Accelerating FAEST Signing on GPU via Fused AES Constraint Generation and Batched Leaf Hashing

Ha-Gyeong Kim[1 [0009−0008−5840−5517]], Si-Woo Eum[1 [0000−0002−9583−5427]], Seung-Won Lee[1 [0009−0009−9583−1387]], Ui-Jae Kim[1 [0009−0004−2617−9659]], Min-Ho Song[1 [0009−0007−6277−0069]], and Hwa-Jeong Seo[1 [0000−0003−0069−9061]]

Hansung University, Seoul, Korea
kimhaha4420@gmail.com, shuraatum@gmail.com, dkajdfhd1@gmail.com, chrisvt424@gmail.com, smino0906@gmail.com, hwajeong84@gmail.com

Abstract. FAEST is a symmetric-key post-quantum digital signature scheme and a third-round candidate in the NIST Additional Digital Signatures standardization process. Its signing path concentrates cost in two operations: round-wise constraint generation, which proves in zero knowledge that the AES circuit is computed correctly, and finite-field multiplication, which computes the leaf nodes of a vector commitment. This paper accelerates both operations on a CUDA-enabled GPU, with AES round constraint generation as the main contribution. Specifically, we fuse the three consecutive stages that make up constraint generation into a fused GPU path and keep the intermediate results between stages resident in device memory, thereby reducing host–device data transfer. As a supporting measure, we batch independent finite-field multiplications for leaf nodes, jointly improving end-to-end signing performance. We verify the correctness of the implementation through byte-level output agreement with the CPU reference implementation. Combining fused constraint generation with batched leaf hashing, the full GPU path achieves up to roughly a 3× end-to-end signing speedup over the CPU reference. Furthermore, when AES constraint generation fusion is added on top of leaf hashing that is already performed on the GPU, an additional reduction in signing latency is observed in five of the six evaluated AES s/f variants. The magnitude of this effect does not scale monotonically with the security parameter; rather, it is determined by the per-variant bottleneck structure.

Keywords: Post-quantum digital signature · FAEST · CUDA · GPU acceleration · AES constraint generation · zero-knowledge proof

1 Introduction

Advances in quantum computing threaten RSA and elliptic-curve public-key cryptography through Shor's algorithm [9]. In response, the National Institute of Standards and Technology (NIST) has standardized post-quantum cryptographic

schemes [5] and runs a separate process for additional digital signatures to diversify the underlying assumptions.

FAEST [1] is a post-quantum signature scheme selected as a third-round candidate in this additional digital signature standardization process [6]. It is built on the VOLE-in-the-Head paradigm [2] and, as a symmetric-key design whose only security assumption is the one-wayness of AES, is distinct from lattice-based schemes. The signer demonstrates knowledge of an AES key, without revealing it, by proving in zero knowledge that the AES encryption circuit has been computed correctly. However, because zero-knowledge constraint generation for the AES circuit and vector commitment computation are performed repeatedly during signing, implementation efficiency directly affects the actual signing latency of FAEST. Accelerating the signing path is therefore important for improving the practical deployability of FAEST.

The signing path of FAEST concentrates its cost in two operations. The first is round-wise constraint generation for proving in zero knowledge the correctness of the AES encryption circuit, an operation specific to FAEST. The second is the finite-field multiplication performed to compute each leaf node of the vector commitment, which is repeated thousands to tens of thousands of times per signature depending on the variant. When executed on a single CPU thread, these two operations account for most of the total signing time.

Because these two operations consist of many repetitive tasks, they are amenable to acceleration with the massively parallel processing model of a GPU. Prior GPU-based cryptographic acceleration [8], however, has mainly targeted polynomial operations or hash-tree structures, which differ structurally from FAEST's VOLE-in-the-Head-based AES constraint generation.

This paper targets the AES-based variants of FAEST and accelerates the two operations of the signing path on a CUDA-enabled GPU. We take the AES round constraint generation specific to FAEST as our central contribution. We fuse the three consecutive stages that constitute constraint generation into a fused GPU path and keep the intermediate results between stages resident in device memory, eliminating repetitive host–device data transfer. As a supporting measure, we batch independent finite-field multiplications for leaf nodes, jointly accelerating end-to-end signing performance. This work builds on the reference implementation of FAEST [3]. The main contributions of this work are as follows.

