GPU Acceleration of VOLEitH-Based Post-Quantum Signatures: A Reusable Fragment-Graph Framework

Victor Shyba[1], **Ricardo Felipe Custódio**[1]

[1]Laboratório de Segurança em Computação (LabSEC)
Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brasil

victor.shyba@gmail.com, ricardo.custodio@ufsc.br

Abstract. *Post-quantum digital signature schemes based on the Vector Oblivious Linear Evaluation in the Head (VOLEitH) paradigm represent a promising direction in the NIST Post-Quantum Cryptography standardization process for additional digital signatures. FAEST, an AES-based scheme, was selected for the third round in May 2026 [Alagic et al. 2026]; PERK, based on the Permuted Kernel Problem, was not selected but adopted the VOLEitH framework and serves here as an architectural case study. To the best of our knowledge, this is the first reported GPU implementation of a VOLEitH-based signature pipeline. This paper presents the design, implementation, and evaluation of a reusable GPU framework for VOLEitH-based signatures, implemented in CUDA and applied to FAEST and PERK. The architecture adopts a fragment-graph model over caller-owned workspaces, covering seed expansion via AES-CTR, GGM tree construction, BAVC commitments, VOLE operations, and scheme-specific stages. Experiments on an NVIDIA H100 GPU report throughput of up to 34,026 sig/s for FAEST-128f and speedups of 1.23×–2.77× for signing and 1.99×–7.59× for verification over a normalized 16-worker AVX CPU baseline across all six FAEST v2 variants. The PERK GPU implementation achieves a speedup of 5.65×–9.64× in signing speed under the same methodology, providing empirical evidence of the developed infrastructure.*

1. Introduction

The advancement of quantum computing represents a concrete threat to the long-term security of cryptographic systems based on the integer factorization and discrete logarithm problems [Shor 1994, Rivest et al. 1978, Diffie and Hellman 1976]. Although quantum computers capable of attacking these algorithms at a practical scale do not yet exist, the transition is already urgent because of Store-Now-Decrypt-Later (SNDL) [Joseph et al. 2022] attacks. In this scenario, adversaries can intercept and store encrypted communications to decrypt them later, once they have access to sufficient quantum hardware. Data requiring long-term confidentiality, such as medical records, industrial secrets, and government communications, would be particularly exposed.

In response to this scenario, NIST has been conducting, since 2016, a standardization process for post-quantum cryptographic algorithms [Alagic et al. 2022], which resulted in the selection of CRYSTALS-Dilithium, Falcon, and SPHINCS+ for digital signatures in 2022 [Ducas et al. 2018, Bernstein et al. 2020b]. Recognizing the need for cryptographic diversity, NIST launched an additional call for signature schemes based on different

mathematical foundations in 2022 [National Institute of Standards and Technology 2022], selecting nine algorithms for the third round in May 2026, among them FAEST [Alagic et al. 2026]. This process highlights the importance of the MPCitH paradigm, which constructs signatures from zero-knowledge proofs, with FAEST standing out among the VOLEitH-based candidates for its conservative security assumptions and competitive performance.

Vector Oblivious Linear Evaluation in the Head (VOLEitH), proposed by Baum et al. [Baum et al. 2023], improves MPCitH by replacing generic multiparty computation protocols with VOLE correlations over finite fields. This choice concentrates the proof cost on non-linear operations, producing measurably more compact signatures and faster operations than MPCitH approaches [Baum et al. 2023]. FAEST [Baum et al. 2025] is an AES-based scheme selected by NIST for the third round in May 2026 [Alagic et al. 2026], and is the primary target of the GPU optimization efforts in this work. PERK [PERK Team 2024], based on the Permuted Kernel Problem, adopted the VOLEitH framework in its second-round version but was not selected for the third round; NIST explained that PERK's smaller signatures did not compensate for its lower computational efficiency on CPU platforms, noting that "FAEST offers superior speed and relies solely on AES" [Alagic et al. 2026]. This CPU-centric comparison motivates a complementary question: *does the relative performance profile change when GPU acceleration is available?* The PERK GPU implementation presented here serves as an architectural case study, demonstrating the cross-scheme reusability of the proposed infrastructure and providing empirical data on PERK's throughput in high-parallelism scenarios.

Despite the high degree of parallelism inherent in VOLEitH, evidenced by the τ independent VOLE instances, nodes at the same level in GGM trees, and vector operations over finite fields, no GPU implementation had been reported in the literature. Reference implementations of FAEST and PERK were optimized for CPUs with AVX2 instructions [Baum et al. 2025, PERK Team 2024]. In parallel, the community has demonstrated that GPUs can provide substantial acceleration for other post-quantum algorithms, including reported factors of 18.5× for SPHINCS+ [Wang et al. 2024] and 166×–181× signing speedups for Dilithium on A100 [Shen et al. 2024], depending on the structure of the algorithm and the adopted baseline.

