LESS on the Cortex-M4: Characterizing the Speed–Memory Design Space of Code-Equivalence Signatures

Minwoo Lee[1], Minjoo Sim[1], Subeen Cho[2], Yulim Hyoung[2] and Hwajeong Seo[2]

[1] Department of Information Computer Engineering, Hansung University, Seoul, Republic of Korea
{minunejip, minjoos9797}@gmail.com
[2] Department of Convergence Security, Hansung University, Seoul, Republic of Korea
{chosubin1208, yulim4hyoung}@gmail.com, hwajeong@hansung.ac.kr

Abstract. LESS is a code-based signature scheme built on the linear equivalence problem and, in its v2.0 round-2 form, a candidate in the NIST call for additional post-quantum signatures. No microcontroller implementation of it has been reported: the official benchmarking effort for the additional signatures excluded it on memory grounds, and an x86-massif cross-check puts the reference's peak stack at up to ≈836 KB — beyond the SRAM of even the largest mainstream Cortex-M4. This paper provides the first such implementation, to the best of our knowledge — a complete characterization of LESS (v2.0) on the Arm Cortex-M4 across all seven parameter sets. NIST did not advance LESS to the third round in May 2026; we develop the implementation techniques and the evaluation methodology to outlast that outcome.

We map the speed–memory design space with three operating points, all byte-identical to the NIST known-answer tests. The primary point is a *balanced* configuration signing 4.1–4.3× faster than the platform-normalized reference *while* using 41–59 % less peak stack. With double-pass signing at category 5 it is the only configuration whose signing we measure on target across all seven sets — at category 5 the resident-matrix configurations exceed the device's 640 KB of SRAM. It is flanked by a *speed* ceiling at 4.5–4.7× (at reference-level memory) and a *memory* floor that signs with about a one-kilobyte measured stack at only 1.2–1.5× reference time at category 1, with key generation on par with or below reference cost — so at category 1 memory minimization is nearly free.

The speedups rest on three increasingly platform-specific layers: algorithmic restructuring, intrinsic-free two-lane SWAR (SIMD-within-a-register) arithmetic for $\mathbb{F}_{127}$ with a lazy-reduction scheme, and hand-scheduled assembly kernels — including a rank-2 row-elimination kernel — together with a systematic audit of all kernels against the Cortex-M4 timing rules. Read against the reference's own profile, these results indicate that implementation maturity, not the scheme, accounts for a $> 4\times$ swing in achievable microcontroller signing — a data point for standardization-time performance evaluation, offered without revisiting the decision. We further report a catalog of negative results (techniques that help on desktop SIMD but hurt on the M4) and an alignment-safe transpose that returns ≈170 KB of static memory to the stack under an `-O3` build. The evaluation methodology — Pareto-monotone adoption criteria, byte-identical correctness gates, and coverage and ablation reporting — transfers to the surviving third-round candidates, for which NIST has explicitly called for constrained-device evaluation.

Keywords: Post-quantum cryptography · Digital signatures · LESS · Code equiv-

alence · Arm Cortex-M4 · Embedded implementation · SWAR · Memory optimization · Design-space exploration

1 Introduction

Code-based cryptography contributes some of the most mature mathematical foundations to post-quantum standardization, and within it, signatures built from *equivalence problems* form a distinct family: instead of decoding, the signer proves knowledge of a map connecting two equivalent codes. LESS [BMPS20, BBB+25] instantiates this idea with the *Linear Equivalence Problem* (LEP): the secret is a monomial transformation linking two generator matrices over a small prime field, and a Fiat–Shamir-compiled sigma protocol turns it into a signature. LESS was submitted to the NIST call for additional post-quantum signatures and advanced to its second round, with the second-round specification (v2.0) adopting canonical-form (CF) commitments [CPS25, BEP+25] that substantially shrink signatures.

In May 2026, NIST concluded the second round of the additional-signature process: nine schemes advanced, and LESS was not among them [Nat26]. The status report cites three reasons: large public keys, slower keygen/sign/verify than competing code-based candidates, and a second-round attack by Budroni et al. [BEFN25] that reduced the effective security of the parameter sets by an estimated 12–24 bits depending on category. We state this plainly at the outset: this paper characterizes a scheme that is, as of this writing, out of the standardization race.

Why study its embedded implementation anyway? Three reasons. *First*, the research line LESS belongs to is active independently of the NIST outcome: linear code equivalence continues to seed new constructions — e.g., blind signatures [LP25] — and LESS remains the canonical, best-studied LEP instance, so those constructions reuse its computational core (RREF, canonical forms, monomial actions) essentially unchanged, and implementation data for that core outlives any single scheme's standardization status. *Second*, LESS is a stress test for a class of workloads — rejection-sampled matrix expansion, Gaussian elimination over a small prime field, and canonicalization by sorting — that recurs across matrix-centric post-quantum schemes; as we show, our memory direction and evaluation methodology are not LESS-specific, and even the field-specific arithmetic transfers to same-field designs (Section 11). *Third*, the constrained-device questions NIST explicitly asks of the surviving third-round candidates (optimized implementations, performance on constrained devices [Nat26]) require methodology: adoption criteria, correctness gates, and reporting formats for speed–memory trade-offs. This paper develops and validates that methodology on LESS, and we transfer it to third-round candidates in follow-up work.

There is also a concrete gap to close. As of July 2026 we are aware of no Cortex-M4 — or any microcontroller — implementation of LESS. The benchmarking effort that extended pqm4 [KRSS19] to the additional signature schemes explicitly excluded LESS because of its memory demands [KKPY24], citing intermediate buffers far beyond microcontroller budgets. That assessment predates the round-2 reference, which no longer allocates such buffers; its appetite nonetheless remains far outside embedded norms: the reference's peak stack (maximum over key generation, signing, and verification) ranges from 167 KB at category 1 up to 836 KB at category 5 (on-target where it fits, otherwise an x86-massif cross-check on the identical host build; Section 6.5) — the category-5 figure exceeding even the 640 KB SRAM of the largest mainstream Cortex-M4 parts, before counting keys.

