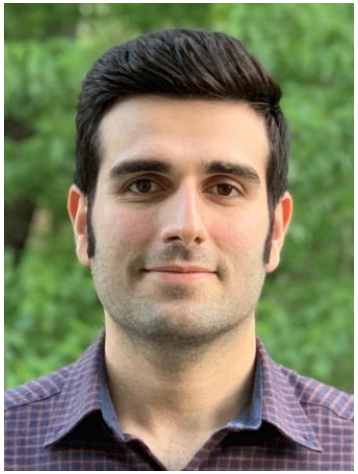
Implementing and Benchmarking Seven Round 2 Lattice-Based Key Encapsulation Mechanisms Using a Software/Hardware Codesign Approach

Farnoud Farahmand,
Viet B. Dang,
Michał Andrzejczak,
Kris Gaj

George Mason University

Co-Authors

GMU PhD Students



Farnoud Farahmand

Viet Ba Dang

Visiting Scholar

Michał Andrzejczak

Military University of Technology in Warsaw, Poland

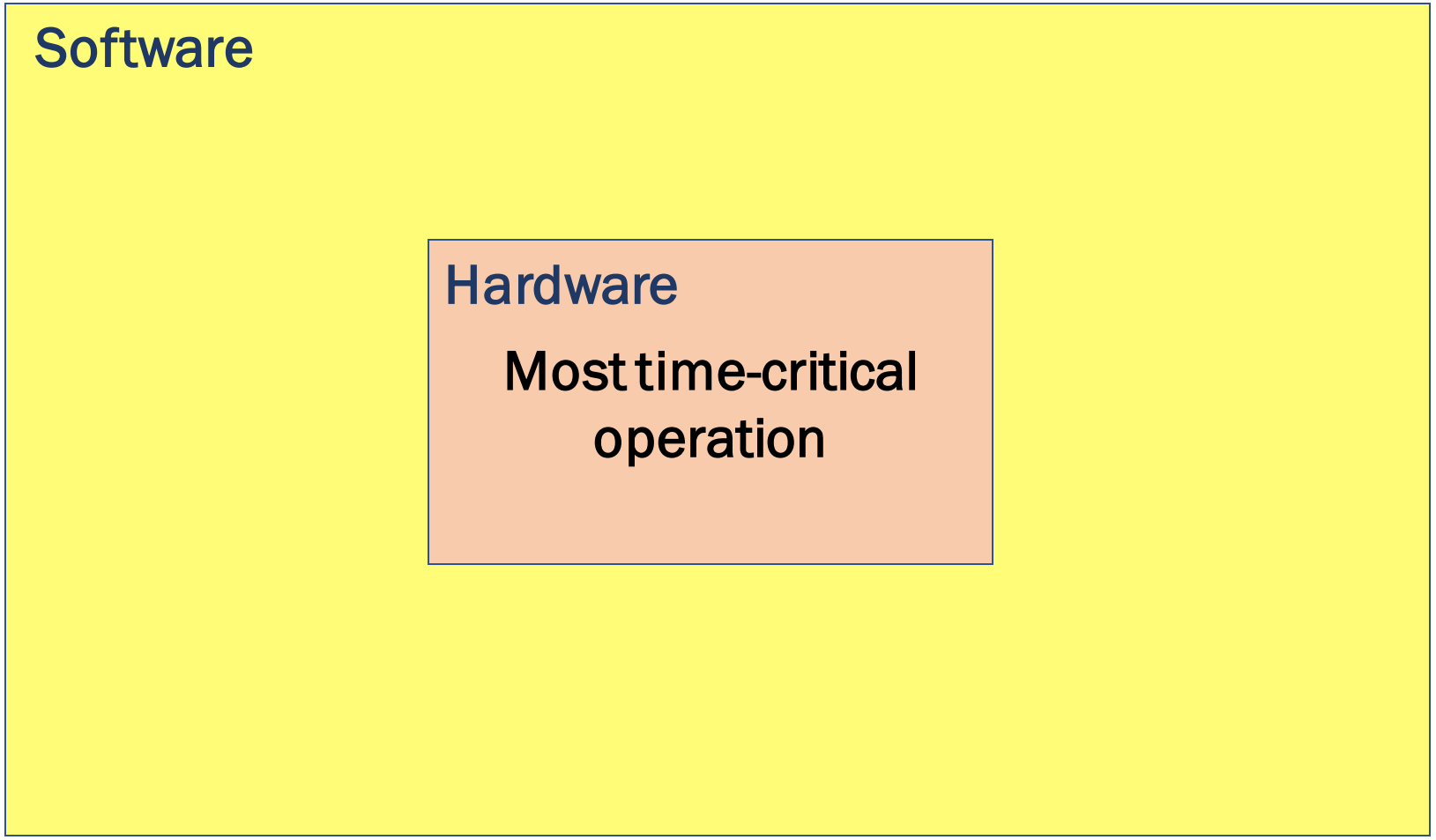
Hardware Benchmarking

Round 2 Candidates in Hardware

	#Round 2 candidates	Implemented in hardware	Percentage
AES	5	5	100%
SHA-3	14	14	100%
CAESAR	29	28	97%
PQC	26	?	?

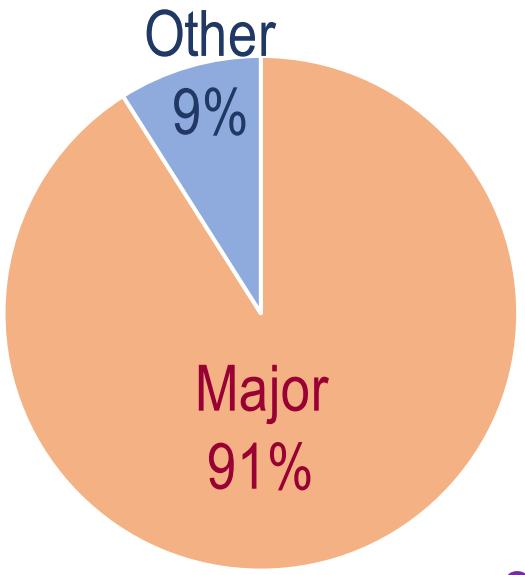

Software/Hardware
Codesign

Software/Hardware Codesign

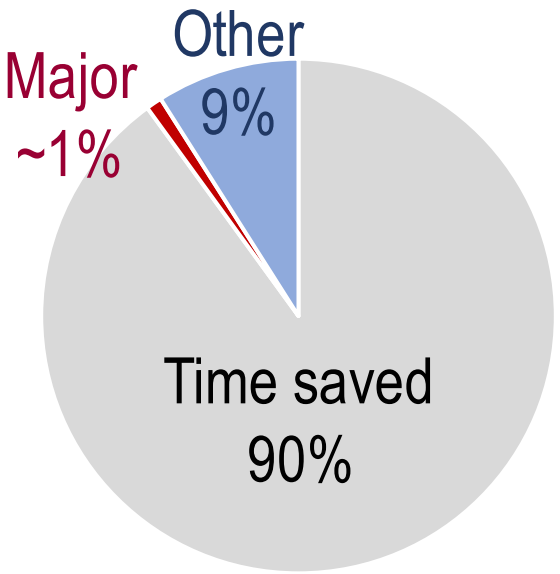


SW/HW Codesign: Motivational Example 1

Software



Software/Hardware



speed-up ≥ 100

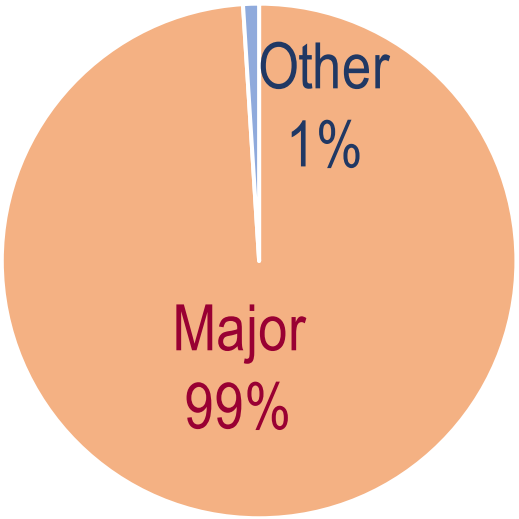
91% major operation(s)
9% other operations