Contributions. To the best of the authors' knowledge, this paper presents the first GPU infrastructure reported in the literature for VOLEitH-based signature schemes. The main contributions are: (i) a *framework* based on a *fragment graph* over caller-owned workspaces for the VOLEitH primitive, including seed expansion via AES-CTR, GGM trees, BAVC commitments, and the VOLECommit, VOLEHash, and VOLEReconstruct operations; (ii) a systematic experimental evaluation on an NVIDIA H100 GPU, showing signing speedups between 1.23× and 2.77× and verification speedups between 1.99× and 7.59× relative to a normalized 16-worker AVX baseline across all six FAEST v2 parameter sets; and (iii) a PERK case study that reuses the same architectural discipline, batching model, and workspace strategy, while preserving scheme ownership of protocol-specific stages, achieving signing speedups between 5.65× and 9.64×.

Organization. Section 2 presents the theoretical background. Section 3 describes the proposed implementation. Section 4 presents the results, compares them with the literature, and discusses limitations. Section 5 concludes the work.

2. Theoretical Background

2.1. Post-Quantum Cryptography

Post-quantum cryptography comprises algorithms designed to resist quantum adversaries, organized into families such as lattices (CRYSTALS-Kyber, CRYSTALS-Dilithium [Bos et al. 2018, Ducas et al. 2018]), error-correcting codes (Classic McEliece, BIKE [Bernstein et al. 2020a, Aragon et al. 2020]), and zero-knowledge proofs over symmetric primitives (FAEST [Baum et al. 2025], PERK [PERK Team 2024]). The latter serves as an architectural case study in this work. Symmetric primitives remain comparatively robust: Grover's algorithm [Grover 1996] reduces exhaustive key search costs from $O(2^N)$ to $O(2^{n/2})$ in the quantum model, so AES-256 retains long-term post-quantum security.

The NIST standardization process for post-quantum algorithms, initiated in 2016, selected four algorithms in 2022 and subsequently issued an additional call for digital signature schemes [National Institute of Standards and Technology 2022]. Both FAEST and PERK participated as second-round candidates in this process. In its second-round status report [Alagic et al. 2026], NIST advanced FAEST to the third round but did not select PERK, observing that PERK's smaller signatures did not offset its lower computational efficiency on CPU platforms. The report notes that "FAEST offers superior speed and relies solely on AES" [Alagic et al. 2026]. Section 4 revisits this comparison from a GPU perspective.

2.2. Zero-Knowledge Proofs and the VOLEitH Paradigm

A zero-knowledge proof (ZKP) [Quisquater et al. 1990] is a protocol between a prover P and a verifier V that satisfies three properties: *completeness*, by which an honest prover always convinces an honest verifier; *soundness*, meaning that a dishonest prover convinces an honest verifier only with negligible probability; and *zero knowledge*, by which the interaction reveals no information about the witness. In a signature setting, the statement is tied to a public key, and the witness is the corresponding secret key. For public-coin interactive protocols, the Fiat–Shamir transformation [Fiat and Shamir 1987] makes them non-interactive by deriving the verifier's challenges from hashes of the transcript and message. Schemes such as FAEST and PERK use this transformation so that the resulting proof transcript is bound to the signed message and can be verified as a signature.

The MPCitH paradigm [Ishai et al. 2007] constructs ZKPs by mentally simulating a multiparty computation protocol: the prover divides the secret into shares, simulates the protocol, and reveals all states except one, determined by the challenge. Although flexible, this approach produces signatures that are proportionally larger with increasing numbers of simulated parties.

The VOLEitH, proposed by Baum et al. [Baum et al. 2023], improves MPCitH by exploiting correlations of the *Vector Oblivious Linear Evaluation* type. Given $\mathbf{u} \in \mathbb{F}_p^\ell$, $\mathbf{V} \in \mathbb{F}_{p^k}^\ell$ and $\Delta \in \mathbb{F}_{p^k}$, a VOLE correlation satisfies:

$$\mathbf{Q} = \mathbf{u} \cdot \Delta + \mathbf{V}. \tag{1}$$

The resulting linear homomorphic commitment enables verification of linear operations over the extension field at no additional proof cost, concentrating the cost exclusively on

non-linear operations (e.g., the AES S-box). This produces measurably more compact signatures and faster operations than MPCitH approaches [Baum et al. 2023]. Figure 1 illustrates the resulting VOLEitH transcript at the level shared by both schemes. The prover first commits to VOLE-derived hidden views; the transcript then proceeds through three Fiat–Shamir challenge barriers. The middle response proves the scheme-specific relation: FAEST uses QuickSilver for the AES-based one-way function statement, while PERK proves PKP consistency. The final opening allows the verifier to reconstruct the challenged views and check both the VOLE consistency and the relation proof.

The central components of this VOLEitH transcript relevant to GPU mapping are: (a) **GGM trees**: the AES-CTR-based PRG expands λ-bit seeds into binary trees; nodes at the same level are independent of one another, exposing intra-tree parallelism; (b) **BAVC** (*Batch All-but-One Vector Commitment*): this structures the leaves of the GGM trees into commitments that allow all but one to be revealed, indexed by the challenge; (c) **VOLECommit, VOLEHash, VOLEReconstruct**: the three fundamental protocol operations, which operate on vectors of length ℓ with arithmetic in $\mathbb{F}_{p^k}$.