1. We propose a structure that fuses the three consecutive stages of the AES zero-knowledge constraint generation specific to FAEST into a fused GPU path and keeps the intermediate results between stages resident in device memory, eliminating repetitive host–device transfer.
2. We construct a full GPU signing path combined with batched leaf hashing and show that, even when leaf hashing is already performed on the GPU, the fusion of AES constraint generation provides an additional reduction in signing latency.
3. Through measurement, we show that the magnitude of the acceleration effect does not scale monotonically with the security parameter, and that the

per-variant bottleneck proportions of constraint generation and leaf hashing determine the effect.
4. Through hierarchical correctness verification, we confirm that the GPU path produces the same output as the CPU reference on the actual signing path.

This work targets the signing path only, and GPU acceleration of the verification path and the FAEST-EM variants is not within its scope.

2 Background

2.1 FAEST Signing Path

FAEST is based on the VOLE-in-the-Head (VOLEitH) paradigm. Here, Vector Oblivious Linear Evaluation (VOLE) is a tool for encoding the secret bits held by the signer as linearly homomorphic commitments. A VOLE correlation of length ℓ is defined by the relation

$$q_i = u_i \cdot \Delta + v_i, \qquad i = 0, \ldots, \ell - 1 \tag{1}$$

among the bit $u_i \in \mathbb{F}_2$ known only to the signer, a masking element v_i, the global key Δ held by the verifier, and a local key q_i. The masking element v_i hides the bit u_i; opening the commitment to a different value would require the signer to guess the global key Δ, which provides binding. The linearity of this relation plays a central role in the proof. Equalities that must hold among the bits committed by the signer—for example, the condition that each operation of the AES circuit was performed correctly—can be verified through a linear combination of the committed values together with additional proof information. That is, linear homomorphism makes it possible to prove, in zero knowledge, the relations satisfied by secret values without revealing those values.

In FAEST, the verifier's challenge is derived from a hash value through the Fiat–Shamir transform, yielding a non-interactive signature that admits public verification. A FAEST signature can thus be viewed as a non-interactive zero-knowledge proof that, for a public AES input–output pair, the signer knows the secret AES key producing that output without disclosing the key.

Signature generation in FAEST proceeds in several stages, with the following flow. First, the signer expands a GGM tree of length-doubling pseudorandom generators to produce the leaf seeds underlying the VOLE correlation. Next, a commitment is computed for each leaf seed, and these are bound together into a single vector commitment. Depending on the variant, this leaf commitment computation is repeated as thousands to tens of thousands of independent tasks, making it a major bottleneck in signing performance. The signer then uses AES as a one-way function to prove, in zero knowledge and without revealing the key, that it knows the AES key. Here the condition that the AES encryption circuit was computed correctly is expressed as constraints and proved, and this constraint generation also accounts for substantial computational cost. Finally, the proof produced in this process is turned into a non-interactive signature through the Fiat–Shamir transform.

2.2 AES Constraint Generation and Leaf Commitment

In the AES zero-knowledge proof, the signer must show that the AES encryption circuit was computed correctly from the committed input. To this end, the conditions that the operations of each encryption round are correct are expressed as polynomial constraints, and QuickSilver [11] is used to verify that all these constraints evaluate to zero. The most demanding part here is the AES S-box. At the core of the S-box is an inversion over a finite field, and expressing the inversion directly as a constraint raises the degree of the constraint and thus the proof cost. To avoid this, FAEST does not constrain the inversion directly; instead, it uses the relation between Galois conjugates and the inverse norm to express the validity of the S-box as a degree-3 constraint that QuickSilver can handle. This constraint generation is repeated for every encryption round and entails numerous finite-field multiplications in forming and checking each constraint. For this reason, AES round constraint generation becomes a computationally expensive stage of the signing path and is specific to FAEST, whose proof targets AES.

Within AES round constraint generation, we distinguish the three consecutive stages targeted for GPU fusion as Galois conjugate generation, inverse-norm constraint generation, and forward AES round computation. For brevity, we hereafter refer to these stages as S1, S2, and S3, respectively. S1 computes the Galois conjugates for each byte of the current state; S2 generates degree-3 constraints that check the validity of the inverse-norm value; and S3 performs the forward round computation comprising the affine transformation of the S-box, row shifting, column mixing, and round-key addition. The three stages are carried out consecutively within a single round, with the output of each stage serving as the input to the next.