~1% major operation(s) in HW
9% other operations in SW

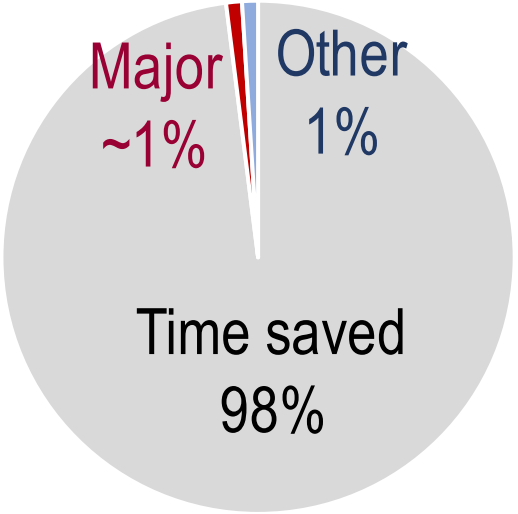
Total Speed-Up ≥ 10

SW/HW Codesign: Motivational Example 2

Software



Software/Hardware



speed-up ≥ 100

99% major operation(s)
1% other operations

~1% major operation(s) in HW
1% other operations in SW

Total Speed-Up ≥ 50

SW/HW Codesign: Advantages

- Focus on a few major operations, known to be easily parallelizable
  - much shorter development time (at least by a factor of 10)
  - guaranteed substantial speed-up
  - high-flexibility to changes in other operations (such as candidate tweaks)
- Insight regarding performance of future instruction set extensions of modern microprocessors
- Possibility of implementing multiple candidates by the same research group, eliminating the influence of different
  - design skills
  - operation subset (e.g., including or excluding key generation)
  - interface & protocol
  - optimization target
  - platform

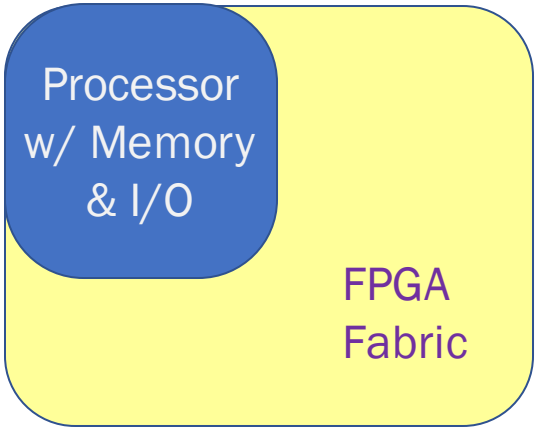
SW/HW Codesign: Potential Pitfalls

- Performance & ranking may strongly depend on
  - A. features of a particular platform
    - Software/hardware interface
    - Support for cache coherency
    - Differences in max. clock frequency
  - B. selected hardware/software partitioning
  - C. optimization of an underlying software implementation
- Limited insight on ranking of purely hardware implementations

First step, not the ultimate solution!

Two Major Types of Platforms

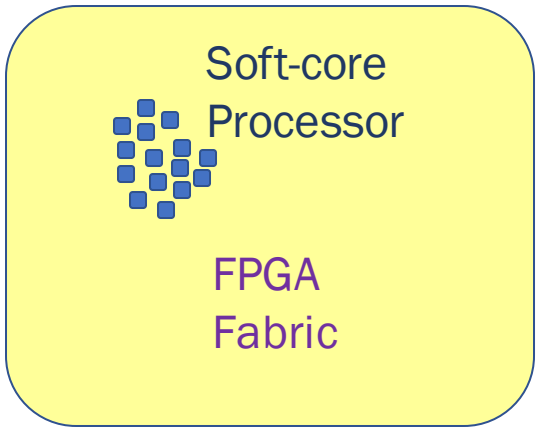
FPGA Fabric & Hard-core Processors



Examples:

- Xilinx Zynq 7000 System on Chip (SoC)
- Xilinx Zynq UltraScale+ MPSoC
- Intel Arria 10 SoC FPGAs
- Intel Stratix 10 SoC FPGAs

FPGA Fabric, including Soft-core Processors



Examples:

Xilinx Virtex UltraScale+ FPGAs

Intel Stratix 10 FPGAs, including

- Xilinx MicroBlaze
- Intel Nios II
- RISC-V, originally UC Berkeley

Two Major Types of Platform

Feature	FPGA Fabric and Hard-core Processor	FPGA Fabric with Soft-core Processor
Processor	ARM	MicroBlaze, NIOS II, RISC-V, etc.
Clock frequency	>1 GHz	max. 200-450 MHz
Portability	similar FPGA SoCs	various FPGAs, FPGA SoCs, and ASICs
Hardware accelerators	Yes	Yes
Instruction set extensions	No	Yes
Ease of design (methodology, tools, OS support)	Easy	Dependent on a particular soft-core processor and tool chain

Xilinx Zynq UltraScale+ MPSoC

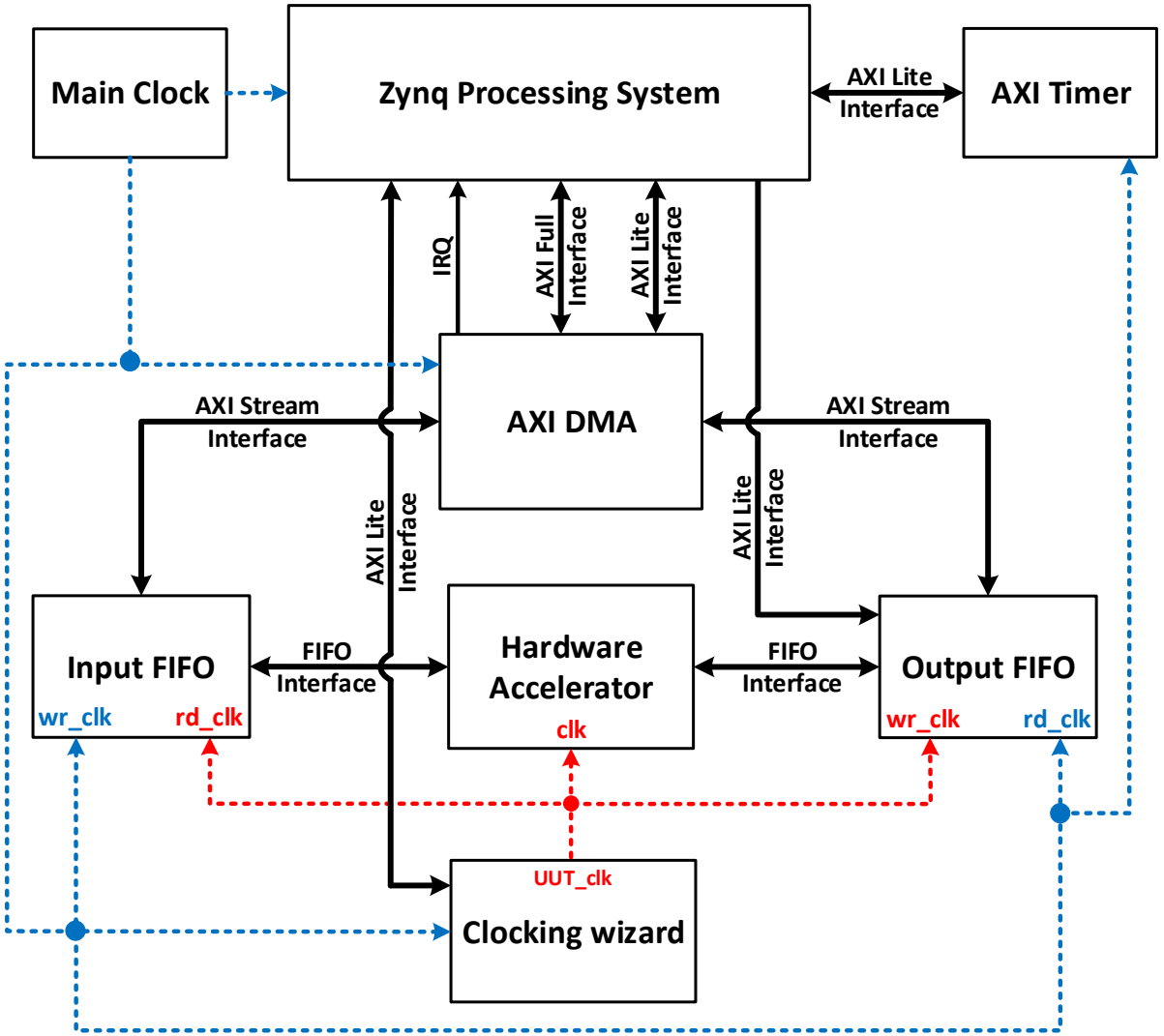
1.2 GHz ARM Cortex-A53 + UltraScale+ FPGA logic

Choice of a Platform for Benchmarking

Embedded Processor:	FPGA Architecture:
In NIST presentations to date:	
ARM Cortex-M4	Artix-7
Our recommendation:	
ARM Cortex-A53	UltraScale+

- No FPGA SoC with ARM Cortex-M4 and Artix-7 on a single chip
- Cortex-M4 and Artix-7 more suitable for lightweight designs, Cortex-A53 and UltraScale+ for high performance
- Zynq UltraScale+:
  - capability to compare SW/HW implementations with fully-SW and fully-HW implementations realized using the same chip
  - likely in use in the first years of the new standard deployments