This work. We make LESS run on the Cortex-M4 across its entire parameter space, and characterize the full speed–memory design space: from a memory floor that signs category-1 parameters with about one kilobyte of stack at 1.2–1.5× the reference signing time — so that, contrary to the usual cost of streaming recomputation, memory minimization is nearly free at this scale — to a speed ceiling 4.5–4.7× faster than the platform-normalized

reference,[1] with a *balanced* operating point that strictly dominates the reference on both axes simultaneously — 4.1–4.3× faster at 41–59 % less peak stack. A single memory wall at category 5 separates the resident-matrix configurations (reference and speed ceiling) from the two memory-lean policies that clear it; the balanced knee, with double-pass signing, is the one whose signing we measure on target across all seven sets. All reported configurations are verified byte-identical against the official NIST known-answer tests; every adoption decision passed through that gate. Rather than presenting one implementation, we present the design space itself: which memory techniques buy how much, which speed layers contribute what, and where the lone memory wall falls and which configurations clear it.

A systems observation. One of the cited factors in the second-round decision was performance. Our results show that the performance profile of the *reference* implementation understated what the scheme admits on a microcontroller by a factor of more than four in signing — a gap we close using only C-level and ISA-level engineering, with no algorithmic changes to the scheme. The gap is specific to the constrained-device evidence base: the scheme's desktop performance had already been optimized in its own vector-ISA work [BEP+25], but no microcontroller implementation existed — pqm4 excluded it on memory grounds [KKPY24] — so the record we correct was empty, not merely immature. We offer this as a data point on the role of implementation maturity in standardization-time performance evaluation, not as a critique of the decision: the attack-driven security loss and the public-key sizes, the other two cited factors, are not addressable by implementation.

Contributions.

- **C1 — First complete M4 characterization.** The first complete characterization, to the best of our knowledge, of LESS (v2.0) on the Arm Cortex-M4, covering all seven parameter sets — including category 5, reachable only by the memory-lean configurations — for a scheme previously excluded from microcontroller benchmarking on memory grounds (Section 6).
- **C2 — A principled speed–memory design space.** A three-point design space — a one-kilobyte-stack memory floor, a balanced knee, and a 4.5–4.7× speed ceiling — in which the balanced point strictly dominates the reference on both axes simultaneously, the three optimized points are mutually non-dominated, and membership is governed by explicit adoption criteria (B1/B2, Section 3.5).
- **C3 — Portable and ISA-level technique layers.** Intrinsic-free two-lane SWAR arithmetic for $\mathbb{F}_{127}$ with a lazy-reduction scheme; a rank-2 RREF elimination kernel; and a systematic timing-rule-driven kernel audit against the Cortex-M4 TRM — with the assembly kernels validated by per-kernel ablation (Section 5).
- **C4 — Engineering findings of independent value.** A category-5 memory-wall analysis with a double-pass enabler; a catalog of negative results — techniques effective on desktop SIMD that *hurt* on the M4; and an alignment-safe transpose whose diagnosis, via the fault-status register rather than the memory map, returns ≈170 KB of static memory to the stack under an `-O3` build (Section 8).
- **C5 — A reusable evaluation methodology.** The apparatus that produced C1–C4 — Pareto-monotone adoption criteria (B1/B2), a three-stage byte-identical correctness regime, and coverage-matrix and per-kernel-waterfall reporting — is a

[1] We normalize the reference's Keccak backend to the one all our configurations use; this is conservative — it changes the reference's own signing time by only a few percent (Section 6) — so the headline gap reflects our techniques, not a hash swap.

method for characterizing a scheme's constrained-device design space. We apply it to LESS and carry it to the surviving third-round candidates, which is where NIST has asked the community to look (Sections 3 and 11).

Organization. Section 2 recalls LESS and the target platform. Section 3 defines configurations, correctness gates, measurement protocol, and adoption criteria. Sections 4 and 5 present the memory- and speed-direction techniques. Section 6 reports coverage, performance, memory, and ablations. Section 7 discusses security considerations, Section 8 negative results and engineering lessons, and Section 9 related work. Section 10 discusses operating-point selection and the limitations of the study, and Section 11 concludes.

2 Background

2.1 LESS and the Linear Equivalence Problem

Let $q = 127$ and let $G \in \mathbb{F}_q^{k\times n}$ be a generator matrix of an $[n, k]$ linear code; all LESS parameter sets use $n = 2k$. Two codes are *linearly equivalent* if their generator matrices are related by $G_1 = S\,G_0\,Q$, where $S \in \mathrm{GL}_k(\mathbb{F}_q)$ is a basis change and Q is an $n \times n$ *monomial* matrix (a permutation with nonzero column scalings). The Linear Equivalence Problem (LEP) asks to recover such a Q given (G_0, G_1). Because the basis change S can be normalized away by reducing to *reduced row echelon form* (RREF), equivalence classes are compared via RREF representatives.

LESS [BMPS20, BBB+25] builds a sigma protocol from this group action: the public key contains $s-1$ matrices $G_i = \mathrm{RREF}(G_0\,Q_i)$ for secret monomials Q_i (plus the seed of a random G_0); a signature on message m executes T parallel rounds. In each round the signer samples an ephemeral monomial $\widetilde{Q}_j$ and commits to the equivalence class of $G_0\widetilde{Q}_j$. A fixed-weight challenge string (weight w out of T, derived from the hash of all commitments and m) then selects, per round, either the ephemeral seed itself (zero challenge) or a monomial linking the challenged key matrix to the commitment. The second-round specification (v2.0) commits not to a matrix but to a *canonical form* (CF) of the code [CPS25, BEP+25]: a single distinguished representative of the matrix's class under the residual monomial action (column permutations and scalings), so that any two matrices related by that action hash to the same commitment. Computing the CF therefore normalizes each row by a scaling and then puts the rows and columns into a canonical order (a lexicographic sort, with a transpose so that columns can be sorted as rows), the scalings and sort keys carried under multiplicative *blinding* argued to prevent the sort order from revealing the secret monomial (we defer the timing classification to Section 7). This is both why the canonical form shortens responses and signatures and why row scaling, sorting, and transposition dominate its cost (Section 5). Unopened rounds are covered by a GGM seed tree, and fixed-weight challenge sampling uses a Batcher odd–even network [BEP+25].