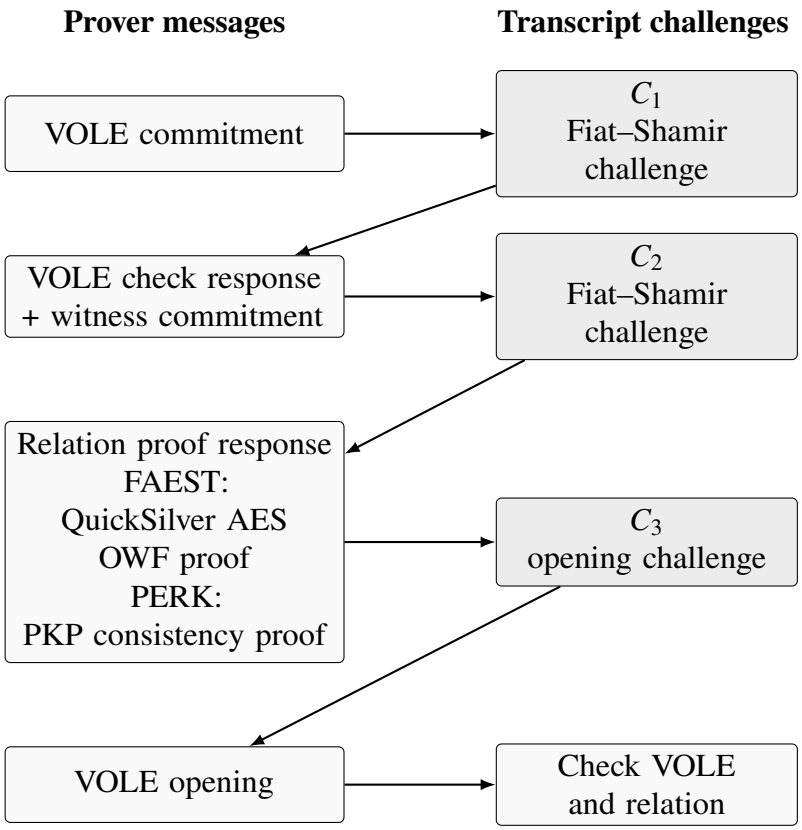

Figure 1. Interactive-transcript view of the VOLE-in-the-head structure, adapted from the FAEST specification Fig. 2.1. In the signature scheme, the challenges are computed non-interactively by the prover via Fiat–Shamir hashing of the transcript. The common framework is the sequence of VOLE commitment, challenge barriers, responses, and opening; the relation proved by the third prover message remains scheme-specific.

2.3. The FAEST and PERK Schemes

The FAEST (*Fast AES-based Signature from VOLEitH*) [Baum et al. 2025] uses AES as a one-way function: the private key is an AES key, and the public key is the pair (x, y) with $y = \mathrm{AES}_{sk}(x)$. The signature proves knowledge of sk via VOLEitH and QuickSilver [Baum et al. 2023], verifying the S-box constraints over $\mathbb{F}_{2^8}$ by degree-3 Galois norms. Version 2 replaced the SHAKE-based PRG with AES-CTR, making the scheme more efficient and increasing its potential for parallelization on GPU [Baum et al. 2025]. Table 1 presents the parameters of the six submitted variants.

Table 1. Parameters of the FAEST v2 [Baum et al. 2025]. λ: security level (bits); τ: VOLE instances; $|\sigma|$: signature size (bytes); times on AVX2 (ms).

Variant	λ	τ	$\lvert\sigma\rvert$	Sign (ms)	Verify (ms)
FAEST-128s	128	11	4,506	3.76	2.88
FAEST-128f	128	16	5,924	0.51	0.42
FAEST-192s	192	16	11,260	16.08	12.44
FAEST-192f	192	24	14,948	2.07	1.79
FAEST-256s	256	22	20,696	22.45	21.93
FAEST-256f	256	32	26,548	3.26	3.01

The PERK (*Post-quantum Efficient aRgument of Knowledge*) [PERK Team 2024] is based on the hardness of the Permuted Kernel Problem (PKP). Starting with version 2.1.0, PERK adopted a VOLEitH-style structure, which makes its GPU pipeline comparable to FAEST in orchestration, batching, workspace management, and symmetric-primitive support. In the implementation studied here, the shared code is concentrated in the build/runtime infrastructure and in selected CUDA support and primitive helpers; protocol stages such as GGM expansion, VOLE conversion/commitment, transcript handling, and the final PKP proof are implemented through PERK-owned modules. In Category I, the *perk-1-fast* variant produces signatures of approximately 4.32 kB, smaller than those of FAEST-128f, with a public key of 104 bytes.

2.4. Implementations of Post-Quantum Algorithms on GPU

The viability of GPUs for post-quantum cryptography has been demonstrated in different algorithmic families. Shen et al. [Shen et al. 2024] reported A100 speedups of 166×–181× for Dilithium signing and 88×–109× for verification over their CPU reference baseline, employing memory coalescing and shared-memory bank-conflict elimination optimizations. Wan et al. [Wan et al. 2022] achieved 26×, 36×, and 35× over an AVX2 CPU implementation for Kyber-1024 key generation, encapsulation, and decapsulation, respectively, using Tensor Cores on an RTX 3080. Wang et al. [Wang et al. 2024] and Wu et al. [Wu et al. 2025] presented GPU implementations of SPHINCS+ with speedups of up to 18.5× for signing and 11.3× for verification under their reported baselines, which are relevant to VOLEitH because they share the hash tree primitive. Lee et al. [Lee et al. 2022] developed AES-CTR on GPU with 1,598 Gbps on V100, establishing T-box placement and coalesced-write techniques that are relevant for optimizing the FAEST PRG.