Experimental Setup



All elements located on a single chip

Code Release

- Full Code & Configuration of the Experimental Setup
- Software/Hardware Codesign of Round 1 NTRUEncrypt

to be made available at

https://cryptography.gmu.edu/athena

under PQC

by August 31, 2019

Our Case Study

SW/HW Codesign: Case Study

7 IND-CCA*-secure Lattice-Based Key Encapsulation Mechanisms (KEMs)
representing
5 NIST PQC Round 2 Submissions

LWE (Learning with Error)-based:

FrodoKEM

RLWR (Ring Learning with Rounding)-based:

Round5

Module-LWR-based:

Saber

NTRU-based:

NTRU

- NTRU-HPS
- NTRU-HRSS

NTRU Prime

- Streamlined NTRU Prime
- NTRU LPRime

* IND-CCA = with Indistinguishability under Chosen Ciphertext Attack

SW/HW Partitioning

Top candidates for offloading to hardware

From profiling:

- Large percentage of the execution time
- Small number of function calls

From manual analysis of the code:

- Small size of inputs and outputs
- Potential for combining with neighboring functions

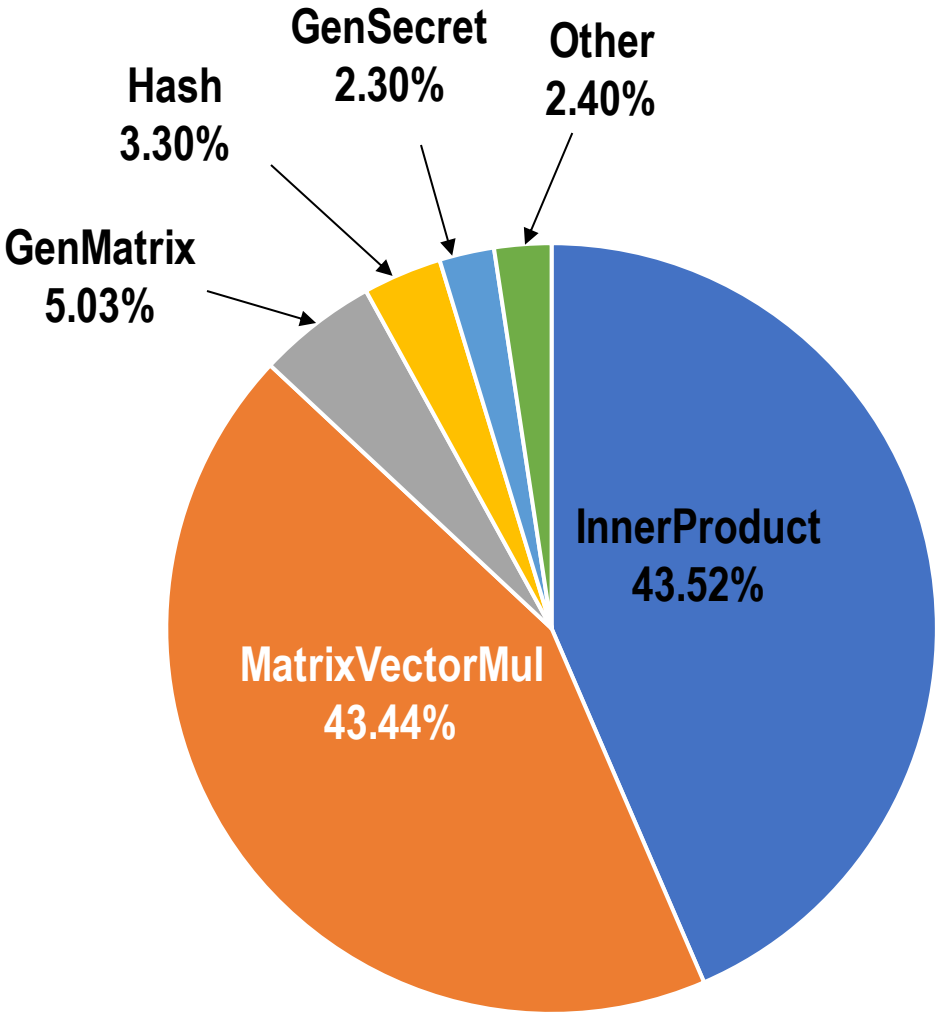
From knowledge of operations and concurrent computing:

- High potential for parallelization

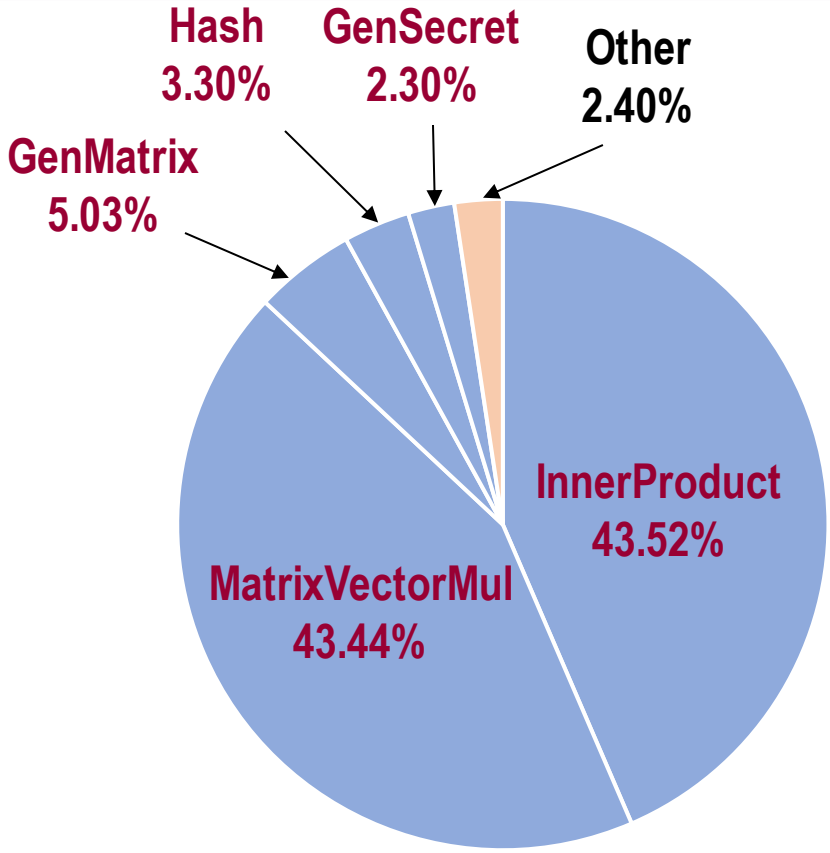
Operations Offloaded to Hardware

- Major arithmetic operations
  - Polynomial multiplications
  - Matrix-by-vector multiplications
  - Vector-by-vector multiplications
- All hash-based operations
  - (c)SHAKE128, (c)SHAKE256
  - SHA3-256, SHA3-512

Example: LightSaber Decapsulation

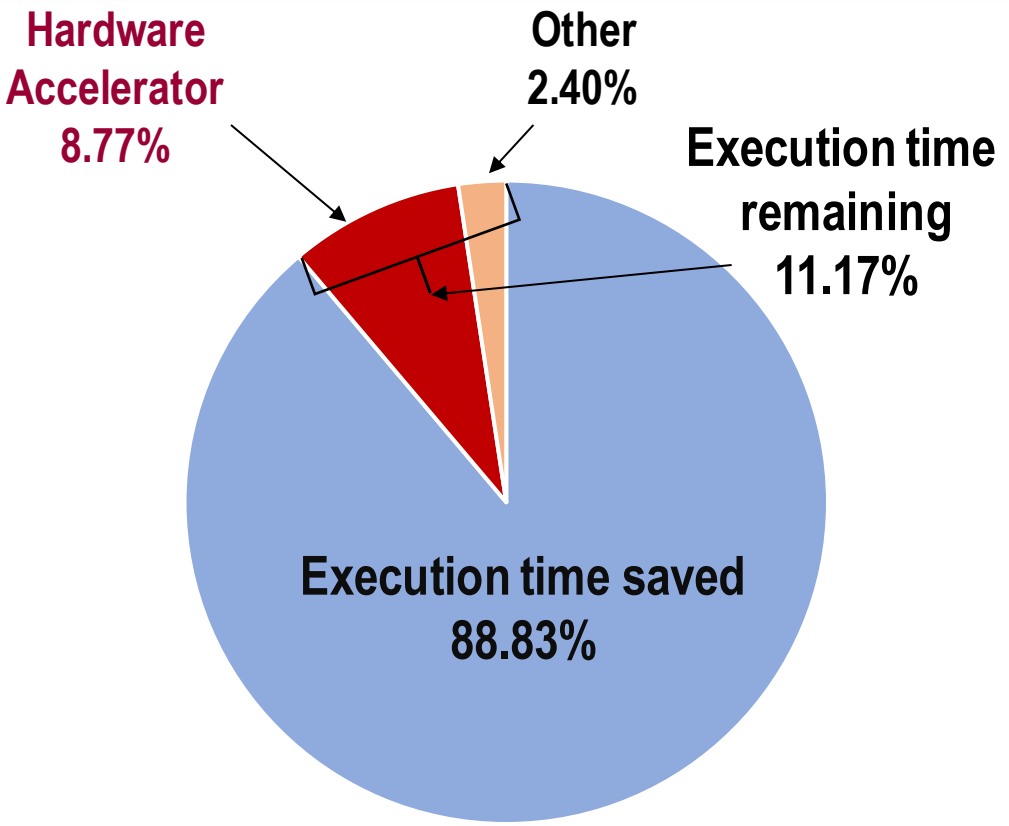


LightSaber Decapsulation



Execution time of functions to be moved to hardware
97.60%

Execution time of functions remaining in software
2.40%



Accelerator Speed-Up = 97.60/8.77=11.1

Total Speed-Up = 100/11.17=9.0

Tentative Results

Software Implementations Used

FrodoKEM, NTRU-HPS, NTRU-HRSS, Saber:

