Evaluation of Optimized PQC Standards ML-KEM and ML-DSA on Sargantana RV64GBV core

Xavier Carril[1,2,*], Emanuele Parisi[1†], Narcís Rodas[1], Raúl Gilabert[1,2], Juan Antonio Rodriguez[1], Oriol Farràs[3‡], Miquel Moretó[1,2]

[1] *Barcelona Supercomputing Center (BSC), Barcelona, Spain*
[2] *Universitat Politècnica de Catalunya (UPC), Barcelona, Spain*
[3] *Universitat Rovira i Virgili (URV), Tarragona, Spain*

Abstract

The emergence of quantum computing threatens traditional cryptographic schemes, requiring the development of post-quantum algorithms. This paper accelerates Module Lattice Key Encapsulation Mechanisms (ML-KEM) and Digital Signature Algorithms (ML-DSA), the two primary NIST standards, on the Sargantana RV64GBV core using standard RISC-V bit manipulation (B) and vector (V) extensions. We compare reference implementations against optimized assembly routines and vector and bit manipulation compiler-generated code. Hand-optimized results show BV extensions yield speedups between 3.18–4.59× for ML-KEM and ML-DSA. Without BV extensions, achieve between 2.73-3.29× speedup. Compiler-generated code lags behind hand-optimized kernels, with bit manipulation outperforming auto-vectorization.

Introduction

Digital communication underpins modern society, enabling critical information exchanges where data integrity and confidentiality are essential. However, advances in quantum computing algorithms threaten the security of existing public-key cryptography schemes [1].

In response, the National Institute of Standards and Technology (NIST) has undertaken a global initiative to standardize public-key cryptography schemes that can withstand attacks from large-scale quantum computers. Among these, Module Lattice Key Encapsulation Mechanism (ML-KEM) and Module Lattice Digital Signature Algorithm (ML-DSA) have garnered significant attention, and many accelerated implementations have been proposed [2]. While the focus has been on custom RISC-V ISA extensions [3], recent studies highlight the potential efficiency gains achievable using standard RISC-V bit manipulation (B) and vector (V) extensions [4]. Our work characterizes the performance of ML-KEM and ML-DSA reference implementation and hand-optimized version exploiting the BV extensions on the Sargantana [5] RV64GBV core. Then, we measure the performance gap between hand-tuned and compiler-emitted code when compiling the reference implementation provided by NIST with the BV extensions activated.

*Corresponding author: `xavier.carril@bsc.es`. Supported by AGAUR-FI Joan Oró grant 2024 FI-1 00520, funded by Generalitat de Catalunya (Department of Research and Universities) and the European Social Fund Plus. This work is supported by Chips JU, EU HORIZON-JU-IA, grant 101140087 (SMARTY).
†Supported by AI4S from the "Generación D" initiative (Red.es, MTDFP, C005/24-ED CV1), funded by EU NextGenerationEU funds through PRTR, partially funded by Generalitat de Catalunya [2021-SGR-00763], and by Spanish MCIU/AEI project PID2023-146511NB-I00 co-funded by EU ERDF.
‡Supported by the grant 2021 SGR 00115, by the project HERMES (INCIBE, EU NextGeneration EU/PRTR), and the project ACITHEC PID2021-124928NB-I00 (MCIN/AEI/10.13039/501100011033/FEDER, EU.)

Methodologies

The ML-KEM and ML-DSA schemes run on Sargantana, a single-issue in-order core that implements the RV64GBV ISA. It features a 7-stage pipeline with out-of-order writeback, register renaming, and a non-blocking memory unit. Sargantana also contains a 128-bit wide SIMD unit supporting RISC-V Vector Extension (RVV) version 1.0, except for LMUL>1 configurations and vector floating-point instructions. The register renaming also includes the vector configuration setting instructions (`vset{i}vl{i}`), leading to a reduced impact on performance.

ML-KEM and ML-DSA serve different purposes but share most computational primitives:

- **Keccak:** Used for polynomial sampling. It relies on SHA-3 primitives built on the Keccak-f1600 permutation function. Keccak represents >50% of the ML-KEM and ML-DSA execution cycles. The bit manipulation (B) extension enhances performance by reducing multiple instructions into a single one.
- **Number Theoretic Transform (NTT):** NTT uses butterfly operations to mix elements and apply modular arithmetic for polynomial multiplication. Vectorization enhances efficiency by executing multiple iterations simultaneously.