To the best of the authors' knowledge, no GPU implementation of the VOLEitH primitive, nor its central components (VOLECommit, VOLEHash, VOLEReconstruct, BAVC), has been reported in the literature in the context of post-quantum signature schemes.

3. Implementation of VOLEitH on GPU

The design pursues four explicit goals: *throughput-first* execution that saturates a GPU through batching; *byte-level compatibility* with the reference implementations, so that signatures interoperate in both directions; *caller-owned workspace discipline* that removes allocation from the hot path; and *partial portability* where possible across the VOLEitH family, demonstrated here by reusing the architecture for PERK. The work is deliberately scoped to a single GPU and does not claim a universal VOLEitH framework, constant-time guarantees, or a fully shared protocol-stage implementation across schemes; these boundaries are revisited in Section 4.4.

Artifacts. The implementation and reproduction package accompanying this submission includes the CUDA source code, build and validation scripts, reference KATs, and the commands used to reproduce Tables 2 and 3. It is publicly available at `https://github.com/shyba/gpu-vole`.

3.1. Parallelism Analysis

Efficiently mapping VOLEitH to GPU requires the explicit identification of the available dimensions of parallelism. Four independent dimensions were identified:

1. **Parallelism across VOLE instances**: the protocol executes τ independent instances of the VOLE commitment (from 11 to 32 in FAEST), which can be assigned to distinct CUDA blocks;
2. **Intra-tree GGM parallelism**: at the level i of each of the τ trees of depth d, there are 2^i nodes expandable in parallel, exposing $\tau \cdot 2^i$ units per level;
3. **Vector parallelism in VOLE operations**: the multiplications $u_j \cdot \Delta$ for $j \in [1, \ell]$ are independent, as are the sums in VOLECommit and the scalar products in VOLEHash;
4. **Batch parallelism**: B independent signatures or verifications processed simultaneously. For variants with lower per-instance costs, this is the dimension required to saturate datacenter GPUs.

The composition $B \cdot \tau \cdot 2^d$ exceeds 10^6 independent work units in typical configurations ($B = 16{,}384$, $\tau = 16$, $d = 8$), supporting the GPU-based approach.

3.2. Fragment-Graph Architecture

The implementation adopts a *fragment graph* architecture over caller-owned *workspaces*, illustrated in Figure 2. Each protocol stage is encapsulated in a *fragment*: a CUDA translation unit with a standardized signature that launches *kernels* over predefined regions of device memory. The host orchestrator invokes the *fragments* in protocol order, while common infrastructure handles build integration, dispatch, scratch sizing, streams, errors, and selected primitive support. Protocol-sensitive stages remain owned by each scheme and carry separate equivalence tests.

The implementation is organized around a *fragment graph* discipline over caller-owned *workspaces*. The graph is best understood as an architectural pattern: the host orchestrator invokes CUDA stages in protocol order over predefined device-memory regions, while common infrastructure handles build integration, dispatch, scratch sizing, streams, errors, and selected primitive support. The current codebase deliberately keeps protocol-sensitive stages owned by each scheme. In particular, transcript ordering, parameter

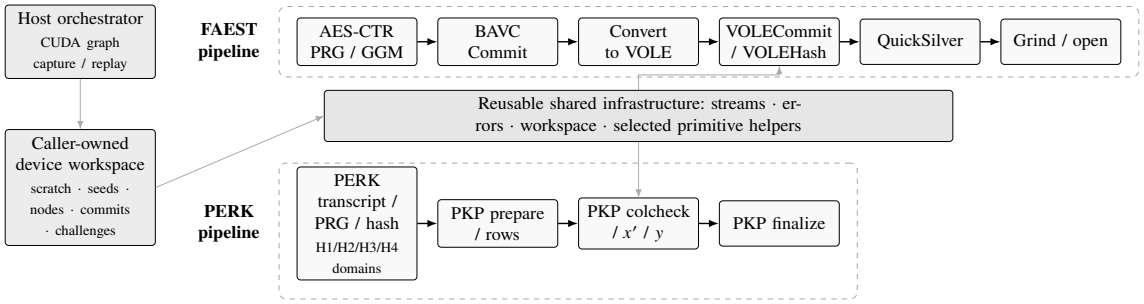

Figure 2. Fragment-graph architecture. A common layer provides host orchestration, caller-owned *workspaces*, CUDA graph capture, and reusable CUDA infrastructure. The dashed protocol pipelines remain scheme-owned.

interpretation, VOLE layout, commitment logic, and the final QS/PKP-specific work are different, with separate equivalence tests.

The three pillars of the architecture are:

a) Decomposition into *fragments*. A monolithic *kernel* is infeasible because inter-stage synchronizations in VOLEitH cross block boundaries. Generic low-level functions would lose the local decisions regarding geometry, occupancy, and layout that make each stage efficient. Decomposition into *fragments* preserves replaceability: changing the implementation of a stage only requires redirecting a call from the orchestrator without affecting the others;

b) Caller-owned *workspace*. All device memory is allocated once when the context is created. The *fragments* receive pointers to precomputed regions and never call `cudaMalloc` in the hot path, eliminating: (i) dynamic allocation overhead with implicit synchronization; (ii) memory fragmentation; and (iii) pointer nondeterminism that prevents capture by CUDA graphs;

c) CUDA graph capture. The deterministic sequence of *kernels* is captured once in a `cudaGraph_t` and re-executed in the hot path without *per-kernel* launch overhead. This mechanism is particularly relevant in the initial levels of the GGM trees, where each level launches a *kernel* with only a few dozen nodes. In this situation, the launch cost begins to dominate the useful work performed.

