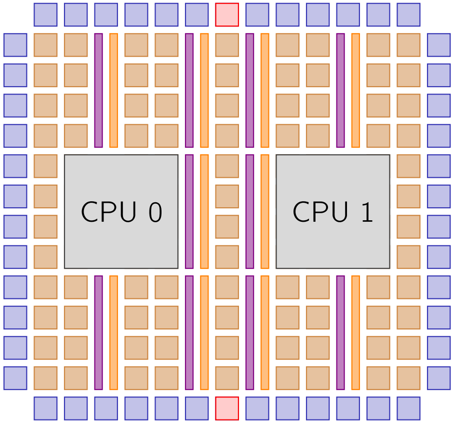
Example: VHDL

HDLs enable programming language-like descriptions of hardware circuits.

```
architecture Behavioral of compare is
begin
  process (A, B)
  begin
    if ( A = B ) then
      C <= ’1’;
    else
      C <= ’0’;
    end if;
  end process;
end Behavioral;
```

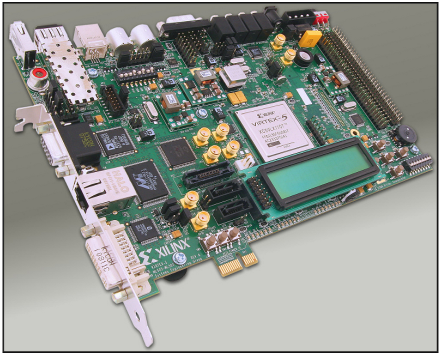
VHDL can be synthesized, but also executed in software (**simulation**).

Real-World Hardware



- Simplified Virtex-5 XC5VFXxxxT floor plan
- Frequently used high-level components are provided in discrete silicon
- BlockRAM (BRAM): set of blocks that each store up 36 kbits of data
- DSP48 slices: 25x18-bit multipliers followed by a 48-bit accumulator
- CPU: two full embedded PowerPC 440 cores

Development Board with Virtex-5 FPGA



source: Xilinx Inc., ML50x Evaluation Platform. User Guide.

	Virtex-5 XC5VLX110T
Lookup Tables (LUTs)	69,120
Block RAM (kbit)	5,328
DSP48 Slices	64
PowerPC Cores	0
max. clock speed	$\approx$ 450 MHz
release year	2006

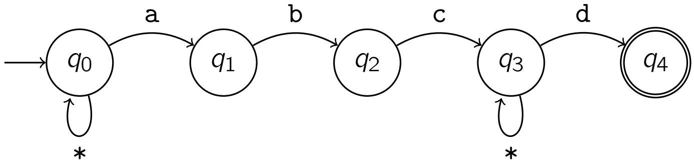
Low-level speed of configurable gates is slower than in custom-fabricated chips (clock frequencies: $\sim$100 MHz).
$\rightarrow$ Compensate with efficient circuit for problem at hand.

State Machines

The key asset of FPGAs is their inherent **parallelism**.

- Chip areas naturally operate independently and in parallel.

For example, consider **finite-state automata**.



$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[r, bend left, "a"] \arrow[loop below, "*"] & q_1 \arrow[r, bend left, "b"] & q_2 \arrow[r, bend left, "c"] & q_3 \arrow[r, bend left, "d"] \arrow[loop below, "*"] & q_4
\end{tikzcd}$$

$\rightarrow$ non-deterministic automaton for `.*abc.*d`

State Machines

✎ **How would you implement an automaton in software?**

Problems with state machine implementations in software:

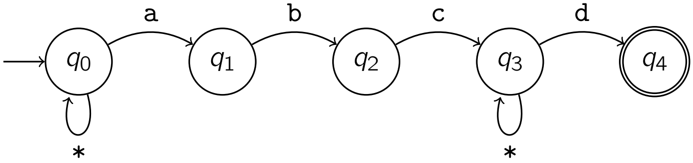
- In **non-deterministic automata**, several states can be active at a time, which requires **iterative** execution on sequential hardware.
- **Deterministic automata** avoid this problem at the expense of a significantly higher **state count**.