Computational profile. For implementation purposes, the scheme's cost decomposes into five recurring blocks, which we refer to throughout:

1. *PRNG expansion*: SHAKE-based rejection sampling of matrices and monomials, and GGM seed-tree derivation;
2. *RREF*: Gaussian elimination of $k \times n$ matrices over $\mathbb{F}_{127}$— the dominant block (about two thirds of signing time in our profiles);
3. *Canonical form (CF)*: row scalings, zero counting, histogram/counting sorts, lexicographic row and column sorting with transposes, computed under multiplicative blinding [BEP+25];

Table 1: LESS parameter sets and resulting sizes, from the round-2 specification [BBB+25] (Table 4). All sets use $q = 127$ and $n = 2k$; T = rounds, w = challenge weight, s = number of generator matrices (the public key holds $s-1$ of them plus a seed). Signature sizes are worst-case.

Set	Cat.	k	T	w	s	|pk|	$\|\sigma\|$
LESS-252-45	1	126	45	34	8	97.5 KB	1,329 B
LESS-252-68	1	126	68	42	4	41.8 KB	1,825 B
LESS-252-192	1	126	192	36	2	13.9 KB	2,625 B
LESS-400-102	3	200	102	61	4	105.2 KB	4,131 B
LESS-400-220	3	200	220	68	2	35.1 KB	6,329 B
LESS-548-137	5	274	137	79	4	197.3 KB	7,436 B
LESS-548-345	5	274	345	75	2	65.8 KB	10,680 B

4. *Monomial arithmetic*: application and composition of permutations with column scalings;

5. *Hashing*: absorbing commitments and the message.

Within each category, the parameter sets (Table 1) trade public-key size against signature size and signing time: larger s shortens signatures but multiplies key generation cost (proportional to $s-1$) and public-key size, while larger T lengthens signing and verification (proportional to T).

2.2 Target Platform: Arm Cortex-M4

The Cortex-M4 [Arm15] is the de facto NIST evaluation target for embedded post-quantum cryptography. It is a 3-stage, in-order, single-issue core without dynamic branch prediction (taken branches cost 2–3 cycles), with single-cycle 32×32 multiplication and the ARMv7E-M DSP extensions (e.g., `UXTB16`, `USADA8`, `SMULBB/SMULTT`, `PKHBT/PKHTB`, `SMLAD`). Two properties of its load/store timing, documented in the technical reference manual [Arm15] and central to Section 5.4, are: (i) *load chaining* — ℓ consecutive loads execute in $\ell+1$ cycles, but a chain broken by an ALU instruction loses the pipelining; and (ii) *address generation (AGEN) stalls* — a load or store whose address derives from the immediately preceding ALU result stalls one cycle.

Our board is the ST Nucleo-L4R5ZI (STM32L4R5ZI [STM21]): 640 KB of SRAM and 2 MB of flash — the largest SRAM in the mainstream Cortex-M4 line, chosen deliberately so that *reference* configurations have the best possible chance to run, making coverage gaps attributable to the scheme rather than to an unusually small device. At the 20 MHz benchmark clock the flash operates with zero wait states, so cycle counts reflect pure compute cost (the pqm4 convention); at the 120 MHz development clock, flash wait states (mitigated by the ART cache) raise cycle counts by a few to about twenty percent for the resident and in-place configurations, growing with memory-traffic intensity — and by far more for the flash-fetch-bound streaming floor (Appendix B).

2.3 pqm4 and Benchmarking Conventions

We integrate all configurations into the pqm4/mupq harness [KRSS19]: identical wrappers, the platform-standard assembly Keccak permutation, DWT-cycle-counter measurement, and the mupq stack-watermarking method. pqm4's additional-signature benchmarking campaign [KKPY24] covered the round-1 additional signature schemes on this same platform class but excluded LESS on memory grounds; our memory accounting (Section 3.3) follows pqm4 conventions (peak stack as the primary metric) and additionally reports

Table 2: Configurations. All percentages and speedups in this paper are relative to ref-norm. All builds link the pqm4 assembly Keccak except ref-sub, which uses the bundled C Keccak. bal-c and spd-c appear in the layered ablation (Table 6); the pure-C build (C Keccak throughout) is used only as the x86 KAT gate and is not benchmarked on the M4.

Name	Description	Role
ref-sub	reference v2.0, as submitted	out-of-the-box context (one line)
ref-norm	reference v2.0, platform-normalized	**baseline for all deltas**
mem-max	streaming/recomputation build	memory floor
bal-c	in-place + portable C speed layers	balanced, C layer (ablation)
bal-asm	bal-c + assembly kernels	**balanced knee; full coverage**
spd-c	resident-matrix + portable C speed layers	speed, C layer (ablation)
spd-asm	spd-c + assembly kernels	speed ceiling

total peak RAM, so that results remain comparable both with pqm4 tables and with whole-system budgets.

3 Methodology

This section fixes the vocabulary and the rules under which every number in this paper was produced: the configurations we compare, the correctness gates each of them passed, the measurement protocol, the hashing policy that makes deltas attributable, and the adoption criteria that define the balanced configuration.

3.1 Configurations

Table 2 defines the seven primary configurations; the memory floor additionally has an inline-assembly arithmetic variant (Section 4.6), reported at the representative parameter set. They are not independent codebases grown apart: all optimized configurations derive from the round-2 reference, share the correctness gates, and differ by explicitly documented technique sets (the artifact preserves a 1:1 mapping between paper names, source trees, and build flags).

