Deferred Reduction Optimizations for Post-Quantum Lattice Cryptography: ML-KEM and ML-DSA

Mamone Tarsha Kurdi
SECEQ Research
mamone@seceq.com

December 2025

Abstract

The standardization of ML-KEM (FIPS 203) and ML-DSA (FIPS 204) marks a critical milestone in post-quantum cryptography adoption. However, achieving competitive performance with classical algorithms remains challenging. We present a unified framework of *deferred modular reduction* optimizations that exploit coefficient bound analysis to minimize expensive arithmetic operations. For ML-KEM, we introduce an optimized polynomial vector multiplication achieving 3.5–4× speedup through operation fusion, common subexpression elimination, and lazy 32-bit accumulation. We further present a 3-layer lazy INTT optimization providing 2.25× speedup with formal safety bounds. For ML-DSA, we develop a radix-4 NTT based on DFT composition theory yielding 16% signing improvement, alongside lazy reduction chains providing 12% verification speedup. All optimizations preserve constant-time execution and are validated against official NIST Known Answer Test vectors. Our open-source implementation, HPCrypt, demonstrates that mathematically-grounded optimizations can significantly improve lattice cryptography performance while maintaining security guarantees.

Keywords: Post-quantum cryptography, ML-KEM, ML-DSA, NTT, lazy reduction, FIPS 203, FIPS 204

1 Introduction

1.1 Background

The National Institute of Standards and Technology (NIST) concluded its post-quantum cryptography standardization process in 2024, selecting ML-KEM (FIPS 203) and ML-DSA (FIPS 204) as the primary lattice-based standards for key-encapsulation and digital signatures [1, 2]. These schemes, derived from CRYSTALS-Kyber and CRYSTALS-Dilithium respectively, are expected to see widespread deployment as organizations transition away from RSA and elliptic curve cryptography.

Performance remains a critical factor for adoption. While lattice-based schemes offer strong security guarantees against quantum adversaries, their computational overhead compared to classical algorithms can be significant. The core operations—polynomial multiplication via the Number Theoretic Transform (NTT) and modular reduction—dominate execution time and present opportunities for optimization.

1.2 Contributions

This paper presents a unified framework of *deferred modular reduction* optimizations applicable to both ML-KEM and ML-DSA. Our key insight is that careful coefficient bound analysis enables

delaying expensive modular reduction operations, accumulating intermediate results in wider integer types before performing a single final reduction.

Our specific contributions are:

1. **ML-KEM Polynomial Vector Multiplication** (Section 3.1): We present an optimized polyvec_basemul_acc_cached operation combining:
   - Operation fusion via the distributive law (reducing 24 to 2 reductions per coefficient pair)
   - Common subexpression elimination through multiplication caching
   - Lazy 32-bit accumulation with formal overflow bounds

   Combined speedup: **3.5–4.0**× on the critical multiplication path.
2. **ML-KEM Lazy INTT** (Section 3.2): A 3-layer deferred reduction strategy for inverse NTT operations following polynomial multiplication, achieving **2.25**× INTT speedup and **15–20%** end-to-end improvement.
3. **ML-DSA Radix-4 NTT** (Section 4.1): A hybrid radix-2/radix-4 NTT based on DFT composition theory, providing **16%** signing and **15%** verification improvement.
4. **ML-DSA Lazy Reduction Chains** (Section 4.2): Deferred reduction in matrix-vector multiplication with formal bounds analysis, achieving **12%** verification speedup.

All optimizations are implemented in Rust with optional SIMD acceleration (AVX2, AVX-512, NEON), preserve constant-time execution, and pass 100% of NIST Known Answer Test vectors.

1.3 Paper Organization

Section 2 establishes notation and background. Sections 3 and 4 present our ML-KEM and ML-DSA optimizations respectively. Section 5 discusses implementation details. Section 6 provides experimental evaluation. Section 7 surveys related work, and Section 8 concludes.

2 Preliminaries

2.1 Notation

We work in polynomial rings of the form $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ where q is prime and n is a power of 2. For ML-KEM, $q = 3329$ and $n = 256$. For ML-DSA, $Q = 8{,}380{,}417$ and $N = 256$.