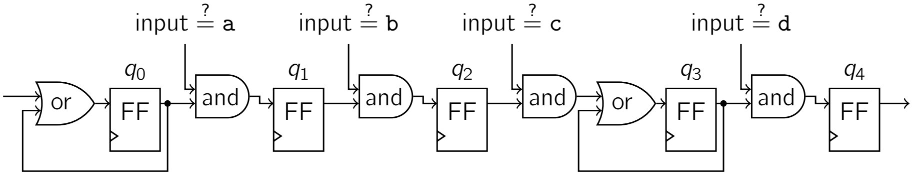
State Machines in Hardware

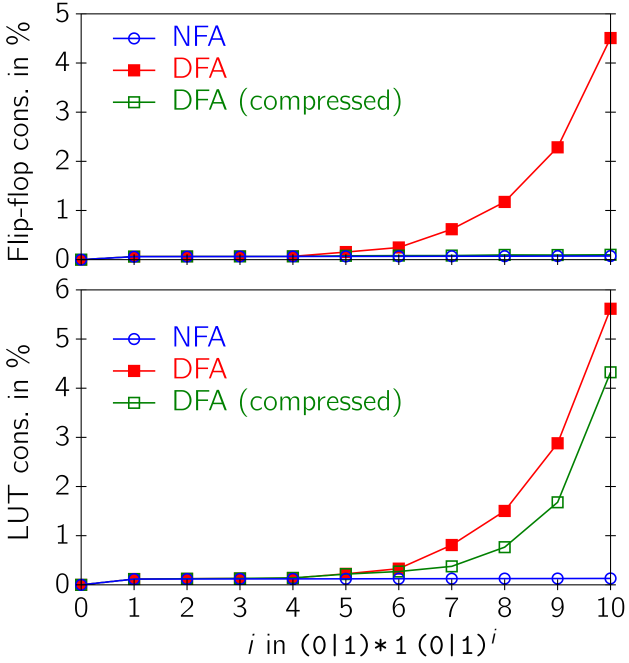
Automata can be translated mechanically into hardware circuits.

- each **state** $\rightarrow$ **flip-flop**
  (A flip-flop holds a single bit of information. Just the right amount to keep the ‘active’/‘not active’ information.)
- **transitions**:
  - $\rightarrow$ **signals** (“wires”) between states
  - **conditioned** on current input symbol ($\leadsto$ ‘and’ gate)
  - **multiple sources** for one flip-flop input $\rightarrow$ **‘or’ gate**.

State Machines in Hardware






NFA
DFA
DFA (compressed)
Flip-flop cons. in %
LUT cons. in %
0
1
2
3
4
5
6
7
8
9
10
i in (0|1)*1(0|1)^i

Use Case: XML Projection

Example:

```
for $i in //regions//item
  return <item>
           { $i/name }
           <num-categories>
             { count ($i/incategory) }
           </num-categories>
         </item>
```

Projection paths:

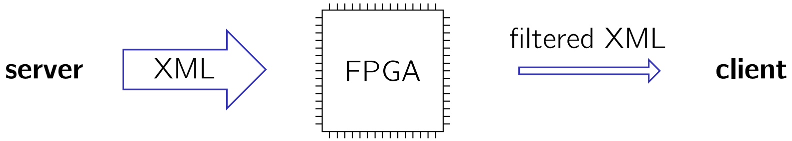
keep descendants

```
{ //regions//item,
  //regions//item/name #,
  //regions//item/incategory }
```

Challenge: Avoid explicit synthesis for each query.

Advantage: FPGA System Integration

Here: In-network filtering



In general: FPGA **in the data path**.

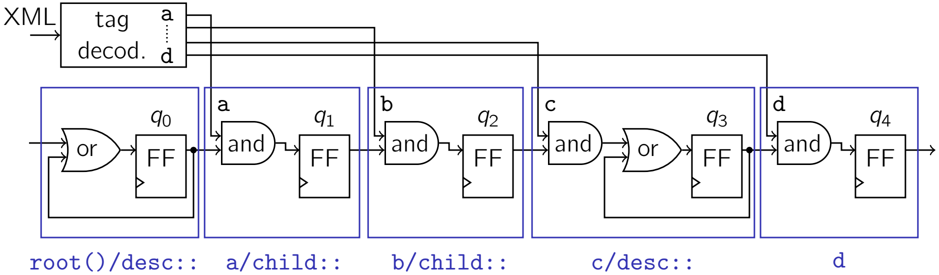
- disk → CPU
- memory → CPU
- . . .