The other axis of cost is leaf commitment computation, the finite-field multiplication performed for each leaf of the GGM tree to build the vector commitment. In the FAEST AES variants targeted in this paper, each leaf value is turned into a commitment through a hash operation involving multiplication over $\mathbb{F}_{2^{3\lambda}}$, and each signature contains thousands to tens of thousands of such leaves. Notably, the two operations share finite-field multiplication as their underlying operation yet differ in parallel structure: leaf commitment consists of fully independent multiplications and is immediately suited to data-parallel processing, whereas AES round constraint generation has ordering dependence across rounds and a narrow parallel width within a round. This difference leads to different acceleration approaches and effects, and the specific target selection is addressed in the next section.

2.3 Related Work

GPU-based acceleration of post-quantum cryptography has been studied actively. For lattice-based signatures, high-throughput implementations that parallelize polynomial multiplication and sampling operations on the GPU have been presented [8], and a GPU implementation of the Falcon signature applying

memory-efficient transformations and adaptive parallelization has also been reported [4]. For hash-based signatures, the hash-tree computation of SPHINCS+ has been accelerated on the GPU [7,10]. The optimization targets of these works are mainly concentrated on the polynomial operations of lattice-based signatures or the tree computation of hash-based signatures. In contrast, this paper targets FAEST's VOLE-in-the-Head-based AES constraint generation and leaf commitment, whose operations and parallel structure differ from these prior targets.

3 Proposed Methodology

3.1 Bottleneck Analysis

The AES-based FAEST variants evaluated in this paper are divided into the s-series and f-series variants according to the security parameter λ and the tree shape. Table 1 summarizes the main parameters of the six variants under evaluation. Even at the same security parameter, the s-series and f-series differ substantially in the number of repetitions τ and the leaf commitment workload L. In particular, the f-series has a shallower tree, so its number of leaves L is smaller than that of the s-series while its signature size is larger; this structural difference leads to the per-variant differences in bottleneck composition analyzed below.

Table 1: Main parameters of the evaluated FAEST AES variants

Variant	λ	τ	ℓ	L	Signature size (B)
FAEST-128s	128	11	1,280	22,528	4,506
FAEST-192s	192	16	2,496	40,960	11,260
FAEST-256s	256	22	3,104	61,440	20,696
FAEST-128f	128	16	1,280	3,072	5,924
FAEST-192f	192	24	2,496	5,120	14,948
FAEST-256f	256	32	3,104	7,168	26,548

To determine the acceleration targets, we performed static and dynamic profiling of the FAEST signing path, using Callgrind to analyze per-function and per-stage instruction shares, perf stat to check compute intensity and hardware counters, and Nsight Systems to inspect CUDA execution, kernels, and host–device transfers.

We measured, per variant, the share of signing time taken by the two operations, summarized in Table 2. Finite-field multiplication in leaf commitment is the largest single operation, accounting for 30% to 64% of total signing time, and in the s-series its share grows with the number of leaves as the security parameter increases. AES round constraint generation accounts for more than 88% within

the AES zero-knowledge proof yet only 15% to 32% of the total signing time, a smaller share than leaf commitment; this difference between two denominators is important for the target selection that follows. The low cache-miss rate further suggests that the main bottleneck is formed by the amount of repetitive finite-field computation rather than by memory access.

Table 2: Per-stage share of the signing path

Variant	Leaf commitment / Sign	Constraint gen. / Sign	Constraint gen. / AES proof
FAEST-128s	30.1%	22.6%	88.0%
FAEST-192s	49.1%	31.7%	96.0%
FAEST-256s	63.7%	15.3%	93.1%

3.2 Target Selection and Single-Stage Limitation

Both operations are based on finite-field multiplication but differ in character as acceleration targets: leaf commitment holds the largest share of signing and is fully independent, which favors end-to-end performance, yet it is a general-purpose operation unrelated to AES, whereas AES round constraint generation, though a smaller share, is the FAEST-specific core operation that proves in zero knowledge the validity of the AES circuit. We therefore take the acceleration of FAEST-specific AES round constraint generation as the central contribution and combine batched leaf hashing, which dominates end-to-end performance, as a supporting role; this preserves the focus on a FAEST-specific operation while reducing the dominant cost of the signing path.