3.3. Implementation of the Components

Seed expansion (AES-CTR as PRG). One *warp* per instance, one AES block per *thread*, with multiple counters per call to increase the useful work per *thread*. In the measured FAEST configuration, the AES round tables reside in constant memory, with lane-strided and interleaved layouts available per launch profile; in PERK, T-box tables reside in shared memory with padded rows and a configurable per-lane replication factor, following Lee et al. [Lee et al. 2022].

Note on T-box lookups. Table-based AES S-box evaluation is known to be vulnerable to cache-timing attacks on CPUs, where table indices depend on secret key material. On GPU, constant-memory lookups serialize on divergent addresses within a warp, and shared-memory accesses can incur bank conflicts that depend on secret-indexed lookups; neither placement removes data-dependent timing by construction. We did not perform a

dedicated side-channel evaluation of this pattern; it is part of the open items discussed in Section 4.4 and should be treated as unverified rather than assumed safe.

GGM trees. One *kernel* per level, with grid $B \cdot \tau \cdot 2^i$ at level i. Capture by CUDA graphs amortizes the *overhead* of the initial levels over repeated executions.

BAVC. Three *kernel* variants were evaluated for computing the leaf *hashes* h_i:*tiles*, *warp-centric*, and *split-stage*. The *split-stage* variant with unrolled scalar expansion produced a 4.2× speedup in the isolated stage, although the end-to-end gain was ≈ 14% because other stages began to dominate the total time. The aggregate *hash* uses a tree reduction with *warps* and blocks cooperating via global memory.

VOLE operations. The *structure-of-arrays* (SoA) layout keeps homologous components of distinct instances contiguous, enabling coalesced accesses per *warp*. The VOLE pipeline operates in *streaming* mode: values are consumed as they are produced, eliminating materialization in global memory. The measured FAEST implementation represents elements of $\mathbb{F}_{2^{128}}$, $\mathbb{F}_{2^{192}}$, and $\mathbb{F}_{2^{256}}$ as two, three, and four little-endian 64-bit *limbs*, respectively. Multiplication uses fixed-count, nibble-at-a-time shift-and-add routines. Reduction uses $x^{128} + x^7 + x^2 + x + 1$ and $x^{192} + x^7 + x^2 + x + 1$ (constant `0x87`) for the 128- and 192-bit fields, and $x^{256} + x^{10} + x^5 + x^2 + 1$ (constant `0x425`) for the 256-bit field. Loop bounds depend only on the field size, masked partial-product selection is branchless, and the remaining branches select limbs by public loop indices or aligned load/store paths. The shift-by-four reduction step indexes a lookup table with a data-derived high nibble, so this field path does not provide data-independent memory addresses. The three-limb representation may entail different resource use at 192 bits, but no retained H100 profile establishes register pressure, occupancy loss, or spills as the cause of the observed throughput.

Scheme-specific stages. The QuickSilver stage of FAEST verifies the AES S-box constraints using degree-3 Galois norms over $\mathbb{F}_{2^8}$ [Baum et al. 2025]. The PKP stage of PERK evaluates the permutation proof over $\mathbb{F}_{2048}$.

3.4. Implemented Optimizations

The optimizations fall into three groups, each addressing a distinct bottleneck.

Launch-overhead reduction. CUDA graph capture re-executes the deterministic kernel sequences without *per-kernel* launch cost. Caller-owned *workspaces* keep device memory preallocated and graph-capture friendly, while independent contexts or streams can use separate workspace instances. Adaptive launch profiles select geometry by GPU architecture.

Memory layout and data movement. A *structure-of-arrays* layout enables coalesced *per-warp* accesses. The VOLE pipeline runs in *streaming* mode with fusion, where stages can be safely combined, and a direct-write output path for VOLEReconstruct eliminates intermediate buffers.

Stage-specific kernel specialization. T-box placement in AES-CTR (shared, constant, or global memory) is selected according to the target architecture; BAVC offers *tiled*, *warp-centric*, and *split-stage* variants with optional fusion between seed expansion and leaf hashing. Thread geometry is also tuned separately for the verification stages.

The final implementations preserve protocol-level compatibility with the reference implementations, as validated by the *Known Answer Tests* (KATs) from each NIST submission package. Tests were also expanded for smaller components by capturing intermediate states from the reference code. Finally, integration tests are in place to verify that the reference code can verify signatures generated by the CUDA implementation, and vice versa.

4. Results, Comparison, and Discussion

4.1. Experimental Methodology

Platforms. The GPU implementation was evaluated on an NVIDIA H100 80 GB HBM3 (*compute capability* 9.0, Hopper architecture) accessed via the Modal platform with precompiled binaries (CUDA 12, driver 535). The H100 is a datacenter GPU with a nominal 700 W thermal design power (TDP), whereas the Intel Core i7-11800H baseline is an 8-core/16-thread mobile processor with a nominal 45 W TDP and no AVX-512 support, running 16 AVX2 reference workers. This comparison reports throughput for these measured configurations only. It does not support conclusions about equivalent hardware classes, cost, energy consumption, or performance per watt.