Round 2 submission packages – Optimized_Implementation

Round5:

https://github.com/r5embed/r5embed (2019-07-28)

Streamlined NTRU Prime, NTRU LPRime:

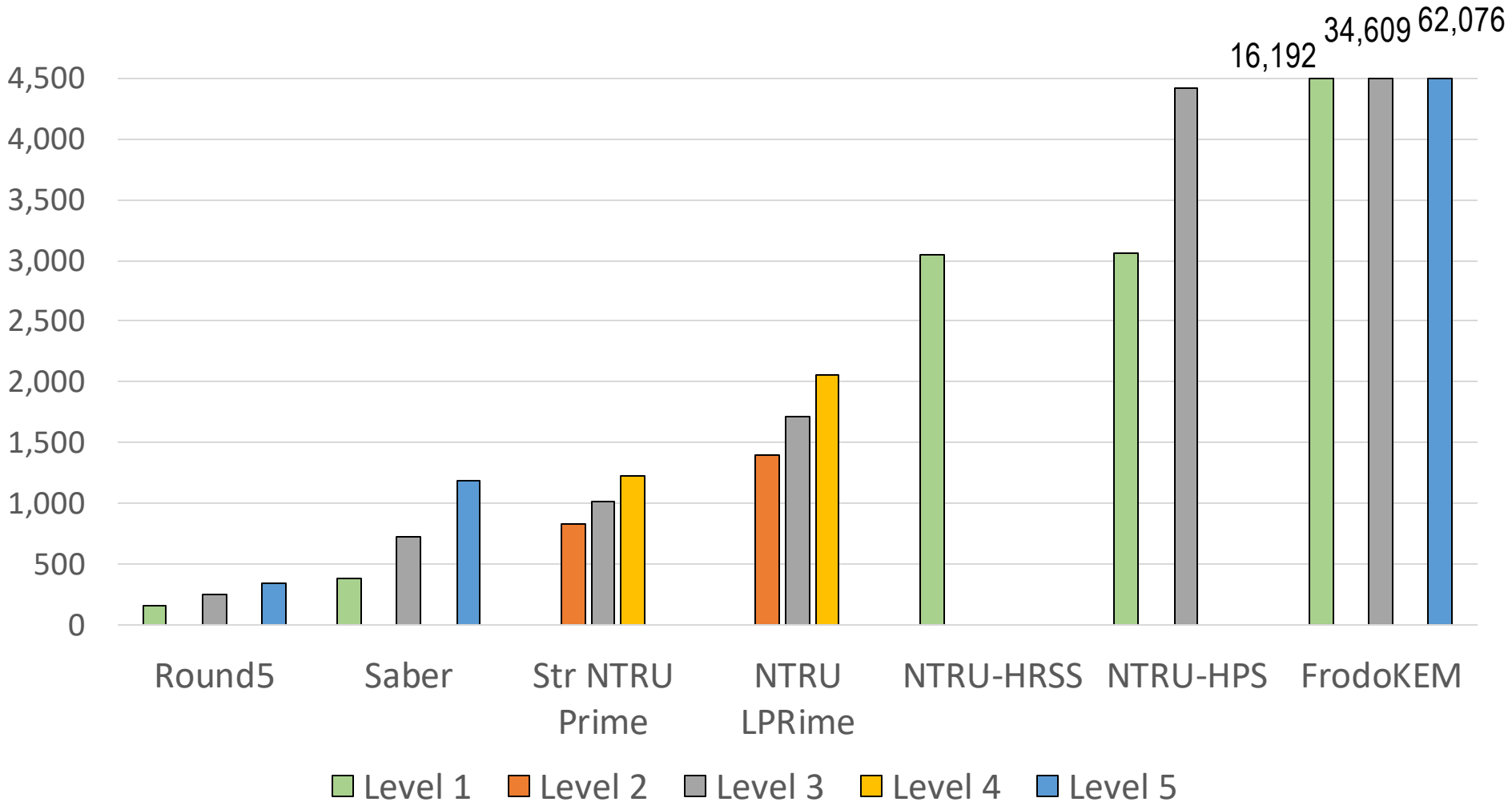
supercop-20190811 : factored

Changes made after the submission of the paper!

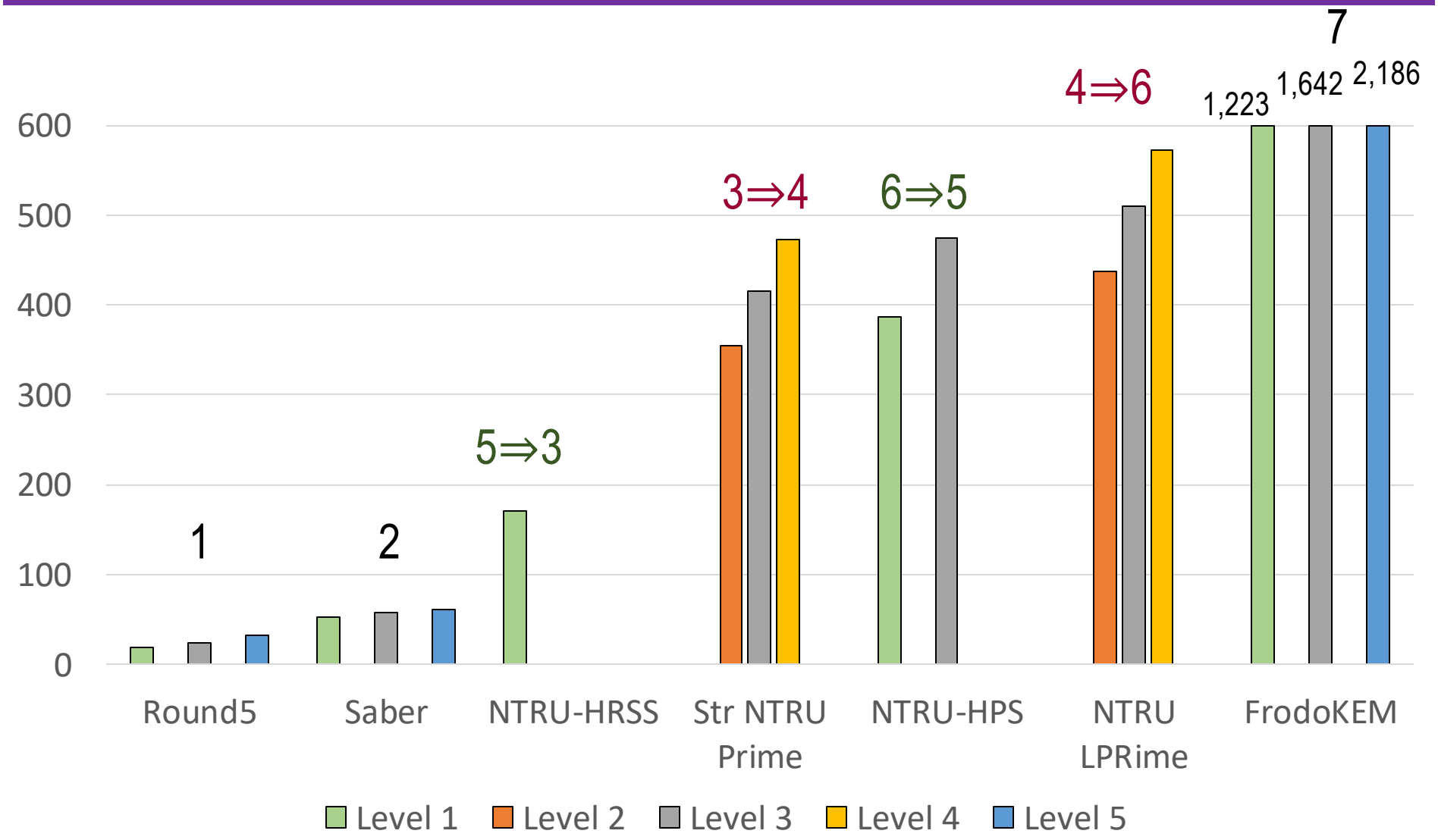
Results substantially different!