As a first attempt to move AES round constraint generation to the GPU, we took S3 (the forward AES round computation) as a standalone target, since it is byte-level independent, free of constraint-accumulation dependence, and thus maps simply onto the GPU. This approach, however, revealed two structural limitations: a single round within one signature exposes only a narrow amount of parallel work, insufficient to fill the thousands of cores of a GPU, and the input and output of S3 had to be exchanged between host and device at every round, so the transfer cost recurred as many times as the number of rounds. As a result, accelerating only a single isolated stage did not reduce end-to-end latency reliably, showing that the limitation arises not from the kernel itself but from insufficient workload and repetitive transfers—which calls for a design that processes consecutive stages together to increase the workload and eliminate inter-stage transfer.

3.3 Fusion Design

The three stages that make up AES round constraint generation—Galois conjugate generation (S1), inverse-norm constraint generation (S2), and forward AES

round computation (S3)—are carried out consecutively within a single round, where the output of one stage becomes the input to the next. This work fuses these three stages into a fused GPU path.

This fusion design targets the two diagnosed limitations at once: grouping the three stages increases the workload processed by the GPU at once, improving compute-resource utilization, while using the intermediate results between stages directly during the round reduces the per-round host–device transfer. Meanwhile, batched leaf hashing is combined as a separate path independent of this fusion path, offloading the two bottlenecks together. Figure 1 contrasts the single-stage approach with the proposed fusion path; its kernel realization and correctness verification are addressed in the next section.

4 Implementation

4.1 Fusion Path of AES Constraint Generation

The three stages each require a different thread configuration, so our implementation does not merge them into a single kernel; instead, it forms a fusion path that executes three kernels in sequence, each matched to the parallel structure of its stage. The key to the fusion is keeping the intermediate results between stages resident in device memory: when a preceding kernel writes its result to a device buffer, the next kernel takes the same device pointer as input, so intermediate results do not return to the host at each kernel boundary. As a result, within a round the data sent to the device is limited to the external input and the data retrieved from it to the final outputs, reducing the per-round host–device round trips described in the previous section.

4.2 Separation of the Constraint-Accumulation Operation

Among the three stages, inverse-norm constraint generation must accumulate the constraints it produces in a fixed order and therefore carries a sequential dependence, which is the greatest obstacle to grouping the three stages on the GPU. Our implementation resolves this by separating the accumulation itself from the computation of its operands: because the operands are determined solely by the input at each position, independently of the accumulation's progress, the GPU generates the per-round operands in parallel while only the light, order-dependent accumulation is performed in the original sequence. This allowed even the stage with sequential dependence to be included in the fusion path while preserving the sequential-dependence structure intact.

4.3 Batch Parallelization of Leaf Hashing

Because the computation of each leaf is fully independent, our implementation organizes leaf hashing into three steps: the host generates per-leaf seeds with a pseudorandom generator and gathers the input buffers, the GPU assigns one

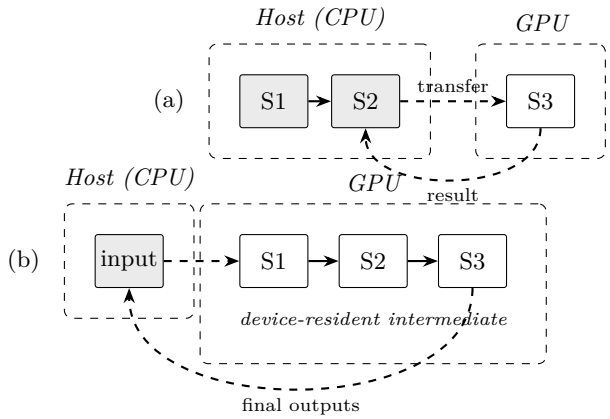


Fig. 1: Comparison of single-stage GPU offloading and the proposed S1–S3 fusion path

thread to each leaf and processes all leaves' finite-field multiplications in a single batch, and the host then reflects the results into the commitment array. The seed generation and commitment accumulation, which must proceed in order along the tree, are kept on the host while only the per-leaf independent multiplications run on the device, so the thousands to tens of thousands of leaves per signature are computed in one call. This path handles different data from the AES constraint generation fusion and runs at a different point in the signing path, so the two accelerations combine without interference while improving end-to-end performance by accelerating leaf commitment, the largest share of the signing path.