Polynomials are represented as coefficient vectors $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$. We denote polynomial vectors of length k as $\mathbf{a} = (\mathbf{a}^{(0)}, \ldots, \mathbf{a}^{(k-1)})$ where each $\mathbf{a}^{(i)} \in R_q$.

2.2 Number Theoretic Transform

The Number Theoretic Transform (NTT) enables efficient polynomial multiplication in R_q. For a polynomial $\mathbf{a} \in R_q$, its NTT representation $\hat{\mathbf{a}} = \mathsf{NTT}(\mathbf{a})$ allows pointwise multiplication:

$$\mathbf{a} \cdot \mathbf{b} = \mathsf{INTT}(\mathsf{NTT}(\mathbf{a}) \circ \mathsf{NTT}(\mathbf{b}))$$

where $\circ$ denotes coefficient-wise multiplication.

The standard Cooley-Tukey radix-2 decimation-in-frequency (DIF) NTT consists of $\log_2(n)$ layers. Each layer performs $n/2$ butterfly operations:

$$a'_j = a_j + \zeta \cdot a_{j+\ell} \mod q \tag{1}$$

$$a'_{j+\ell} = a_j - \zeta \cdot a_{j+\ell} \mod q \tag{2}$$

where ζ is an appropriate twiddle factor (power of a primitive root of unity).

2.3 Modular Reduction

Montgomery reduction computes $a \cdot R^{-1} \mod q$ without division, using precomputed constants. For 32-bit operations with $R = 2^{32}$:

$$\mathsf{montgomery\_reduce}(a) = \frac{a - q \cdot ((a \cdot q^{-1}) \mod R)}{R}$$

Barrett reduction computes $a \mod q$ using precomputed $\mu = \lfloor 2^k/q \rfloor$:

$$\mathsf{barrett\_reduce}(a) = a - q \cdot \lfloor a \cdot \mu / 2^k \rfloor$$

Both methods require multiple CPU cycles (typically 3–10 cycles each), making reduction a significant performance bottleneck.

2.4 ML-KEM Operations

ML-KEM's critical path involves polynomial vector dot products. For vectors $\mathbf{a}, \mathbf{b}$ of length K:

$$\mathbf{r} = \sum_{i=0}^{K-1} \mathbf{a}^{(i)} \cdot \mathbf{b}^{(i)}$$

In NTT domain, each coefficient-pair multiplication (basemul) computes in the quotient ring $\mathbb{Z}_q[X]/(X^2 - \zeta)$:

$$r_0 = a_0 \cdot b_0 + \zeta \cdot a_1 \cdot b_1 \mod q \tag{3}$$

$$r_1 = a_0 \cdot b_1 + a_1 \cdot b_0 \mod q \tag{4}$$

2.5 ML-DSA Operations

ML-DSA signing involves computing $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1$ where c is a challenge polynomial with exactly τ non-zero coefficients (each ± 1). Verification requires matrix-vector products $\mathbf{A} \cdot \mathbf{z}$ where $\mathbf{A}$ is a $K \times L$ matrix of polynomials.

3 ML-KEM Optimizations

3.1 Optimized Polynomial Vector Multiplication

The polynomial vector dot product is the dominant operation in ML-KEM encapsulation and decapsulation. We present three complementary optimizations that together achieve 3.5–4× speedup.

3.1.1 Operation Fusion via Distributive Law

Theorem 3.1 (Operation Fusion)**.** *For polynomial vectors $\mathbf{a}, \mathbf{b}$ of length K in NTT domain, the dot product can be computed with a single modular reduction per output coefficient pair instead of K reductions.*

Proof. The standard approach computes:

$$r = \sum_{i=0}^{K-1} \mathsf{basemul}(\mathbf{a}^{(i)}, \mathbf{b}^{(i)})$$

with each basemul performing reductions. By the distributive law:

$$r_0 = \sum_{i=0}^{K-1} (a_0^{(i)} \cdot b_0^{(i)} + \zeta \cdot a_1^{(i)} \cdot b_1^{(i)}) \tag{5}$$

$$= \sum_{i=0}^{K-1} a_0^{(i)} \cdot b_0^{(i)} + \zeta \cdot \sum_{i=0}^{K-1} a_1^{(i)} \cdot b_1^{(i)} \tag{6}$$

Accumulating products before reduction gives a single reduction instead of K reductions per coefficient. □

This transformation reduces the operation count from $K \times 6$ reductions to 2 reductions per coefficient pair—a 12× reduction for $K = 4$.