Two structural choices deserve emphasis. First, *speed* and *balanced* are not separate optimization efforts but two memory policies over the same technique stack (Sections 4–5): the speed configurations keep the working matrices resident (reference-level stack), while the balanced configurations apply the in-place memory techniques and accept the recomputation they imply. Second, at category 5 the balanced configuration additionally enables double-pass signing (Section 4.3); we mark those cells explicitly.

3.2 Correctness Gates

Every adopted change — including build-flag and Keccak-linkage changes — had to re-pass three gates before any measurement was recorded:

G1 (x86 KAT). The official NIST known-answer tests, 100 vectors per parameter set: the produced `.rsp` must be byte-identical to the reference's. The gate runs on the pure-C build; for the large category-5 sets we built a 16-way parallel KAT runner that exploits the per-vector independence of the DRBG stream (it re-derives the

master stream, partitions the vector range, and reassembles a byte-identical `.rsp`), reducing, e.g., LESS-548-345 from hours to under six minutes (Appendix A).

G2 (M4 cross-build equivalence). On-target deterministic testvectors: the full transcript of keypairs, signatures, and verification results (≈1.8 MB of hex output) must be byte-identical between the C build and the assembly build, pinning the assembly kernels to the C semantics on the actual hardware.

G3 (M4 roundtrip). Sign–verify roundtrip on target, including rejection of a tampered signature.

Since FIPS-202 outputs are implementation-independent, KAT validity established with the C Keccak transfers verbatim to builds linking the assembly Keccak; this is what makes G1 (x86) meaningful for M4 builds. The gates are not ceremony: G1 caught a real bug during development — a classic word-wise zero-byte detector ($(w -$ `0x01..01`$) \,\&\, \sim w \,\&$ `0x80..80`) whose borrow propagation falsely flags a byte adjacent to a genuine zero, silently changing a canonical-form candidate selection. The M4 measurements looked healthy; only the byte-identical gate exposed it (we replaced it with a carry-free form, Section 5.2).

3.3 Measurement Protocol

Cycles. All headline numbers are measured on a Nucleo-L4R5ZI at **20 MHz** (`CLOCK_BENCHMARK`): at this clock the flash serves with zero wait states, so counts reflect pure compute cost and follow the pqm4 reporting convention [KRSS19]. The 120 MHz development clock (`CLOCK_FAST`), where flash wait states and the ART cache come into play, is reported in Appendix B. Cycles are collected with the DWT cycle counter through the mupq harness; we abbreviate 10^6 cycles as Mcyc and 10^9 cycles as Gcyc. The schemes are deterministic given the DRBG seed, so run-to-run variation is purely algorithmic — the rejection-sampling path each instance takes — not measurement noise. We characterized it over N=20 instances: key generation is deterministic to within 0.05 %, signing varies within about ±1.0 %, and verification within about ±1.6 % (the challenge changes how many full canonical-form evaluations run). The final tables therefore report the median of N=10 instances at categories 1 and 3 — a few long-running reference and floor items captured 4–5 instances, well within the band above — and the median of N=5 (LESS-548-137) or N=3 (LESS-548-345) instances at category 5, whose hours-scale per-signature cost caps the instance count. Key generation, deterministic to within 0.05 % in the resident and in-place configurations, is reported as its median over the same runs; the streaming floor is the one exception — its rejection-sampled regeneration makes even key generation vary by a few percent, so it too is reported as a median of N. Toolchain: `arm-none-eabi-gcc` 10.3.1, `-O3` for every configuration (per-configuration flags in the artifact); we did not sweep compilers, so the codegen-dependent observations of Section 8.2 are for this toolchain. So that every configuration builds and runs at this `-O3` setting — including the reference baselines we benchmark (ref-sub, ref-norm) — all builds use the alignment-safe byte-transpose of Section 8.2, which is byte-identical to the official KATs and alters only how the 8-byte transpose loads are emitted (its cycle effect falls within the variation band above).

Memory. Our primary metric is **peak stack**, measured on target by the mupq stack-watermarking method; this is the value comparable with pqm4 tables. We additionally report **total peak RAM** = peak stack + `.bss` + `.data` (all configurations are heap-free), which is the number that determines whether an operation fits a device class. Three accounting notes, stated here once and applied throughout: (i) on the reference, 98.9 % of `.bss` ($2 \times 15{,}876$ B at category 1) consists of two *scheme-intrinsic* canonical-form working

buffers — they are not harness overhead and we charge them to the scheme; (ii) mem-max keeps a 29.4 KB static workspace (a 16 KB `less_shared_work` array in `.data` plus smaller buffers) and stages the matrix T in the caller-provided public-key buffer — its headline sub-KB stack therefore must be read together with its total-RAM column; caller-provided key and signature buffers are charged to the application, as in pqm4, so for key generation — which writes the public key — the floor's working set is its total RAM *plus* the public-key buffer; (iii) as a cross-check we also run the structurally identical x86 build under `valgrind massif -stacks=yes` (labeled *C-Keccak, x86-64*); absolute x86 values run higher because 64-bit call frames are wider, e.g., 1,920 B versus the 1,156 B on-target watermark for mem-max signing. That 66 % gap is a *frame-dominated* worst case — mem-max's sub-kilobyte stack is almost all call frame. The category-5 and LESS-400 reference watermarks we cross-check this way (Table 8, ‡) are the opposite regime: dominated by the resident $\mathbb{F}_{127}$ generator matrices, whose byte size is architecture-independent (a $k \times n$ matrix over $\mathbb{F}_{127}$ is kn bytes on x86-64 and on the M4 alike), so the frame inflation is a small *additive* term on a hundreds-of-kilobyte base, not a multiplicative factor. We therefore read the ‡ stacks as tight over-estimates and use them only for trend confirmation; the category-5 wall does not rest on them alone — it is set independently by the analytic budget of Section 4.3, where the resident matrix plus the T retained monomials already exceed the free SRAM.