New version of the paper available on ePrint soon!

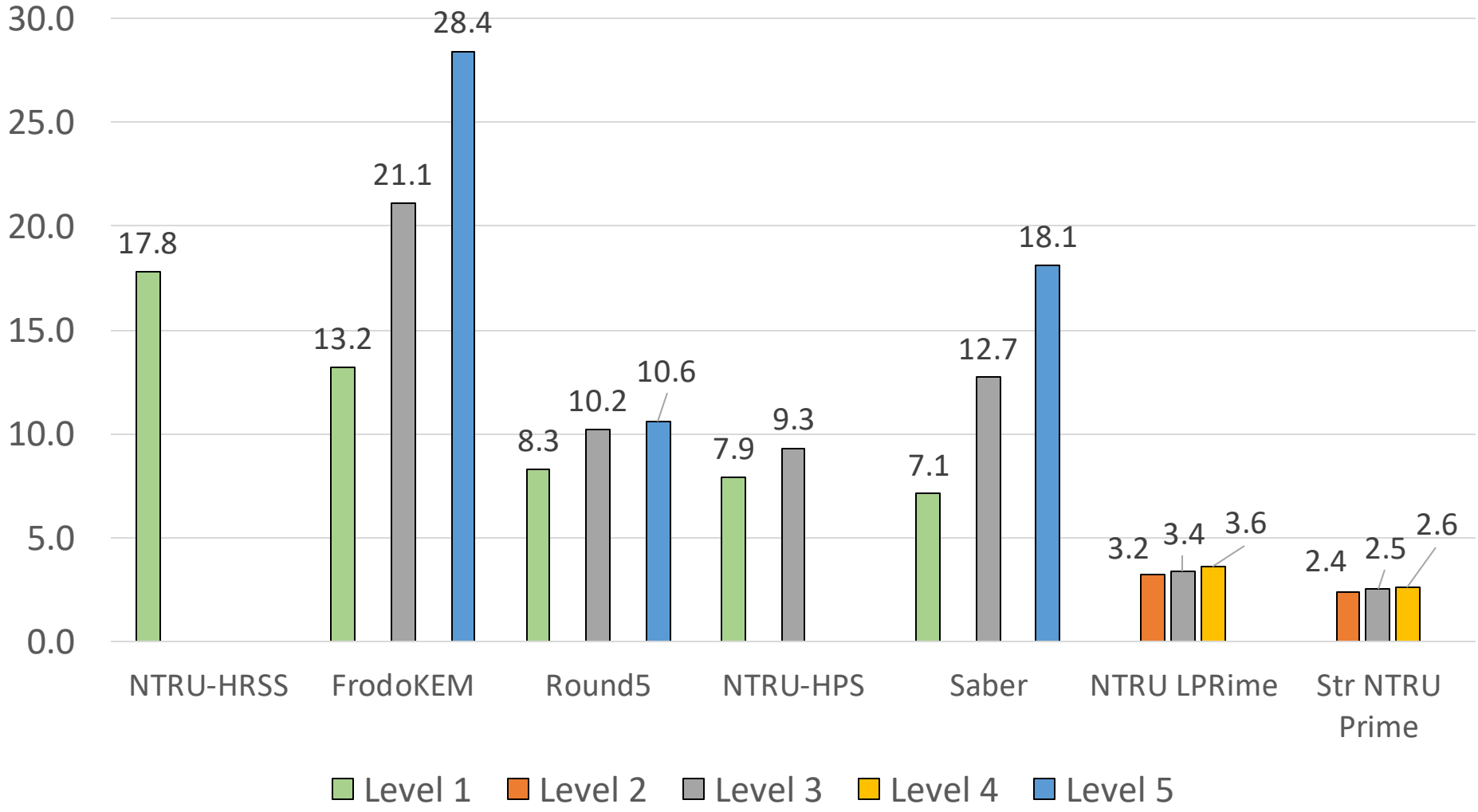
Total Execution Time in Software [μs] Encapsulation



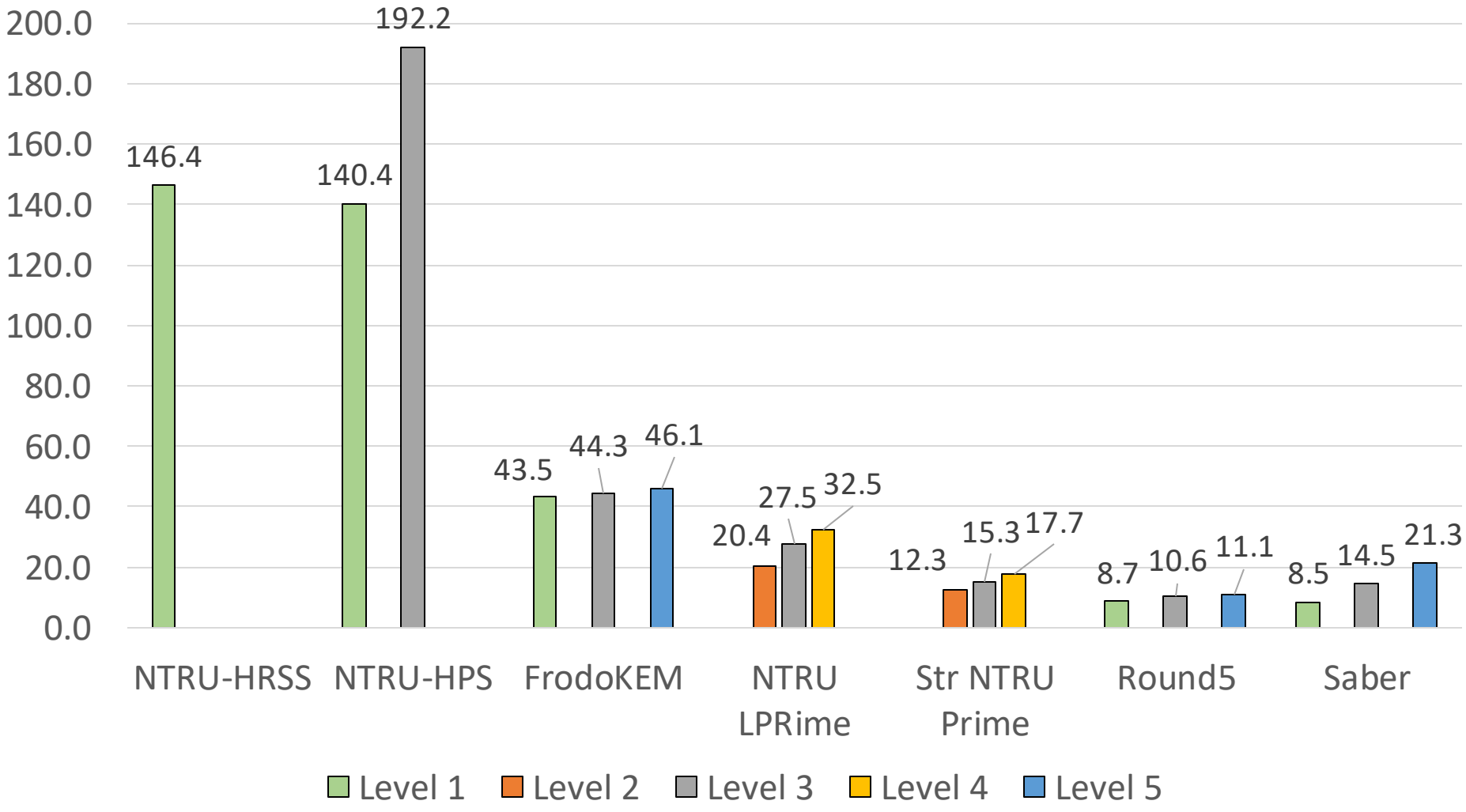
Total Execution Time in Software/Hardware [μs] Encapsulation