3.1.2 Common Subexpression Elimination (Mulcache)

Observation 3.2. *In the expression $r_0 = \sum_i (a_0^{(i)} \cdot b_0^{(i)} + \zeta \cdot a_1^{(i)} \cdot b_1^{(i)})$, the term $b_1^{(i)} \cdot \zeta$ depends only on* $\mathbf{b}$ *and the twiddle factor, not on* $\mathbf{a}$.

Definition 3.3 (Multiplication Cache)**.** *For polynomial* $\mathbf{b}$ *in NTT domain, define:*

$$\mathsf{cache}[j] = b_{2j+1} \cdot \zeta_j \mod q$$

for $j \in \{0, \ldots, n/2 - 1\}$. This requires $n/2 = 128$ cached values.

When the same polynomial $\mathbf{b}$ is multiplied against multiple polynomials $\mathbf{a}^{(i)}$ (as in matrix-vector multiplication), pre-computing the cache eliminates K multiplications per coefficient pair.

Storage: 128 values × 2 bytes = 256 bytes per cached polynomial.

Savings: 33% fewer multiplications when $\mathbf{b}$ is reused.

3.1.3 Lazy 32-bit Accumulation

The key enabler for deferred reduction is proving that intermediate accumulations do not overflow.

Theorem 3.4 (Accumulation Safety for ML-KEM)**.** *For ML-KEM coefficients bounded by $|c| \leq 1664$, accumulating $K \leq 4$ products in 32-bit signed integers is safe:*

$$\max |acc| \leq 2K \cdot 1664^2 = 8 \cdot 2{,}768{,}896 = 22{,}151{,}168 < 2^{31} - 1$$

Proof. Each coefficient in reduced form satisfies $|c| \leq (q-1)/2 = 1664$. A single product contributes at most $1664^2 = 2{,}768{,}896$. The fused computation accumulates at most $2K$ such products (for r_0 and r_1 combined), giving maximum magnitude $2 \cdot 4 \cdot 2{,}768{,}896 = 22{,}151{,}168$, well within 32-bit signed range. □

3.1.4 Combined Algorithm

Algorithm 1 presents the optimized implementation.

3.1.5 Performance Analysis

Table 1 summarizes the reduction in modular operations.

3.2 Lazy INTT Optimization

After polynomial multiplication in NTT domain, the inverse NTT transforms results back to coefficient representation. We observe that polynomials produced by basemul have tighter coefficient bounds than arbitrary polynomials, enabling deferred reduction in early INTT layers.

Algorithm 1 Optimized polyvec_basemul_acc_cached

Require: Polynomial vectors $\mathbf{a}$, $\mathbf{b}$ of length K; precomputed $\mathsf{cache}[\mathbf{b}]$
Ensure: Result polynomial $\mathbf{r} = \sum_{i=0}^{K-1} \mathbf{a}^{(i)} \cdot \mathbf{b}^{(i)}$

for $j = 0$ to $n/2 - 1$ **do**
  $\mathsf{acc}_0 \leftarrow 0$; $\mathsf{acc}_1 \leftarrow 0$ ▷ 32-bit accumulators
  for $i = 0$ to $K - 1$ **do**
    $\mathsf{acc}_0 \leftarrow \mathsf{acc}_0 + a_{2j}^{(i)} \cdot b_{2j}^{(i)} + a_{2j+1}^{(i)} \cdot \mathsf{cache}^{(i)}[j]$
    $\mathsf{acc}_1 \leftarrow \mathsf{acc}_1 + a_{2j}^{(i)} \cdot b_{2j+1}^{(i)} + a_{2j+1}^{(i)} \cdot b_{2j}^{(i)}$
  end for
  $r_{2j} \leftarrow \mathsf{montgomery\_reduce}(\mathsf{acc}_0)$ ▷ Single reduction
  $r_{2j+1} \leftarrow \mathsf{montgomery\_reduce}(\mathsf{acc}_1)$
end for
return r