3.4 Keccak Policy

All Cortex-M4 measurements link the same platform-standard assembly Keccak (pqm4) [Ado24, KRSS19], so that every delta between configurations is attributable to our techniques. The as-submitted reference with its bundled C Keccak is reported once for context (ref-sub). All C-level code is plain, intrinsic-free C, verified byte-identical against the NIST KATs on x86-64; KAT validity transfers across Keccak backends (Section 3.2). Empirically the normalization matters less for the reference than folklore suggests — ref-sub signs only 1.8–3.3 % slower than ref-norm across the measured sets — but it matters for attribution in the optimized configurations, where hashing is a much larger relative share.

3.5 Balanced Adoption Criteria (B1/B2)

The balanced configuration is defined by two rules, fixed before the final optimization campaign and applied to every candidate change:

B1 (Pareto-monotone adoption). Balanced adopts a change only if Δcycles < 0 *and* Δpeak RAM ≤ 0 (up to +1 KB tolerated for alignment and measurement noise). Peak RAM is checked per operation (keygen/sign/verify) via the on-target watermark and the x86-massif cross-check; the maximum over the three operations is the representative value.

B2 (Attribution of RAM-positive techniques). A speed technique that increases RAM is assigned to the *speed* configuration, not to balanced. Balanced's coordinate stays fixed at the knee; it may move only along the cycle axis.

The rationale is that B1/B2 *are* the claim: balanced asserts "the maximum speed achievable at the knee's RAM." If RAM may creep, the knee moves and the frontier claim degrades into a soft trade-off narrative. Memory-neutral techniques (register-level changes such as lazy reduction, Section 5.2) pass B1 trivially. Double-pass signing adds a second expansion pass, so it never satisfies B1's Δcycles < 0 clause: at categories 1 and 3, where a single-pass baseline exists, B1 correctly rejects it; at category 5 no single-pass configuration fits at all, so it is adopted as a *feasibility enabler* outside B1 — a documented exception, marked explicitly in the coverage table. B1's cycle clause was evaluated per adopted change

at the exploration clock; one kernel group whose sign-time effect flips sign between clocks is retained and documented as the only other deviation (Section 6.4). We use B1/B2 as a rigor device for the design-space claims of Section 6, not as a freestanding contribution.

4 Memory-Direction Techniques

The memory footprint of the reference implementation is concentrated in a handful of large objects: the expanded generator matrices ($k \times n$ over $\mathbb{F}_{127}$, 31–150 KB each), the per-round ephemeral monomials retained between commitment and response (T entries, up to 189 KB at LESS-548-345), and the canonical-form working buffers. The techniques below remove or shrink each of these. They come in two strengths: the *recomputation family*, which defines mem-max and accepts extra PRNG work to keep almost nothing resident; and the *in-place family*, which defines the balanced configurations and restructures computation so that one resident matrix suffices. All of them preserve byte-identical outputs (gate G1/G2).

4.1 Streaming RREF with Row-Block Regeneration

mem-max never materializes the expanded matrix at all. The matrix is a deterministic function of a seed via rejection-sampled SHAKE output; since the rejection stream is strictly sequential, any element can be recomputed by re-expanding from the seed. The streaming RREF therefore keeps only the current pivot information and regenerates matrix content on demand, trading PRNG work for memory.

Naïvely this trade is prohibitive: the original streaming formulation re-expanded the stream once per output row, i.e., k full expansions per elimination pass, and the early mem-max signed an order of magnitude slower than the reference. Two observations repair it. First, regeneration cost amortizes over *row blocks*: a batch of 8 output rows can be served by a single expansion pass (Algorithm 1), cutting expansions by 8×, with the per-batch state held in a 1,008 B buffer overlaid (by `union`) on an existing workspace — $\Delta\text{RAM} = 0$ by construction. Second, the inner update is the same arithmetic as everywhere else in this paper and benefits from the same SWAR treatment (Section 5.2): the masked source row is built once per row and applied with two-lane packed multiply-accumulate, whose intermediate stays below 2^{14} ($\leq$ 16,256: a folded lane $\leq$ 254 plus the bias 16,002), keeping the packed fold exact. Key generation uses the same restructuring with a block of 4. The combined effect is the single largest result of the memory direction: at category 1 mem-max signs at only 1.2–1.5× the reference time (Table 5) with a measured stack watermark near one kilobyte — and its key generation, which is elimination-dominated, is on par with or faster than the reference (at LESS-400-102, 37 % faster; at category 5, about 28 % faster). The recomputation cost scales super-linearly with the matrix dimensions, so at the highest-round sets signing becomes impractical (hours); there mem-max serves as a memory/correctness floor rather than a speed point (Section 6.1).

4.2 On-the-Fly GGM Tree and Seed-Cache Verification

The reference materializes the full GGM seed tree ($2T-1$ nodes) to derive per-round seeds and to publish the paths for unopened rounds. mem-max stores only the root: leaf seeds are derived on the fly by walking the path, and the published nodes for the signature are recomputed when needed (with an offset correction for the truncated, non-power-of-two tree). Verification, which receives published nodes rather than the root, uses a small cache of the published seeds so each is expanded once. This removes the tree buffer from both sign and verify at the cost of $O(\log T)$ extra SHAKE calls per access — which the row-block lesson above already taught us is the right currency to spend.

Algorithm 1 Streaming RREF with row-block regeneration (the mem-max floor). The expanded matrix is never stored; a block of b rows ($b = 8$ for elimination, 4 for key generation) is served by one rejection-sampled SHAKE expansion, held in a 1,008 B buffer union-overlaid on existing workspace ($\Delta\text{RAM} = 0$). The inner update is the packed SWAR MAC of Algorithm 3.

Require: seed sd; pivots established so far
for each row block $[i, i{+}b)$ of the matrix **do**
  $Blk \leftarrow \textsc{ExpandRows}(sd, i, b)$ ▷ one SHAKE pass regenerates b rows into the shared buffer
  for all established pivots (c, P_c) **do**
    for all rows $R \in Blk$ **do**
      $\textsc{RowUpdate}(P_c, R, R[c])$ ▷ Algorithm 3
    end for
  end for
  reduce Blk to its own new pivots; **discard** Blk
end for

Algorithm 2 Double-pass signing (the category-5 enabler). Single-pass signing retains all T ephemeral monomials from commitment to response (189 KB at LESS-548-345); the two passes below regenerate them instead, at the cost of one extra expansion over the challenged rounds, removing the entire T-sized array.