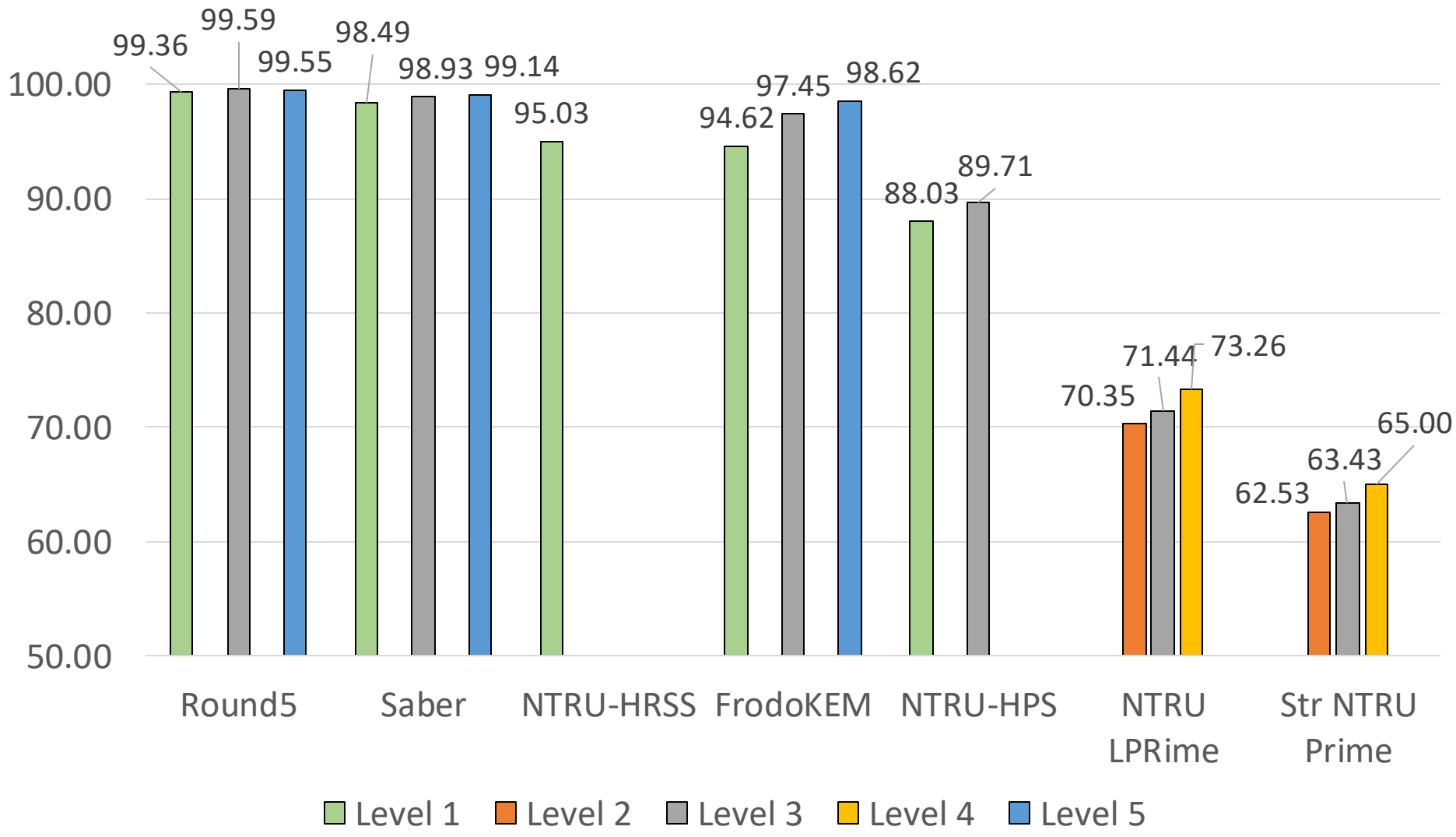
Total Speed-ups: Encapsulation



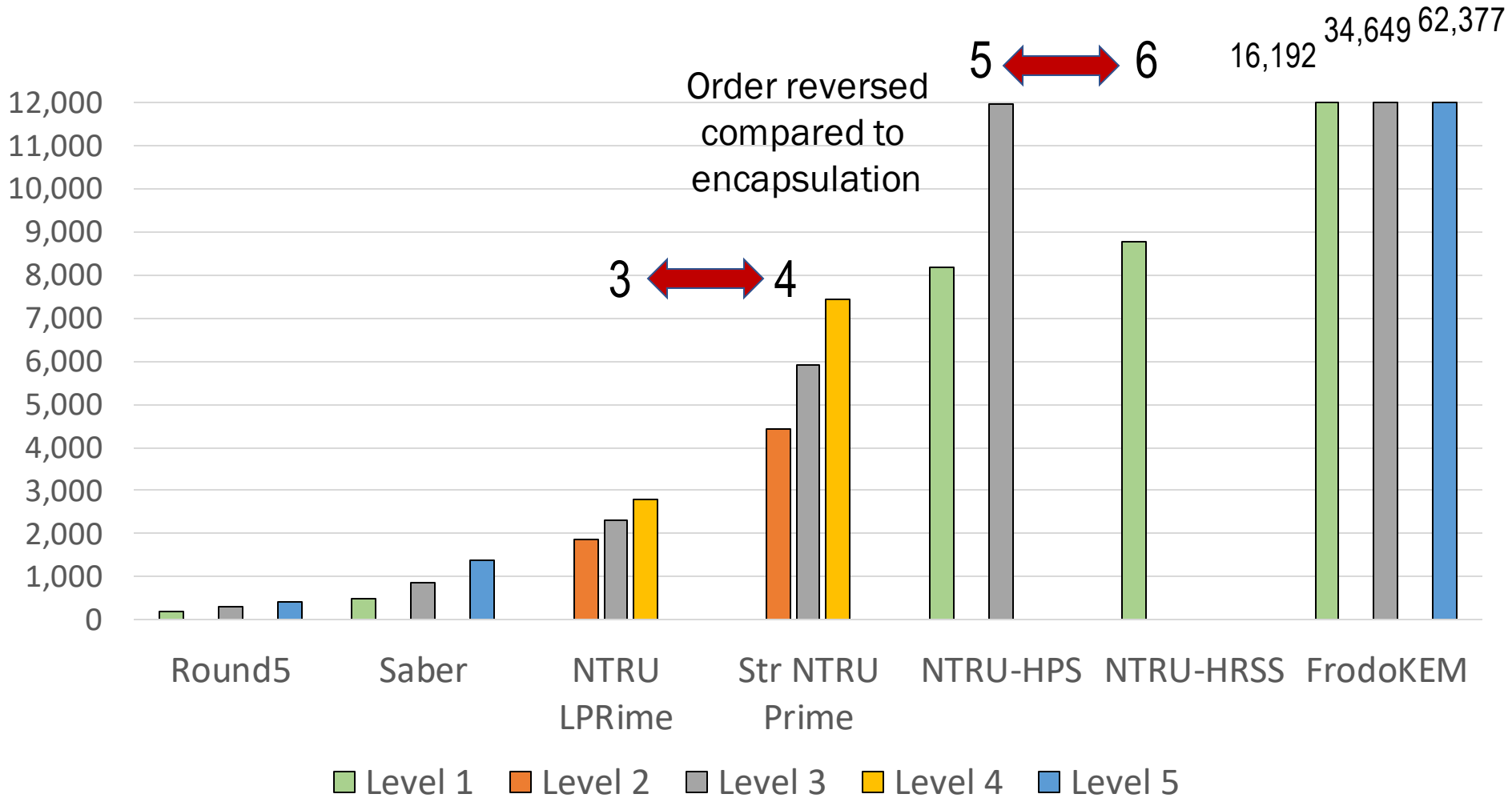
Accelerator Speed-ups: Encapsulation



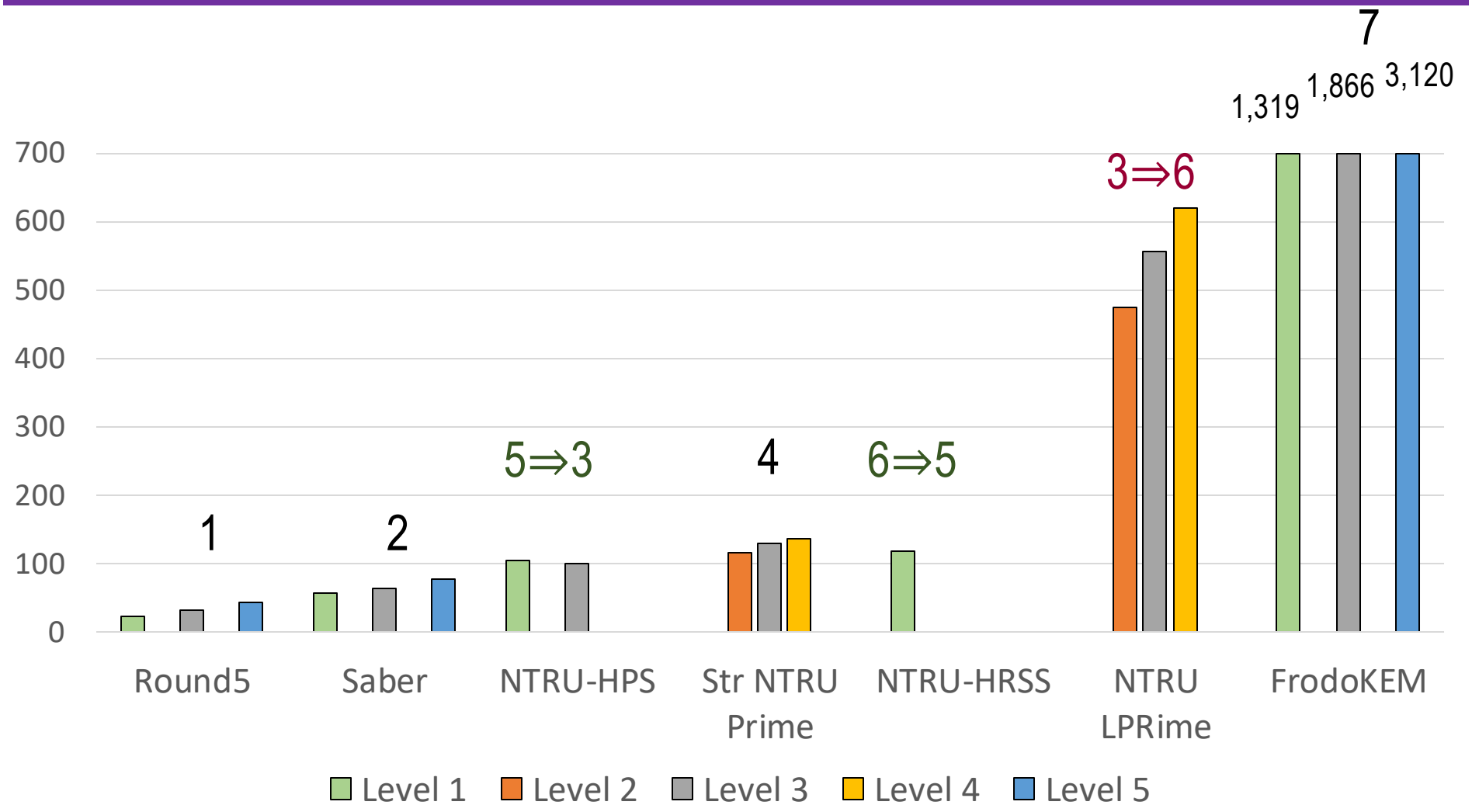
SW Part Sped up by HW[%]: Encapsulation