Table 1: Operation count reduction for $K = 4$

Technique	Reductions	Multiplications	Speedup
Baseline	24 per pair	12 per pair	1.0×
+ Fusion	8 per pair	12 per pair	1.75×
+ Mulcache	8 per pair	8 per pair	1.50×
+ Lazy Reduction	2 per pair	8 per pair	2.00×
Combined	**2 per pair**	**8 per pair**	**3.5–4.0×**

3.2.1 Coefficient Bound Analysis

Theorem 3.5 (INTT Layer Safety Bounds)**.** *For polynomials produced by basemul operations with coefficients bounded by $|c| \leq 0.57q \approx 1{,}908$, skipping Barrett reduction in the first 3 INTT layers is safe:*

$$\textit{After layer 1:} \quad |c| \leq 1.15q \approx 3{,}421 \tag{7}$$

$$\textit{After layer 2:} \quad |c| \leq 2.30q \approx 5{,}546 \tag{8}$$

$$\textit{After layer 3:} \quad |c| \leq 4.60q \approx 7{,}644 < 32{,}767 \tag{9}$$

All values remain within 16-bit signed integer range with a 4.3× safety margin.

Proof. The INTT butterfly computes $a' = a + b$ and $b' = \zeta(a - b)$. Starting with $|c| \leq B$, after one layer we have $|c'| \leq 2B$ (worst case when $|\zeta| \approx 1$). After 3 layers: $|c| \leq 8 \cdot 0.57q = 4.56q \approx 7{,}600 < 2^{15} - 1$. □

3.2.2 Algorithm

3.2.3 Performance Results

- **Micro-benchmark**: INTT time reduced from 995 ns to 441 ns (**2.25× speedup**)
- **End-to-end**: ML-KEM decapsulation improved by **15–20%**

4 ML-DSA Optimizations

4.1 Radix-4 NTT

The standard radix-2 NTT for $N = 256$ requires 8 layers of butterfly operations. We investigate merging consecutive layers into radix-4 butterflies.

Algorithm 2 Lazy INTT after basemul

Require: Polynomial $\mathbf{p}$ from basemul (coefficients $\leq 0.57q$)
Ensure: $\mathsf{INTT}(\mathbf{p})$ in coefficient form
1: **for** layer $\in \{1, 2, 3\}$ **do** ▷ Lazy layers
2: Perform butterfly operations **without** Barrett reduction
3: **end for**
4: **for** layer $\in \{4, 5, 6, 7\}$ **do** ▷ Full reduction layers
5: Perform butterfly operations **with** Barrett reduction
6: **end for**
7: **return p**

Table 2: NTT structure comparison for $N = 256$

Approach	Stages/Layers	Butterflies	Muls	Adds
Radix-2	8 stages	1,024	1,024	2,048
Radix-4	4 layers	256	1,024	2,048

4.1.1 DFT Composition Theory

Theorem 4.1 (Radix-4 Composition)**.** *Two consecutive radix-2 DIF stages can be composed into a single radix-4 stage processing 4 elements simultaneously.*

Proof. Consider radix-2 stages with lengths ℓ and $\ell/2$. The first stage computes:

$$b_0 = a_0 + \zeta_1 \cdot a_2, \quad b_2 = a_0 - \zeta_1 \cdot a_2 \tag{10}$$
$$b_1 = a_1 + \zeta_1 \cdot a_3, \quad b_3 = a_1 - \zeta_1 \cdot a_3 \tag{11}$$

The second stage then computes:

$$c_0 = b_0 + \zeta_2 \cdot b_1, \quad c_1 = b_0 - \zeta_2 \cdot b_1 \tag{12}$$
$$c_2 = b_2 + \zeta_3 \cdot b_3, \quad c_3 = b_2 - \zeta_3 \cdot b_3 \tag{13}$$