Evaluation protocol. Batch sizes were selected by an auto-tuning sweep over powers of two, using the same candidate set for all variants and recording signing and verification throughput separately. The search phase used short warm-up and measurement runs to identify stable high-throughput batch sizes; the final runs used additional warm-up passes and longer measurements at the selected sizes. Timing was measured with CUDA events around the graph replay or individual CUDA operations. One-time context creation, graph capture, and workspace allocation were performed before the timed region and are therefore excluded. Input and output buffers were prepared outside the timed interval, so the reported figures reflect batched GPU operations without end-to-end application I/O. They are therefore device-resident throughput measurements rather than application-level end-to-end timings. Measurements used CUDA graphs (`graphs=1 parameter`) and the configured workspace-pool setting for that run.

Data transfer cost. Host–device transfer of input/output buffers is excluded from the timed region, reflecting a deployment model where batches are assembled and consumed on-device across a pipeline of calls (e.g., an issuance service maintaining device-resident state) rather than a single-shot call-and-transfer pattern.

4.2. Results

Tables 2 and 3 present the throughput results for FAEST and PERK, respectively.

The highest signing throughput observed is 34,026 sig/s for FAEST-128f with a batch of 16,384 signatures. The highest verification throughput reaches 122,121 ver/s for the same variant, using a batch size of 4096. All speedups reported for our implementation are measured against the normalized 16-worker AVX baseline used in this study.[1]

[1]For reference, against the portable *single-thread* implementation, the H100 accelerates FAEST-128f signing by ≈ 58.8× and FAEST-256s by ≈ 51.4×; this baseline is not optimized and is reported only for context.

Table 2. FAEST throughput comparison: normalized AVX CPU (16 processes, batch 4) and H100 GPU (auto-selected batch). GPU/CPU factor indicated.

Variant	Signing (sig/s)			Verification (ver/s)		
	CPU	GPU	×	CPU	GPU	×
FAEST-128f	25,294	34,026	1.35	27,192	122,121	4.49
FAEST-128s	2,739	5,784	2.11	3,145	23,864	7.59
FAEST-192f	5,563	8,681	1.56	6,207	12,368	1.99
FAEST-192s	707	1,959	2.77	868	2,308	2.66
FAEST-256f	3,507	4,316	1.23	3,747	15,611	4.17
FAEST-256s	453	993	2.19	452	1,052	2.33

Table 3. PERK signing throughput (`aes_aes` family, *fast* variants): H100 GPU vs. i7-11800H CPU (16 processes).

Variant	CPU (sig/s)	GPU (sig/s)	GPU/CPU
perk-1-fast	7,506	42,446	5.65
perk-3-fast	1,359	9,655	7.10
perk-5-fast	671	6,468	9.64

4.3. Comparison with Related Work

Table 4 compares the speedup factors reported in this work with representative results from the literature for other post-quantum algorithms on GPU.

The speedups reported here are lower than the largest reported factors for Dilithium and Kyber. Because the algorithms, optimizations, baselines (single vs multiple threads and different CPU models), and platforms differ, the comparison serves to explain what drives the magnitude of each factor rather than to rank the works. The largest gap is structural: Dilithium and Kyber are dominated by polynomial arithmetic and NTT-heavy kernels, which expose highly regular massive parallelism [Shen et al. 2024, Wan et al. 2022].

Compared with SPHINCS+ [Wang et al. 2024, Wu et al. 2025], the GGM tree structure of VOLEitH is conceptually similar to Merkle trees but uses AES-CTR as a PRG rather than hash functions. This characteristic enables exploitation of native AES instructions in modern CPUs, reducing the gap. On the other hand, the speedup factors obtained for PERK, between 5.65× and 9.64×, are higher because PKP is well-suited to GPU architecture, as matrix permutations are a natural fit.

Platform context and the NIST elimination of PERK. The NIST second-round report eliminated PERK on the grounds that its computational efficiency on CPU platforms did not justify its smaller signatures relative to FAEST [Alagic et al. 2026]. The GPU results presented here provide a complementary perspective. On an NVIDIA H100, PERK achieves signing speedups of 5.65× to 9.64× over the normalized CPU baseline, whereas FAEST achieves speedups between 1.23× and 2.77× under the same evaluation methodology. This difference suggests that the relative performance profiles of the two schemes depend on the target computing platform.

We do not suggest that these results would have altered the NIST decision, since

Table 4. Speedup factors for post-quantum algorithms on GPU. Distinct baselines.