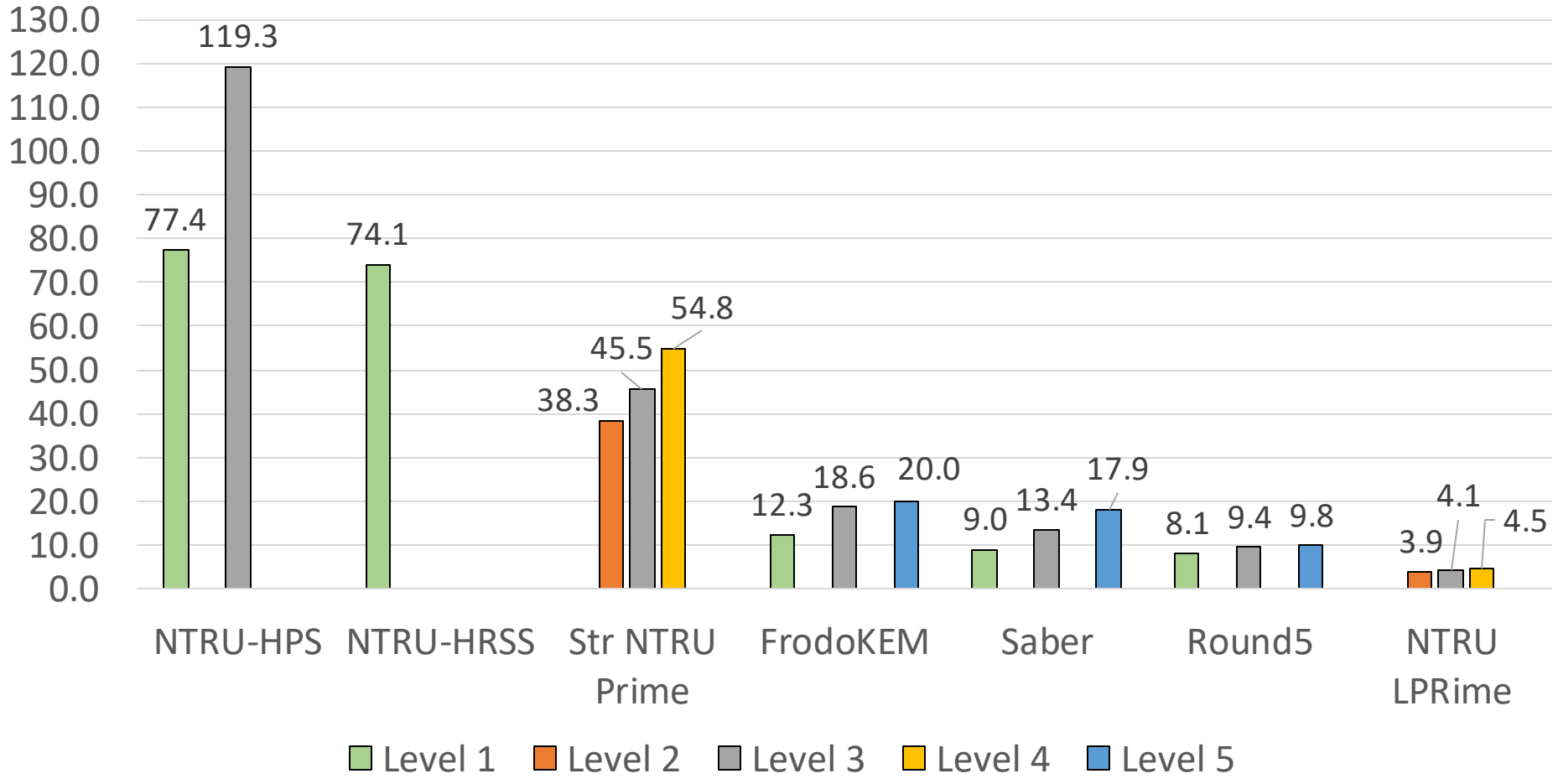
Total Execution Time in Software [μs] Decapsulation



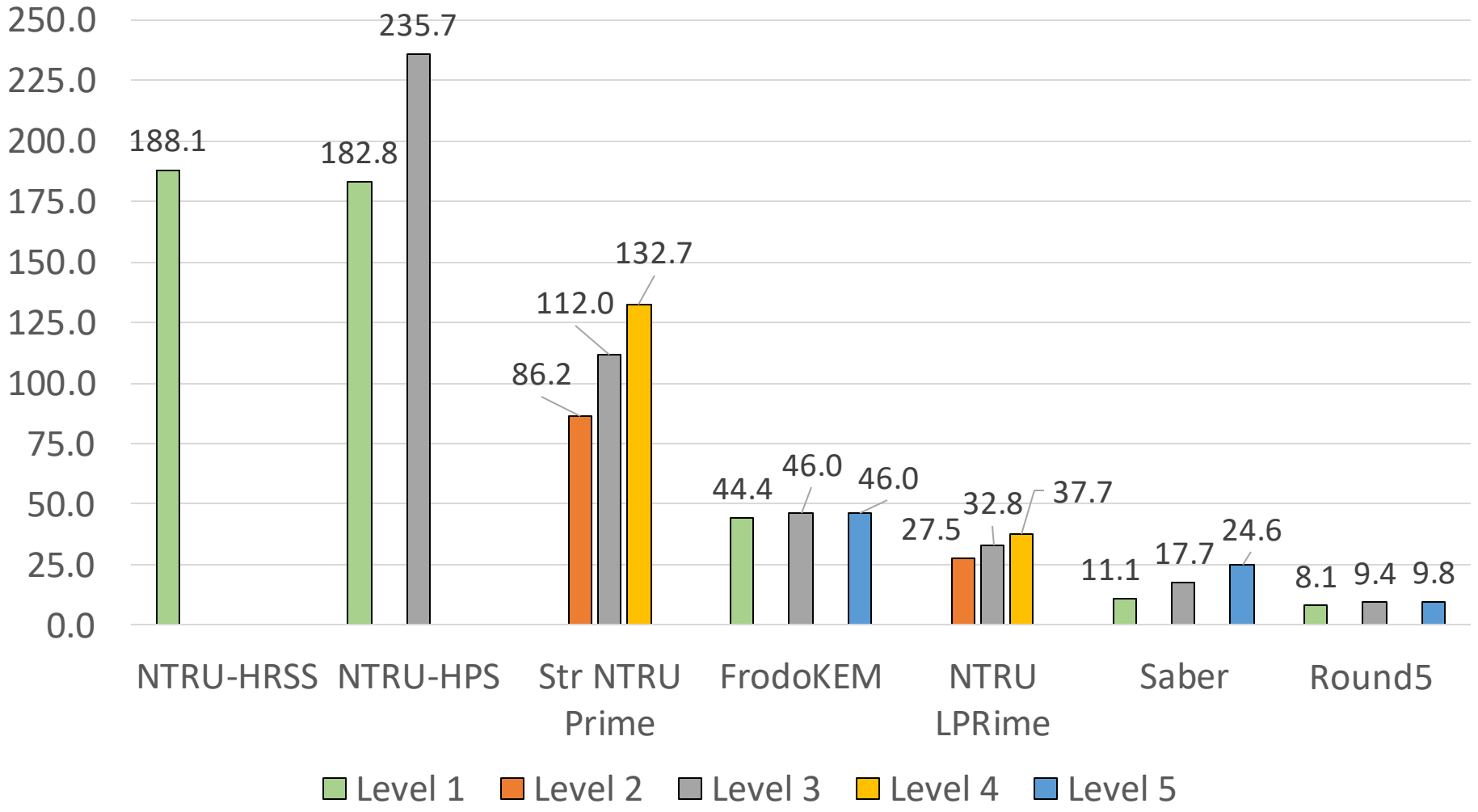
Total Execution Time in Software/Hardware [μs]: Decapsulation