These 8 equations define a radix-4 butterfly with twiddle factors $(\zeta_1, \zeta_2, \zeta_3)$. □

4.1.2 Operation Count Analysis

Table 2 shows that radix-4 has identical operation counts to radix-2. However, practical benefits arise from:

- Fewer loop iterations (4 vs 8 layers)
- Better instruction-level parallelism (4 independent paths)
- Reduced loop overhead and branch mispredictions

4.1.3 Twiddle Factor Indexing

For radix-4 layer j processing block b:

$$\zeta_1 = \mathsf{ZETAS}[\mathsf{base}_1 + b] \tag{14}$$
$$\zeta_2 = \mathsf{ZETAS}[\mathsf{base}_2 + 2b] \tag{15}$$
$$\zeta_3 = \mathsf{ZETAS}[\mathsf{base}_2 + 2b + 1] \tag{16}$$

where base indices depend on the layer number.

4.1.4 Hybrid Implementation

Empirical evaluation revealed that radix-4 benefits the forward NTT but causes regression in inverse NTT due to register pressure. Our implementation uses:

- **Forward NTT**: Radix-4 (16% speedup)
- **Inverse NTT**: Radix-2 (no regression)

4.1.5 Performance Results

- **ML-DSA-65 Signing**: 16% improvement
- **ML-DSA-65 Verification**: 15% improvement

4.2 Lazy Reduction in Arithmetic Chains

ML-DSA verification involves computing $\mathbf{w}' = \mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t}_1 \cdot 2^d$, requiring matrix-vector products with L polynomial multiplications accumulated per output element.

4.2.1 Coefficient Bound Analysis

Theorem 4.2 (Chain Accumulation Safety)**.** *For ML-DSA polynomials with coefficients in $[-Q, Q]$, accumulating k polynomial additions without intermediate reduction is safe for $k \leq 256$:*

$$\max |\mathsf{acc}| \leq k \cdot Q = 256 \cdot 8{,}380{,}417 = 2{,}145{,}386{,}752 < 2^{31} - 1$$

For ML-DSA parameter sets with $L \in \{4, 5, 7\}$, this bound is satisfied with large safety margins.

4.2.2 Standard vs. Lazy Approach

Standard approach (reduction after each operation):

1: **for** $j = 0$ to $L - 1$ **do**
2: prod ← poly_multiply($\mathbf{A}[i][j], \mathbf{z}[j]$)
3: prod.reduce() ▷ Reduction here
4: acc ← acc.add(prod)
5: **end for**

Lazy approach (single reduction at end):

1: **for** $j = 0$ to $L - 1$ **do**
2: prod ← poly_multiply($\mathbf{A}[i][j], \mathbf{z}[j]$)
3: acc ← acc.add_lazy(prod) ▷ No reduction
4: **end for**
5: acc.reduce() ▷ Single reduction

4.2.3 Performance Results

Table 3 shows speedup scaling with chain length. For ML-DSA-65 verification with $L = 5$, we achieve **12% end-to-end improvement**.

Table 3: Lazy reduction chain speedup

Chain Length	Speedup	Reductions Saved
2 operations	1.11×	50%
4 operations	2.51×	75%
8 operations	4.21×	87.5%
$L = 5$ (ML-DSA-65)	1.12×	80%

5 Implementation

5.1 Architecture

Our implementation, HPCrypt, is written in Rust, targeting both portability and performance:

- **Portable baseline**: Pure Rust with no platform-specific code
- **SIMD acceleration**: Optional AVX2, AVX-512 (x86-64), and NEON (ARM) backends
- **Compile-time specialization**: Const generics enable parameter-specific optimization
- **Runtime detection**: CPU feature detection with automatic fallback

5.2 SIMD Considerations

The lazy INTT optimization requires careful SIMD integration:

- **Layers 1–3 (lazy)**: Use scalar operations to handle potential overflow gracefully
- **Layers 4–7 (full reduction)**: Use SIMD vectorization for parallel butterfly operations

This hybrid approach achieves both the benefits of lazy reduction and SIMD parallelism.