Work	Algorithm	GPU	Reported factor	Baseline
Shen et al. [Shen et al. 2024]	Dilithium	A100	166×–181×	signing vs. CPU reference; verification 88×–109×
Wan et al. [Wan et al. 2022]	Kyber-1024	RTX 3080	26×–36×	KeyGen/Enc/Dec vs. AVX2 reference
Wang et al. [Wang et al. 2024]	SPHINCS+	RTX 3090	up to 18.5×	signing vs. previous fastest implementation
Wu et al. [Wu et al. 2025]	SPHINCS+ (verif.)	RTX 3090	up to 11.3×	verification vs. prior GPU implementations
Lee et al. [Lee et al. 2022]	AES-CTR/FrodoKEM	V100	2.99×; 1,598 Gbps	FrodoKEM vs. Gupta et al.; AES-CTR throughput
This work	FAEST (sign.)	H100	up to 2.77×	signing vs. normalized 16-worker CPU
This work	PERK (sign.)	H100	up to 9.64×	signing vs. normalized 16-worker CPU

standardization criteria extend beyond computational performance and include factors such as security maturity, simplicity, and broad applicability across deployment environments. Nevertheless, the results illustrate that platform assumptions can significantly influence comparative performance assessments. In deployment scenarios where batched GPU execution is available, PERK achieves substantially higher signing throughput than FAEST relative to their respective CPU baselines.

4.4. Critical Analysis of the Results

Scalability with security level. For PERK, the GPU/CPU speedup grows monotonically with the security level, ranging from 5.65× for *perk-1-fast* to 9.64× for *perk-5-fast*. Variants associated with higher security levels incur a higher computational cost per instance, which favors amortizing launch *overhead* and increases effective GPU occupancy. In practical terms, the gain provided by the GPU becomes more significant precisely in scenarios with higher long-term confidentiality requirements.

Signing/verification asymmetry on FAEST. For most variants, the GPU speedup in verification exceeds that in signing (verification 1.99× to 7.59× vs. signing 1.23× to 2.77×), though the two ranges overlap, with FAEST-192s being very close. Several implementation effects contribute to this asymmetry. Signing includes *grinding*, a variable-latency first-success search over counter values: candidate counters can be tested in parallel on the GPU, but the signer must still select the first acceptable counter and continue with that canonical value, which creates ordering, divergence, and load-balance costs. Signing also has to generate, store, and pack large intermediate workspaces for the commitment, VOLE, QuickSilver, and transcript stages. Verification is more regular: the signature already carries the opened material and the grinding counter, so the verifier reconstructs the transcript, checks that the recomputed challenge matches the signed challenge, and performs only a non-interactive grinding-condition test rather than searching over counters.

Structural limitations. Three identified limitations delimit the available space for

improvement.

Arithmetic in $\mathbb{F}_{2^{192}}$. The non-monotonic relative drop from 128f→192f (3.92× drop in signing throughput) versus 192f→256f (2.01×) suggests that the 192-bit path may be under-optimized relative to the neighboring parameter sets. A plausible cause is the three-limb representation, which may reduce packing efficiency or increase register pressure; however, no retained H100 profile establishes this explanation, and confirming it requires profiling the VOLECommit and VOLEHash stages. Alternative memory representations or future PTX/toolchain support for carry-less multiply instructions may mitigate this effect.

Non-optimized PERK verification. The verification operation of PERK was implemented experimentally but was not optimized, resulting in throughput below the CPU baseline. It is used for testing the signature algorithm, but optimizing it was left out of scope. The main stages that are good candidates for optimization are the GGM tree reconstruction and PKP proof evaluation during verification.

Security aspects of GPU. The current implementation seeks to reduce classical timing-leakage sources through fixed *thread* geometry and the elimination of data-dependent early returns. It does not establish data-independent memory access: as noted above, the field-reduction path indexes a lookup table with a data-derived nibble. Formal guarantees of constant-time execution are also difficult in modern GPUs due to microarchitectural effects such as dynamic scheduling, memory contention in multi-tenant environments, intra-warp divergence, and occupancy variations that can introduce data-dependent timing variability. Shared-memory bank conflicts and register allocation deserve explicit mention as additional vectors, but a systematic audit of register spills and shared-memory access patterns for secret dependence has not been performed.

In deployments where adversaries can observe execution times, this variability will require dedicated mitigation. A systematic analysis of GPU-specific side channels is an important direction for future work. Until such a review is performed, any real-world use of the current code should treat this as an open attack vector, reinforce hardware isolation, and avoid shared environments.

Memory layout. Only local memory layout changes were explored. More aggressive restructuring could change the internal representation of workspaces, stage boundaries, and temporary buffers while preserving the externally visible behavior: public keys, signatures, verification results, transcript encodings, and KAT compatibility would remain unchanged. Similar implementation-level trade-offs have been demonstrated for FAEST on embedded devices [Aranha et al. 2025]. In this GPU implementation, however, such changes would require substantial architectural rewriting.

5. Final Considerations

This work presented a GPU implementation and optimizations for the VOLEitH cryptographic primitive, enabling the successful porting of two post-quantum digital signature algorithms to a new platform. Although not always directly shareable, as every VOLEitH based algorithm might have its own way of using the primitives, the techniques demonstrated translated well from FAEST to PERK and might also do the same for other VOLEitH based algorithms.

The experimental results on an NVIDIA H100 GPU support three central conclu-

sions. *First*: the implementation can outperform the normalized 16-worker AVX baseline across all FAEST variants, achieving speedups between 1.23× and 2.77× for signing and between 1.99× and 7.59× for verification. *Second*: the integration of PERK, using the same techniques first applied in FAEST, achieved speedups between 5.65× and 9.64× for signing, with monotonic growth as the security level increases, particularly in scenarios where greater cryptographic robustness is required. *Third*: many bottlenecks were identified, and some optimizations yielded far more gains than others, such as the PERK PKP structure being a natural fit for GPU, while FAEST required more effort and still requires future work to widen the margin over AES-NI CPU gains.