Pass 1 — commit.
for $i \leftarrow 1$ **to** T **do**
  $e_i \leftarrow \textsc{Ephemeral}(leaf_i)$ ▷ regenerate from the leaf seed
  $c_i \leftarrow \textsc{Commit}(e_i)$; absorb c_i into the hash; **discard** e_i
end for
$ch \leftarrow \textsc{Challenge}$(running hash) ▷ fixed weight w
Pass 2 — respond.
for all challenged rounds $i \in ch$ **do**
  $e_i \leftarrow \textsc{Ephemeral}(leaf_i)$ ▷ regenerate only what the response needs
  emit the response from e_i and the secret monomial
end for

4.3 Double-Pass Signing: The Category-5 Enabler

The structural reason the reference (and our speed configurations) cannot sign category 5 on a 640 KB part is not one big matrix but the *retention* requirement: all T ephemeral monomials must survive from the commitment phase to the response phase, 189 KB at LESS-548-345 — on top of public keys up to 197 KB (LESS-548-137) and the resident working matrices. The budget is unforgiving: roughly 493 KB remains after `.bss`, and single-pass signing overruns it, as we confirmed by the fault on the board.

Double-pass signing removes the retention (Algorithm 2): pass 1 generates each ephemeral from its leaf seed, computes the commitment, absorbs it into the running hash, and *discards* the ephemeral; after the challenge is derived, pass 2 regenerates exactly the ephemerals needed for responses. The cost is one extra expansion pass over the challenged rounds; the saving is the entire T-sized array. Balanced enables this only at category 5 (flagged `DP` in our tables), where it is the difference between "does not load" and "runs": it is one of the two memory-lean policies that clear the category-5 memory wall — the other being the streaming floor of Section 4.1 — and the one whose signing we measure on target across all seven sets (Section 6.1). At categories 1 and 3 the single-pass variant is Pareto-better and B1 keeps it.

4.4 In-Place Computation: The Balanced Memory Policy

The balanced configurations keep exactly one resident generator matrix and make every consumer work inside it:

- *In-place monomial application*: the product $G\,Q$ is applied by walking the permutation cycles of Q and scaling in place, eliminating the second matrix buffer of the reference formulation.
- *In-place non-IS packing*: the non-identity-set columns needed by the canonical form are packed into the front of G itself rather than into a separate buffer.
- *Scratch absorption*: the sorting routines of the canonical form need two 16 KB scratch areas (row sort and column sort); both are placed in regions of G that are provably idle at the call sites, and a `union`-typed view ensures the overlay is explicit in the type system rather than a cast. The same overlay style hosts the row-block buffer of Section 4.1.

The result is the knee of the design space: peak stack falls by 41–59 % across categories 1 and 3 (at LESS-252-45: 67,816 B versus 166,852 B on-target; category 5 reduces further, but against an x86-massif reference and so excluded from this range) while the recomputation overhead this policy implies is, after the speed layers of Section 5, only 8–9 % of signing time relative to the resident-matrix speed configuration. We emphasize what the residual is: the remaining peak is essentially the resident matrix itself plus minimal call frames. The knee is a structural floor of the in-place design, not an artifact of unfinished optimization — pushing below it requires giving up the resident matrix, which is exactly the mem-max regime.

4.5 Stack Discipline

Two unglamorous practices matter at the sub-kilobyte scale of mem-max. First, watermark-driven frame audits: every candidate change is re-measured for stack, not only for cycles (criterion B1), because optimizers readily trade stack for speed behind the implementer's back. Second, call-tree flattening on the hot path (selective `noinline` placement to bound worst-case frame accumulation). The measured floor at category 1 is 436–1,156 B across operations and parameter sets — the watermark of the deepest call chain (SHAKE absorb under the streaming regeneration), at which point the stack is no longer the binding constraint: the 29.4 KB static workspace and the key buffers are (Section 3.3).

4.6 An Assembly Arithmetic Core for the Floor

The floor's arithmetic is portable C by default: this is the variant we validate on the host KAT and report across all parameter sets, and it is what keeps mem-max fully portable. At the representative LESS-252-45 the arithmetic admits an inline-assembly core that holds the *same* stack watermark while cutting signing from 1.47× to 1.33× the reference (and key generation by 10.0 %), measured at 20 MHz. Three ingredients make it stack-neutral and fast. The core is written as GCC inline assembly rather than called routines, so it shares the caller's frame and adds no AAPCS spill — stack-neutrality by construction, confirmed by byte-identical paired watermarks; the conditional subtraction becomes a branchless `USUB16`/`SEL` packed select (five instructions and one constant, from seven and three); and the per-column inner product of the matrix regeneration is a `SMLAD` dot product. That last point is the converse of a negative result: `SMLAD`, which *regresses* the lane-independent speed-direction RREF (Section 8.1), is exactly the right instruction here, where the regeneration genuinely computes dot products — the same instruction, opposite verdict, decided by the loop's dependence structure. The core is validated identically to the

Table 3: What we inherit versus what is new on the Cortex-M4. The algorithm-level optimizations of the round-2 scheme are reused unmodified; our speed contribution is their realization on a core with no vector ISA. The memory-direction techniques (Section 4) and the evaluation methodology (Section 3.5) have no counterpart in the desktop-targeted prior work [BEP+25].