4.4 Finite-Field Multiplication and Data Packing

The finite-field multiplication shared by both operations is internally sequential, but multiplications on different inputs are independent, so we process many independent multiplications in batches; we also verify the multiplication independently first, so that the cause of any later error can be separated between the multiplication itself and the per-stage data layout. For host–device transfers, we use a stride based on the actual byte width of the element fixed for each security parameter, rather than the memory size of the data structure holding it. This prevents the silent boundary misalignment that would otherwise produce incorrect outputs without runtime errors, a mismatch detectable only through the hierarchical correctness verification of the next subsection.

4.5 Correctness Verification

To confirm that the GPU path correctly replaces the CPU reference implementation, we applied a four-level hierarchical correctness verification procedure. Each level and its result are summarized in Table 3.

In L1 and L2, synthetic inputs confirmed byte-level CPU–GPU agreement for each individual stage and for the connected fusion path, verifying that the final result is correct even when intermediate values remain resident on the device. In L3, shadow verification compared the GPU output against the real signing-path intermediates without affecting the actual signature computation, so correctness could be confirmed safely over the intermediate values that change at every run; in L4, the CPU computation path was excluded from compilation so that the GPU output became the sole source used in signing, and the generated signatures passed verification.

After establishing this procedure on one security parameter, we applied it to the remaining s-series variants by replacing the finite field and constants while keeping the thread configuration unchanged, and all three passed the four levels. The f-series, which shares the same AES round structure as the corresponding s-series variants, reused the same kernels and therefore did not repeat the four levels separately; its correctness was confirmed through per-variant wiring and real signing and verification.

5 Evaluation

5.1 Experimental Setup

All measurements were performed on a single server equipped with an RTX 4080 GPU and a desktop CPU. Signing latency was measured for a 76-byte message. For each measurement, we took the median over 50 signatures after excluding one warm-up run, and report the mean and sample standard deviation over five independent runs. To reduce interference from other load during measurement, the runs were pinned to a single performance core.

One asymmetry exists in the build conditions: the CPU+CPU baseline used link-time optimization, whereas the three combinations containing CUDA objects did not, so the speedups of the GPU combinations over the CPU+CPU baseline are conservative. By contrast, the core comparison, CPU+leaf versus S1S2S3+leaf, uses no link-time optimization in either case, so the additional effect of AES constraint generation fusion is compared without bias.

5.2 Comparison Configurations

To separate the contributions of AES constraint generation fusion and leaf hashing, we measured signing latency under four configurations that turn the two accelerations on and off individually. CPU+CPU is the baseline that performs both operations on the CPU; CPU+leaf performs only leaf hashing on the GPU;

Table 3: Hierarchical correctness verification

Level	Target	Verification content	Result
L1	Individual stage	CPU–GPU match of per-stage output on synthetic input	PASS
L2	Fusion path	CPU–GPU match of output after connecting the three stages	PASS
L3	Real intermediate values	Output match on intermediate values of the real signing path	PASS
L4	Real signature	Verification pass of a signature generated from GPU output	PASS

S1S2S3+CPU performs only AES constraint generation fusion on the GPU; and S1S2S3+leaf is the configuration that applies both accelerations.

Among these, the core comparison of this paper is the change from CPU+leaf to S1S2S3+leaf. This represents the end-to-end effect of adding AES constraint generation fusion on top of the condition where leaf hashing is already performed on the GPU. Because both configurations are measured under the same build condition, this comparison is unaffected by build-condition differences and provides the fairest basis for isolating the effect; we therefore take it as the main result of this paper.

5.3 Signing Latency Results

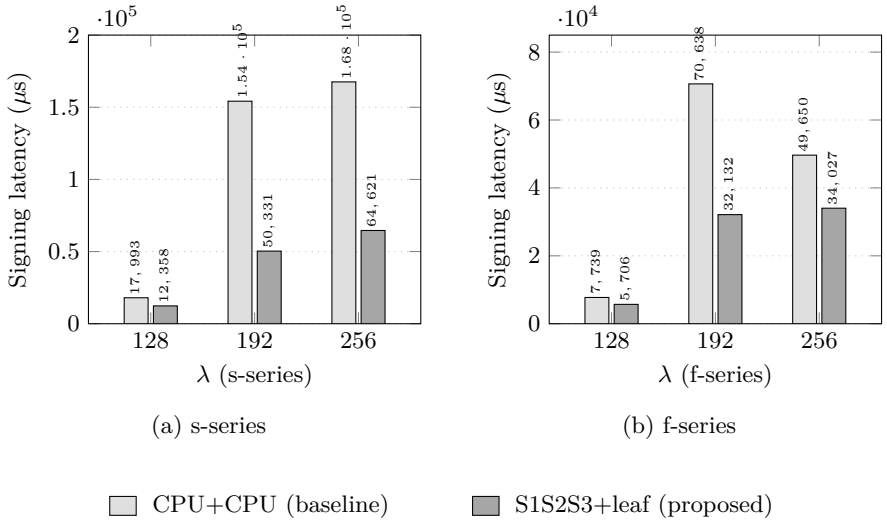
Table 4 summarizes the four-configuration signing latency and effect decomposition for the s- and f-series variants.