5.3 Constant-Time Guarantees

All optimizations preserve constant-time execution:

- No secret-dependent branches
- No secret-dependent memory access patterns
- Branchless conditional operations using arithmetic masking

6 Evaluation

6.1 Experimental Setup

- **Hardware**: Intel Core i7-12700K (Alder Lake), AMD Ryzen 9 5900X
- **Software**: Rust 1.75, compiled with `-C target-cpu=native`
- **Benchmarking**: Criterion.rs with 100 iterations and statistical analysis
- **Validation**: 100% pass rate on NIST KAT vectors

Table 4: ML-KEM performance improvement

Operation	Baseline	Optimized	Improvement
ML-KEM-512 Decaps	26.2 μs	23.2 μs	11.5%
ML-KEM-768 Decaps	44.5 μs	35.9 μs	19.3%
ML-KEM-1024 Decaps	91.4 μs	85.0 μs	7.0%
Basemul (micro)	1,200 ns	320 ns	3.75×
INTT (micro)	995 ns	441 ns	2.25×

Table 5: ML-DSA performance improvement

Operation	Baseline	Optimized	Improvement
ML-DSA-44 Sign	380 μs	328 μs	14%
ML-DSA-65 Sign	450 μs	388 μs	16%
ML-DSA-87 Sign	620 μs	540 μs	13%
ML-DSA-44 Verify	120 μs	106 μs	12%
ML-DSA-65 Verify	150 μs	132 μs	12%
ML-DSA-87 Verify	210 μs	185 μs	12%

6.2 ML-KEM Results

6.3 ML-DSA Results

7 Related Work

NTT Optimizations. Seiler [3] presented AVX2-optimized NTT implementations for lattice cryptography. Becker et al. [4] developed NEON implementations achieving significant speedups on ARM platforms.

Lazy Reduction. The concept of deferred modular reduction appears in various contexts. Our contribution is the systematic application to ML-KEM/ML-DSA with formal bounds analysis.

Radix-4 NTT. Higher-radix NTT has been explored for hardware implementations [5]. We provide a practical software implementation with hybrid radix-2/radix-4 strategy.

8 Conclusion

We presented a unified framework of deferred modular reduction optimizations for ML-KEM and ML-DSA. Our key contributions include:

1. **Formal bounds analysis** proving safety of lazy accumulation strategies
2. **3.5–4× speedup** on ML-KEM polynomial vector multiplication
3. **2.25× speedup** on ML-KEM INTT operations
4. **16% signing improvement** for ML-DSA via radix-4 NTT
5. **12% verification improvement** for ML-DSA via lazy reduction chains

All optimizations preserve constant-time execution and are validated against NIST test vectors. Our open-source implementation, HPCrypt,[1] demonstrates that mathematically-grounded

[1] `https://github.com/seceq/hpcrypt`

optimizations can significantly improve post-quantum cryptography performance while maintaining security guarantees.

Future Work. Promising directions include formal verification of our bounds proofs, extension to other lattice schemes (NTRU, FrodoKEM), and investigation of hardware-specific optimizations (AVX-512, Apple Silicon).

References

[1] National Institute of Standards and Technology. FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard, 2024.

[2] National Institute of Standards and Technology. FIPS 204: Module-Lattice-Based Digital Signature Standard, 2024.

[3] Gregor Seiler. Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography. Cryptology ePrint Archive, Report 2018/039, 2018.

[4] Hanno Becker, Vincent Hwang, Matthias J. Kannwischer, Bo-Yin Yang, and Shang-Yi Yang. Neon NTT: Faster Dilithium, Kyber, and Saber on Cortex-A72 and Apple M1. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2022.

[5] Ahmet Can Mert, Erdinc Ozturk, and Erkay Savas. Design and implementation of a fast and scalable NTT-based polynomial multiplier architecture. In Euromicro Conference on Digital System Design, 2020.

[6] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber: Algorithm Specifications and Supporting Documentation. NIST PQC Round 3 Submission, 2020.

[7] Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: Algorithm Specifications and Supporting Documentation. NIST PQC Round 3 Submission, 2020.