Component	Inherited (v2.0 / [BEP+25])	This work (Cortex-M4)
RREF elimination	pivot reuse, constant-time bound	rank-2 kernel; lazy reduction
$\mathbb{F}_{127}$ arithmetic	scalar fold reduction	two-lane SWAR; lazy reduction (bias trick); carry-free detector
Canonical form	early abort, blinding, histogram sort	transpose-grid / histogram / scatter–gather kernels
Verification, challenge sampling	variable-time RREF, Batcher network	inherited unmodified (§7)
Instruction scheduling	—	TRM timing-rule audit (all kernels)

C floor (host KAT byte-identical, on-target cross-build byte comparison, roundtrip); we demonstrate it at LESS-252-45 and leave propagation to the other sets to the measurement campaign. We report this gain at the 20 MHz benchmark clock deliberately: at the development clock the same change appears far larger on key generation (−64.9 % vs. −10.0 %), an artifact of the floor being flash-bound there rather than a compute gain (Section 3.3).

5 Speed-Direction Techniques

Profiling the reference on the M4 shows signing dominated by Gaussian elimination (about two thirds of cycles), followed by the canonical form and hashing. We present our speed work as three layers — algorithmic restructuring, portable C SWAR arithmetic, and hand-scheduled assembly kernels — because that is how correctness and attribution were maintained: each layer is separately gated, the first two are validated directly by the x86 KAT, and the assembly layer is pinned to the C layer by the on-target cross-build gate (G2). The per-kernel contribution of the third layer is exhibited by the waterfall ablation in Section 6.4.

An attribution note up front: most *algorithm-level* optimizations of RREF and the canonical form in LESS v2.0 originate in the work of Beckwith et al. [BEP+25] — pivot reuse with a constant-time reuse bound, the variable-time verification RREF, canonical-form early aborts, histogram-based sorting, multiplicative blinding, and Batcher-network challenge sampling are all inherited by the round-2 reference we start from, and we use them unmodified. Our contribution in this section is what happens *below* that level on a vector-ISA-less, load-latency-bound core: the restructuring, arithmetic representation, and instruction scheduling that those algorithms admit on the Cortex-M4. Table 3 draws the line explicitly, so that no novelty claim here is mistaken for an algorithmic one.

5.1 Layer 1: Algorithmic Restructuring

Four platform-independent restructurings, each KAT-neutral:

- *Skip-left elimination.* In the RREF inner loop, columns left of the current pivot column are already zero in the pivot row — an invariant the reference notes in

comments; we restructure the loop to act on it, starting the row update at the pivot column rather than column 0 and removing a quarter of the update work at no cost.

- *Fused single reduction.* The row update $r \leftarrow (r + 128q - m \cdot p) \bmod q$ is computed with one fused reduction instead of separate reductions after multiply and subtract.
- *Linear monomial composition.* The per-round composition of the ephemeral with the secret monomial, written as an $O(n^2)$ matrix-style loop in the reference path we inherited, is a permutation composition and computes in $O(n)$.
- *Run-copy packing.* The columns surviving into the canonical-form input are determined by the pivot set; because LESS pivots are near-diagonal, the surviving columns form long contiguous runs. Packing copies whole runs word-wise (with a packed fold on the way) instead of walking a per-column cursor, halving packing time.

5.2 Layer 2: Portable Two-Lane SWAR over $\mathbb{F}_{127}$ with Lazy Reduction

The M4 has no vector unit, but $q = 127 = 2^7 - 1$ leaves headroom in a 32-bit word. We pack two $\mathbb{F}_{127}$ elements per word in alternating byte lanes (mask `0x00FF00FF`); a product of two reduced lanes is at most $126^2 = 15{,}876 < 2^{14}$ (and, with the lazy bias, the running intermediate stays $\leq 16{,}256$), so two lanes multiply independently inside one 32×32 multiply, and the Mersenne structure reduces each lane with a two-operation fold $(t \mapsto (t \,\&\, 127) + (t \gg 7))$, once more for the carry). All of this is plain C — no intrinsics — which is what lets gate G1 validate it directly on x86. Eight-element unrolling interleaves independent packed updates so that load-use latencies overlap with arithmetic of the neighboring lanes (the M4 is in-order; instruction placement *is* the scheduler).

Lazy reduction. The reference reduces every element to canonical form after every update. We instead let row-update results accumulate *folded but unnormalized* in $[0, 254]$ and reduce only at six consumption sites (pivot search, inversion/scaling of the pivot row, multiplier extraction, packing into the canonical-form input, key compression). The subtlety is the bias constant (Algorithm 3): the naïve non-negativity bias lets the fold's shifted term reach $(t \gg 7) = 128$, whose carry crosses the byte lane and corrupts the packing once inputs reach 254; the bias $B = 16{,}002 = 126 \cdot 127 \equiv 0 \pmod{127}$ caps the pre-fold intermediate at 16,256, so $(t \gg 7) \leq 127$ and the two-operation fold $t \mapsto (t \,\&\, 127) + (t \gg 7)$ stays exact up to 254 *while preserving the two-lane packing* — the elimination inner loop then runs with no per-element conditional subtraction at all. This is the single most valuable arithmetic change in the paper — toggling it off costs roughly 20 % of elimination time in a development-clock profile, reproducible via the artifact's `LESS_LAZY_RREF` flag — it is memory-neutral (B1-trivial), and it is what later frees the registers that make the rank-2 kernel allocatable (Section 5.3).

Supporting SWAR forms. The same representation accelerates the surrounding loops: single-multiplier packing for pivot-row scaling; row-major SWAR for generator-monomial multiplication and for the blinding right-multiplication; and word-wise zero counting for the canonical form using the carry-free detector $\sim(w + \texttt{0x7F7F7F7F}) \,\&\, \texttt{0x80808080}$, which is exact for lanes ≤ 126 — adopted after the KAT gate exposed the borrow-propagation false positive of the textbook detector (Section 3.2).