We first compare the baseline CPU execution with the full proposed method. Contrasting the baseline that performs both operations on the CPU (CPU+CPU) with the proposed method that applies both accelerations (S1S2S3+leaf), the end-to-end signing latency is reduced by 1.46× to 3.06× over the CPU baseline depending on the variant, with the largest speedup of 3.06× at 192s (Figure 2). This end-to-end gain is composed of the two accelerations, leaf hashing and AES constraint generation fusion; to isolate the additional effect of the fusion that is the central contribution of this paper, we compare adding the fusion (S1S2S3+leaf) on top of the condition where leaf hashing is already performed on the GPU (CPU+leaf). The effect of this core comparison differs greatly across variants. For 192s, signing latency decreases by 38.1%, a clear reduction that is well beyond the run-to-run variation, and for 256s the decrease is 6.1%, a small but clear reduction that exceeds the range of variation. For 128s, by contrast, the mean latency was 1.4% lower, but this difference is close to the range of run-to-run variation, so we do not claim it as an additional effect.

For the f-series, the core-comparison effect is clearly observed in all three variants, with signing latency decreasing by 12.3% to 47.2%. Since we did not perform per-stage profiling for the f-series, we do not attribute the reductions to a specific bottleneck; instead, the four-configuration effect decomposition shows

Table 4: Signing latency and effect decomposition for the s- and f-series

Variant	CPU+CPU (μs)	CPU+leaf (μs)	S1S2S3 +CPU (μs)	S1S2S3 +leaf (μs)	Combined	Added on leaf
128s	17,993±94	12,532±208	16,764±256	12,358±180	1.46×	−1.4%
192s	154,195±734	81,258±1,153	121,306±713	50,331±1,260	3.06×	−38.1%
256s	167,565±66	68,817±112	172,163±1,198	64,621±26	2.59×	−6.1%
128f	7,739±18	6,883±13	6,404±17	5,706±7	1.36×	−17.1%
192f	70,638±56	60,880±34	41,060±33	32,132±42	2.20×	−47.2%
256f	49,650±106	38,816±56	49,379±72	34,027±21	1.46×	−12.3%



(a) s-series (b) f-series

Fig. 2: End-to-end signing latency of the CPU baseline (CPU+CPU) versus the full proposed method (S1S2S3+leaf), by security parameter for the s- and f-series. The proposed method reduces signing latency for every variant, with the largest reduction at $\lambda = 192$ in both series.

that AES constraint generation fusion contributed to the end-to-end latency reduction.

5.4 Effect Analysis

One important point evident from the measurements is that the magnitude of the AES constraint generation fusion effect does not scale monotonically with the security parameter. In the core comparison, the largest effect appears at the 192 variant for both the s-series and the f-series, not at the 256 variant. The end-to-end combined speedup was also largest at 3.06× for 192s.

This non-monotonicity is interpreted as arising because the share that AES constraint generation and leaf hashing occupy in signing differs across variants. As Table 1 shows, even at the same security parameter the s-series and f-series differ greatly in the leaf commitment workload L, so the relative shares of leaf hashing and AES constraint generation are formed differently across variants. In the s-series, AES constraint generation holds the largest share at 192s, so accelerating it yields a large end-to-end effect; at 256s, leaf hashing dominates and the AES constraint-generation share is low, yielding a smaller relative gain. In the f-series as well, the largest decrease is observed at 192f, but because we did not perform per-stage profiling, its cause is interpreted only within the scope of the four-configuration effect decomposition. Therefore, the simple extrapolation that the effect of AES constraint generation fusion grows as the security parameter increases does not hold under measurement; the magnitude of the effect is determined by the per-variant bottleneck structure.