XPath → Finite State Automata

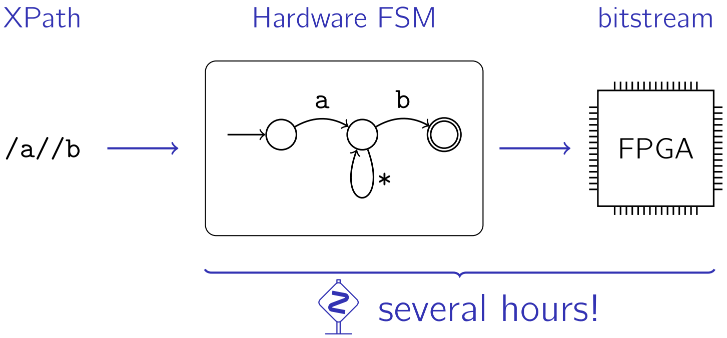
Automaton for `//a/b/c//d`:

$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[loop below, "*"] \arrow[r, bend left, "a"] & q_1 \arrow[r, bend left, "b"] & q_2 \arrow[r, bend left, "c"] & q_3 \arrow[loop below, "*"] \arrow[r, bend left, "d"] & \boxed{q_4}
\end{tikzcd}$$

In hardware: (see also earlier slides)

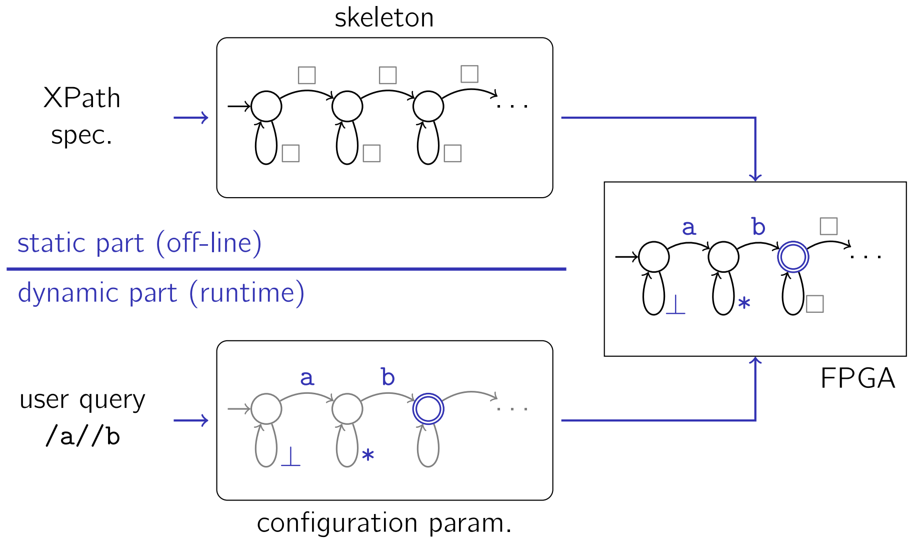


Compilation to Hardware



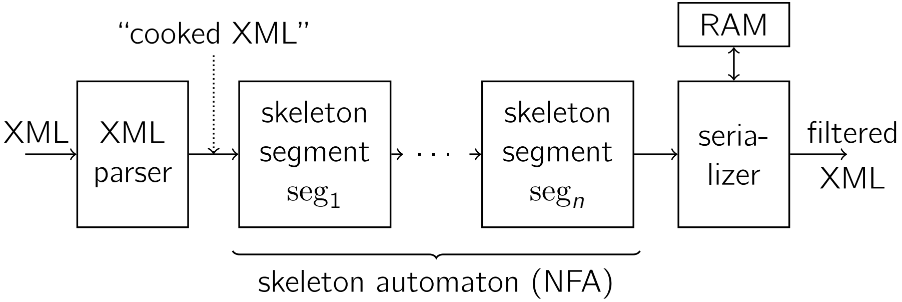
Skeleton Automaton

Separate the **difficult** parts from the **latency-critical**