Future work includes gains from alternative memory layouts, AES optimizations, polynomial multiplication, and pipeline/data-dependency analysis; optimizing PERK verification and the FAEST 192-bit variants to close the current benchmark gaps; measuring energy consumption and sig/s/W efficiency; and a systematic side-channel analysis, which is critical before adoption on multi-tenant devices.

The developed infrastructure and the results obtained demonstrate the practical viability of VOLEitH-based post-quantum schemes in high-throughput scenarios. For FAEST, a third-round NIST candidate [Alagic et al. 2026], the results confirm that GPU acceleration is achievable across all submitted variants, with particularly significant gains in verification throughput. For PERK, the infrastructure demonstrates that the VOLEitH architectural discipline generalizes across schemes within the same family and that the GPU performance profile of a scheme may differ substantially from its CPU profile. This observation is relevant to both deployment planning and future standardization discussions involving GPU-capable environments.

6. Use of Generative Artificial Intelligence

Generative AI tools were used for language revision, LaTeX formatting, diagram preparation, and exploratory implementation experiments. All outputs were manually reviewed, and the authors remain fully responsible for the technical content, experimental results, security-related claims, and conclusions.

References

Alagic, G., Bros, M., Ciadoux, P., Dang, Q., Dang, T. H., Kelsey, J., Lichtinger, J., Liu, Y.-K., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Silberg, H., Smith-Tone, D., and Waller, N. (2026). Status report on the second round of the additional digital signature schemes for the NIST post-quantum cryptography standardization process. NIST Internal Report NIST IR 8610, National Institute of Standards and Technology, Gaithersburg, MD.

Alagic, G. et al. (2022). Status report on the third round of the nist post-quantum cryptography standardization process. NIST IR 8413, National Institute of Standards and Technology.

Aragon, N. et al. (2020). BIKE: Bit flipping key encapsulation. Submission to NIST Post-Quantum Cryptography Standardization Round 3.

Aranha, D. F., Degn, J., Eilath, J., Nielsen, K., and Scholl, P. (2025). Faest for memory-constrained devices with side-channel protections. *Cryptology ePrint Archive*.

Baum, C., Braun, L., Saint Guilhem, C. D. d., Klooss, M., Orsini, E., Roy, L., and Scholl, P. (2023). Publicly verifiable zero-knowledge and post-quantum signatures from vole-in-the-head. Cryptology ePrint Archive, Paper 2023/996.

Baum, C. et al. (2025). *FAEST v2: Algorithm Specifications*.

Bernstein, D. J. et al. (2020a). Classic McEliece: Conservative code-based cryptography. Submission to NIST Post-Quantum Cryptography Standardization Round 3.

Bernstein, D. J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., and Schwabe, P. (2020b). SPHINCS+: Stateless hash-based signatures. Submission to NIST Post-Quantum Cryptography Standardization Round 3.

Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehlé, D. (2018). CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE.

Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654.

Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., and Stehlé, D. (2018). CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):238–268.

Fiat, A. and Shamir, A. (1987). How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology–CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer.

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, pages 212–219. ACM.

Ishai, Y., Kushilevitz, E., Ostrovsky, R., and Sahai, A. (2007). Zero-knowledge from secure multiparty computation. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, pages 21–30. ACM.

Joseph, D. et al. (2022). Transitioning organizations to post-quantum cryptography. *Nature*, 605(7909):237–243.

Lee, W.-K., Seo, H. J., Seo, S. C., and Hwang, S. O. (2022). Efficient implementation of aes-ctr and aes-ecb on gpus with applications for high-speed frodokem and exhaustive key search. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(6):2962–2966.

National Institute of Standards and Technology (2022). Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process. Updated October 2022.

PERK Team (2024). Improvements for the perk signature scheme. Submission to NIST Post-Quantum Cryptography Standardization – Additional Digital Signatures Round 2.

Quisquater, J.-J., Guillou, L. C., and Berson, T. A. (1990). How to explain zero-knowledge protocols to your children. In *Advances in Cryptology–CRYPTO'89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 628–631. Springer.

Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Shen, S., Yang, H., Dai, W., Zhang, H., Liu, Z., and Zhao, Y. (2024). High-throughput gpu implementation of dilithium post-quantum digital signature. *IEEE Transactions on Parallel and Distributed Systems*.

Shor, P. W. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE.

Wan, L., Zheng, F., Fan, G., Wei, R., Gao, L., Wang, Y., Lin, J., and Dong, J. (2022). A novel high-performance implementation of crystals-kyber with ai accelerator. In *European Symposium on Research in Computer Security*, pages 514–534. Springer.

Wang, Z., Dong, X., Chen, H., Kang, Y., and Wang, Q. (2024). Cuspx: Efficient gpu implementations of post-quantum signature sphincs+. *IEEE Transactions on Computers*.

Wu, J., Yu, Y., Chen, Z., Yang, H., Li, C., and Liu, Z. (2025). Cbpspx: A cuda-based batch parallel optimization of post-quantum signature sphincs+. *IEEE Internet of Things Journal*.