6 Conclusion

This paper presented CUDA-based GPU acceleration for two core operations in the FAEST signing path. The central contribution is the fusion of the three consecutive stages that make up FAEST-specific AES round constraint generation into a fused GPU path. By keeping intermediate results between stages resident in device memory, this fusion reduced the host–device transfers that would otherwise recur at every round. As a supporting measure, we batched the finite-field multiplications in leaf commitment, which hold the largest share of the signing path, to improve end-to-end performance. Through a four-level hierarchical verification spanning from synthetic input to the real signing path, we confirmed that the GPU path correctly replaces the CPU reference.

On an RTX 4080 GPU, the full GPU path combining fused constraint generation with batched leaf hashing achieved up to roughly a 3× end-to-end signing speedup over the CPU reference. Furthermore, when AES constraint generation fusion was added on top of leaf hashing already performed on the GPU, end-to-end signing latency decreased in five of the six evaluated AES s/f variants. The magnitude of this effect did not scale monotonically with the security parameter but was determined by the per-variant bottleneck structure, with the largest effect appearing at the 192 variant for both the s-series and the f-series. These results suggest that the effect of acceleration should be judged by which operation dominates the signing cost in each variant, rather than extrapolated from the security parameter alone. Beyond this, through implementation and measurement, this work showed that the signing path of FAEST, which is based on symmetric-key primitives, can be a practical target for GPU parallelization.

This work targets only the signing path of AES-based FAEST variants in a single-GPU setting, focusing on the latency of a single signature; verification, the FAEST-EM variants, and throughput-oriented multi-signature batching are outside its scope. Future work includes raising GPU utilization for AES constraint

generation under its limited intra-signature parallelism, and further optimizing transfers with pinned host memory and asynchronous execution.

References

1. Baum, C., Braun, L., Munch-Hansen, A., Razet, B., Scholl, P.: FAEST: Algorithm specifications and supporting documentation, version 2.0.4. https://faest.info (2024)
2. Baum, C., Braun, L., Delpech de Saint Guilhem, C., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In: Advances in Cryptology – CRYPTO 2023. Lecture Notes in Computer Science, vol. 14085, pp. 581–615. Springer (2023). https://doi.org/10.1007/978-3-031-38554-4_19
3. FAEST Project: FAEST reference implementation (version 2). https://github.com/faest-sign/faest-ref (2024)
4. Li, W., Wei, H., Shen, S., Yang, H., Dai, W., Zhao, Y.: cuFalcon: An adaptive parallel GPU implementation for high-performance Falcon acceleration. IEEE Transactions on Parallel and Distributed Systems **37**(5), 1153–1167 (2026). https://doi.org/10.1109/TPDS.2026.3675891
5. National Institute of Standards and Technology: Post-quantum cryptography standards: FIPS 203, FIPS 204, and FIPS 205 (2024), https://csrc.nist.gov/projects/post-quantum-cryptography
6. National Institute of Standards and Technology: Status Report on the Second Round of the Additional Digital Signature Schemes for the NIST Post-Quantum Cryptography Standardization Process. Tech. Rep. NIST IR 8610, National Institute of Standards and Technology (May 2026), https://csrc.nist.gov/pubs/ir/8610/final
7. Ning, Y., Dong, J., Lin, J., Zheng, F., Fu, Y., Xiao, F.: GRASP: Accelerating hash-based PQC performance on GPU parallel architecture. IEEE Transactions on Computers **75**(4), 1579–1592 (2026). https://doi.org/10.1109/TC.2026.3654650
8. Shen, S., Yang, H., Dai, W., Zhang, H., Liu, Z., Zhao, Y.: High-throughput GPU implementation of Dilithium post-quantum digital signature. IEEE Transactions on Parallel and Distributed Systems **35**(11), 1964–1976 (2024). https://doi.org/10.1109/TPDS.2024.3453289
9. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Review **41**(2), 303–332 (1999). https://doi.org/10.1137/S0036144598347011
10. Wang, Z., Dong, X., Chen, H., Kang, Y., Wang, Q.: CUSPX: Efficient GPU implementations of post-quantum signature SPHINCS+. IEEE Transactions on Computers **74**(1), 15–28 (2025). https://doi.org/10.1109/TC.2024.3457736
11. Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 2986–3001. ACM (2021). https://doi.org/10.1145/3460120.3484556