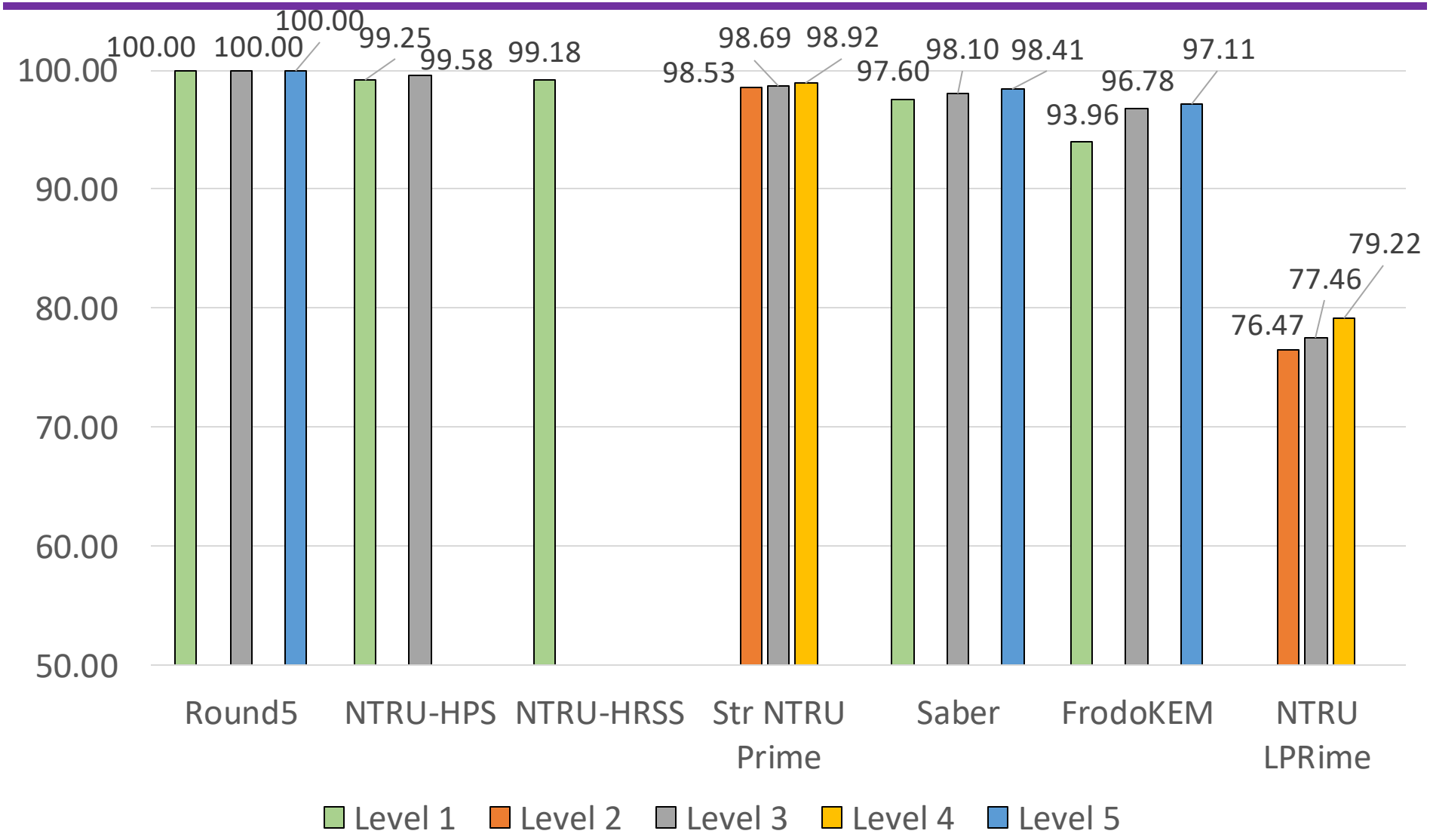
Total Speed-ups: Decapsulation



Accelerator Speed-ups: Decapsulation



SW Part Sped up by HW[%]: Decapsulation



Conclusions

- Total speed-ups
  - for encapsulation from 2.4 (Str NTRU Prime) to 28.4 (FrodoKEM)
  - for decapsulation from 3.9 (NTRU LPRime) to 119.3 (NTRU-HPS)
- Total speed-up dependent on the percentage of the software execution time taken by functions offloaded to hardware and the amount of acceleration itself
- Hardware accelerators thoroughly optimized using Register-Transfer Level design methodology
- Determining optimal software/hardware partitioning requires more work
- Ranking of the investigated candidates affected, but not dramatically changed, by hardware acceleration
- It is possible to complete similar designs for all Round 2 candidates within the evaluation period (12-18 months)
- Additional benefit: Comprehensive library of major operations in hardware

Future Work

Breadth

Current work

4 Remaining Lattice-based KEMs

3 Lattice-based Digital Signatures

7 Code-based KEMs

7 Other Candidates

More operations moved to hardware / **C code optimized for ARM Cortex-A53***

Algorithmic optimizations of software and hardware*

Hardware library of basic operations of lattice-based candidates

Hardware library of basic operations of code-based candidates

Hardware library for PQC

Full hardware implementations

Depth

***collaboration with submission teams and other groups very welcome**

Q&A

Thank You!

Questions?

Comments?

Suggestions?

CERG: http://cryptography.gmu.edu

ATHENa: http://cryptography.gmu.edu/athena

Backup

Clock Frequency & Resource Utilization

Level: Algorithm	Clock Freq [MHz]	#LUTs	#Slices	#FFs	#36kb BRAMs	#DSPs
1:FrodoKEM	402	7,213	1,186	6,647	13.5	32
1: Round5	260	55,442	10,381	82,341	0	0
1: Saber	322	12,343	1,989	11,288	3.5	256
1: NTRU-HPS	200	24,328	4,972	19,244	2.5	677
1: NTRU-HRSS	200	27,218	5,770	21,410	2.5	701
2: Str NTRU Prime	244	55,843	8,134	28,143	3.0	0
2: NTRU LPRime	244	50,911	7,874	34,050	2.0	0
Device		274,080	34,260	548,160	912	2,520
		< 21%	< 31%	< 15%	< 2%	< 28%

of total resources of the given device

Minor Modifications to C code Bare Metal vs. Linux

- No functions of OpenSSL – standalone implementations of:
  - **AES:** Optimized ANSI C code for the Rijndael cipher (T-box-based) by Vincent Rijmen, Antoon Bosselaers, and Paulo Barreto https://fastcrypto.org/front/misc/rijndael-alg-fst.c
  - **SHA-3:**
    - fips202.c from SUPERCOP by Ronny Van Keer, Gilles Van Assche, Daniel J. Bernstein, and Peter Schwabe (for all candidates other than Round5)
    - r5_xof_shake.c by Markku-Juhani O. Saarinen and keccak1600.c from SUPERCOP, by the same authors as fips202.c (for Round5)
  - **randombytes():** based on SHAKE rather than AES in NTRU-HPS, NTRU-HRSS, and Streamlined NTRU Prime
- No support for SUPERCOP scripts

randombytes()

- Function used for generating pseudorandom byte sequences
- The implementation vary among various benchmarking studies, depending on the mode of operation (Bare Metal vs. Operating System), and availability of libraries, such as OpenSSL
- Used to different extent by implementations of various candidates

Algorithm	#Calls	#Bytes (security category 1)
FrodoKEM	1	16
Round5	1	16
Saber	1	32
NTRU-HPS	1	**3211**
NTRU-HRSS	1	**1400**
Str NTRU Prime	1	**2612**
NTRU LPRime	1	32

- For 3 algorithms was sped-up over 3 times by using SHAKE128