Skeleton Automaton

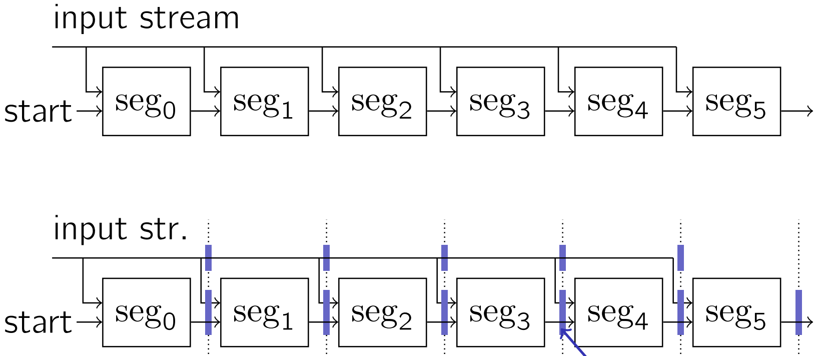
Thus: Build skeleton automaton that can be **parameterized** to implement **any** projection query.



Intuitively:

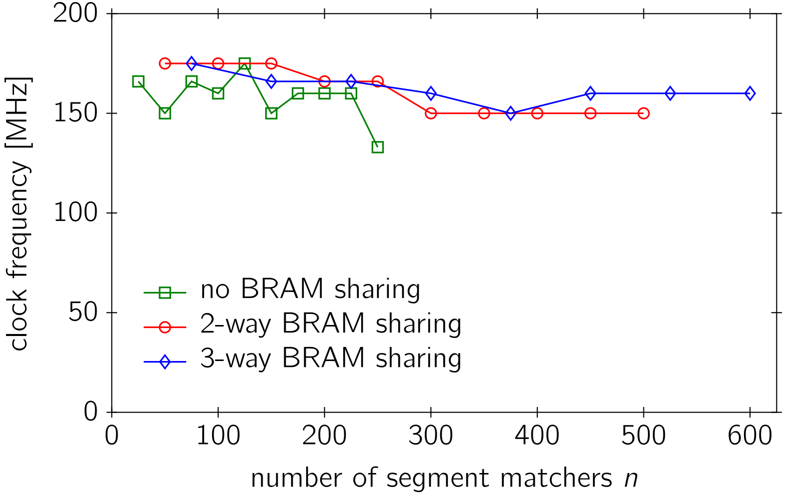
- Runtime-configuration determines presence of $\circlearrowleft *$.

Pipelining

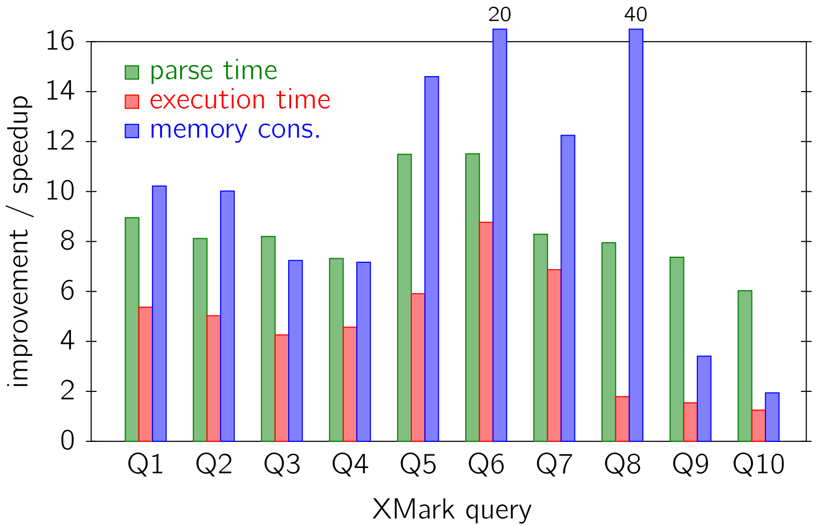


→ Side effect: Can support `self` and `descendant-or-self` axes.

Scalability



Application Speedup



↗ Jens Teubner, Louis Woods, and Chongling Nie. Skeleton Automata for FPGAs: Reconfiguring without Reconstructing. *SIGMOD 2012*.