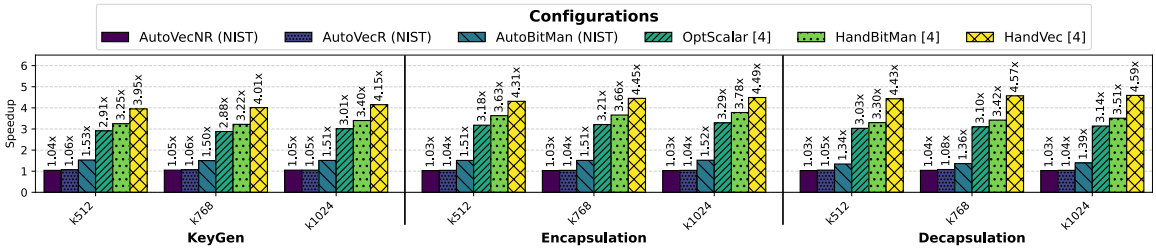

Figure 1: *ML-KEM speedup over the NIST baseline compiled for RV64G*

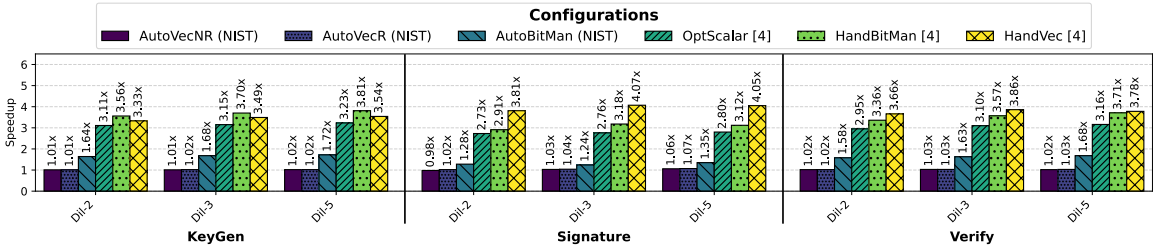

Figure 2: *ML-DSA speedup over the NIST baseline compiled for RV64G*

Results and Discussion

The PQC schemes are executed on an Alveo U55c FPGA running at a 25 MHz clock frequency.

Benchmarks are compiled using GCC 14.2, leveraging RISC-V auto-vectorization and auto-bit manipulation. For the comparison, we also execute hand-optimized code from [4], with modifications to the vector code to meet the LMUL ≤ 1 hardware constraint. Figures 1 and 2 show the speedup performance against the NIST RV64G reference cycles for all ML-KEM and ML-DSA security levels.

For each level, six configurations are tested:

1. **Auto-Vector No Renaming (AutoVecNR):** The `vsetvl` instruction flushes the pipeline.
2. **Auto-Vector with Renaming (AutoVecR):** The `vsetvl` instr. does not flush the pipeline.
3. **Auto-Bit Manipulation (AutoBitMan):** Compiler-inserted bit manipulation instructions.
4. **Optimized Scalar Code (OptScalar):** Hand-optimized scalar code from Zhang et al. [4].
5. **Hand-Based Bit Manipulation Optimization (HandBitMan):** Manual bit manipulation optimizations from Zhang et al. [4].
6. **Hand-Based Vector Opt. (HandVec):** Manually optimized vector code from Zhang et al. [4].

Our results show similar speedups across all security levels. The hand-optimized RV64IM implementation of ML-KEM and ML-DSA achieves 2.73-3.29× speedups without BV extensions. Hand-optimized vectorization yields gains of up to 3.33-4.59×, significantly outperforming automatic vectorization (0.98-1.08×). Manual bit manipulation optimizations improve performance, achieving 2.91-3.81× speedups, outperforming compiler-inserted instructions (1.28-1.68×). In addition, optimizing Keccak with compiler-inserted bit manipulation proves to be more effective than auto-vectorized NTT. While register renaming slightly improves auto-vectorization ($\sim 0.02\times$), it remains far from the efficiency of manual optimizations.

In conclusion, the obtained results highlight two main facts. First, compiler automatic optimization provides significantly worse results than carefully hand-optimized code, which outperforms ML-KEM and ML-DSA performance even when BV extensions are activated. Paying the cost of integrating BV standard extensions in the RISC-V core provides extra speedup only if such extensions are exploited manually, ranging from 21.5% (Verify, Dil-5) to 84.3% (Decapsulation, k768) over-optimized scalar code.

References

[1] Michele Mosca. "Cybersecurity in an Era with Quantum Computers: Will We Be Ready?" In: *IEEE Security & Privacy* (2018). DOI: 10.1109/MSP.2018.3761723.

[2] Xavier Carril et al. "Hardware Acceleration for High-Volume Operations of CRYSTALS-Kyber and CRYSTALS-Dilithium". In: *ACM TRETS* (2024). DOI: 10.1145/3675172.

[3] Tim Fritzmann et al. "RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography". In: *IACR TCHES* (2020). DOI: 10.13154/tches.v2020.i4.239-280.

[4] Jipeng Zhang et al. "Optimized Software Implementation of Keccak, Kyber, and Dilithium on RV{32,64}IM{B}{V}". In: *IACR TCHES* (2024). DOI: 10.46586/tches.v2025.i1.632-655.

[5] Víctor Soria-Pardos et al. "Sargantana: A 1 GHz+ In-Order RISC-V Processor with SIMD Vector Extensions in 22nm FD-SOI". In: *2022 25th DSD*. 2022. DOI: 10.1109/DSD57027.2022.00042.