5.3 Layer 3: Hand-Scheduled Assembly Kernels

About 89 % of signing cycles in spd-asm execute inside the hand-written kernels below; everything data-dependent and branch-dominated (quicksort partitioning, rejection control flow) stays in C, where we measured hand-assembly to be unprofitable. The kernels:

Algorithm 3 Two-lane SWAR row update over $\mathbb{F}_{127}$ with lazy reduction. Two $\mathbb{F}_{127}$ elements share one 32-bit word in byte lanes 0 and 2 (input mask M = `0x00FF00FF`, values $\leq$126); after a multiply each lane occupies its own 16-bit half. Results stay folded-but-unnormalized in $[0, 254]$ and are reduced only at the six consumption sites of Section 5.2. Packed bias B = `0x3E823E82` ($16{,}002 = 126 \cdot 127 \equiv 0 \pmod{127}$ per lane).

function FOLD(t) ▷ Mersenne-reduce one lane; exact while $t \leq 16{,}383$
  return $(t \,\&\, 127) + (t \gg 7)$
end function

procedure ROWUPDATE(pivot row P, target row R, multiplier m)
  for each packed word j **do** ▷ two $\mathbb{F}_{127}$ lanes per word
    $x \leftarrow m \cdot P[j]$ ▷ one 32×32 multiply; lane products fill disjoint 16-bit halves, each $\leq 126^2$
    $t \leftarrow R[j] + \mathsf{B} - x$ ▷ invariant $t \leq 16{,}256$, so $t \gg 7 \leq 127$: no carry crosses the lane
    $R[j] \leftarrow \text{FOLD}(t \,\&$ `0xFFFF`$) \mid \big(\text{FOLD}(t \gg 16) \ll 16\big)$
  end for
end procedure

Rank-2 row elimination. The workhorse. The rank-1 AXPY (row $\leftarrow$ row $- m \cdot$ pivot) reloads the pivot row for every destination row; processing *two* destination rows per pass shares one load-and-unpack of the pivot row, bringing the kernel to 37 instructions per 8 packed updates (4.625 per packed update, against rank-1's 5.5; with the dead-mask removal below applied to both kernels, the like-for-like comparison is rank-1's 5.0 against rank-2's 4.625). The register budget only closes because lazy reduction freed the conditional-subtract constants and a dead-mask removal freed one more register (a packing-stage byte mask that is a no-op on fold outputs ≤ 254 but that the compiler could not prove dead — two instructions per four elements across all kernels). A rank-3 variant will not allocate without spills, so we take 4.625 instructions per packed update as the practical floor of this kernel family on ARMv7E-M.

Transpose grid. Column sorting transposes the matrix twice. The inherited C path fast-paths only a 64×64 block of the 126×126 matrix — nearly three quarters of elements fell to a scalar residue loop — so the kernel transposes the full grid in one call as 4×4 byte tiles steered by `PKHBT`/`PKHTB` and the barrel shifter. Building this path at `-O3` is what motivated the alignment-safe byte-transpose of Section 8.2.

Histogram. The counting-sort increment address depends on the just-loaded value — a textbook address-generation stall, removed by software-pipelining the value loads one iteration ahead.

Row accumulate and row scalings. Byte-row sums via `USADA8` (four-byte horizontal accumulate, one instruction); the `row_mul` family as `SMULBB`/`SMULTT` lane products re-packed with `PKHBT`, where hand scheduling recovered 4–7 % even over the already two-lane C SWAR.

Monomial scatter / blinding gather. Permutation application with each index extracted ahead of its use, again avoiding the address-generation stall (Section 5.4).

Keccak. The platform-standard pqm4 assembly permutation [Ado24] — part of the baseline by policy (Section 3.4), and deliberately *not* a bar in our waterfall.

5.4 A TRM-Driven Timing Audit

After the kernels were functionally optimal, one further pass produced consistent gains with zero algorithmic content: auditing every kernel against the Cortex-M4 TRM's documented load/store timing rules [Arm15]. The rules that matter: (i) a run of consecutive loads pipelines — ℓ back-to-back `LDR`s cost ℓ+1 cycles — but an ALU instruction splitting the run forfeits the pipelining, so loads are grouped rather than interleaved naïvely; (ii) an address computed by the immediately preceding instruction stalls address generation — so indices are extracted early (scatter/gather kernels); (iii) `LDM` does not pair with neighboring loads — grouped `LDR`s are never worse. The audit found at least one violation in *every* kernel — including a pivot-row load isolated from its chain in the rank-2 kernel and an address-dependent table load in the histogram — and fixing them yielded 3–5 % per kernel with no regressions. We highlight this as a methodology result: on an in-order core, the TRM is a checklist, and auditing against it is cheaper and more systematic than profile-guided trial-and-error. The audit generalizes to any ARMv7-M kernel work, independent of LESS.

6 Results

6.1 Coverage

Table 4 is a result, not bookkeeping: which configuration runs which parameter set on the device is the characterization the scheme has been missing. A *single hard wall* structures it — memory, at category 5. The resident-matrix configurations (the reference and the speed ceiling) cannot sign or verify there: after `.bss`, about 493 KB of SRAM remains, against a single-pass requirement we break down to 550–567 KB (the public key, the retained per-round monomials, and the working matrices), and we confirmed the resulting fault on the board. Key generation, which touches one matrix and none of the retained per-round state, still fits: both resident-matrix configurations remain keygen-capable at category 5 (cells kg), and the speed ceiling retains its keygen advantage there (about 5.2× the reference, Table 5). Two memory-lean policies clear the wall, both byte-identical to the reference: the *in-place* balanced knee, which we measure on target at every parameter set (with double-pass signing at category 5), and the *streaming* floor (mem-max), whose correctness we verify against the host KAT at every set and whose key generation we measure on-target at every set. The streaming floor is not a slow fallback at the scales where it is practical: at category 1 it signs at only 1.2–1.5× the reference and generates keys on par with (within the run-to-run band) or below reference cost (Section 6.2). Its recomputation cost scales super-linearly, however; by the highest-round sets (LESS-400-220 and category 5) a single signature runs for hours, and since the regeneration is flash-bound the faster development clock does not help. There mem-max is a correctness-and-memory existence proof (host-KAT-verified across all seven sets, with key generation measured on-target and signing reported as an hours-scale bound — cells marked M) rather than a usable speed point. Category 5 therefore has exactly one configuration that both fits the device and signs at practical speed: bal-asm-DP. The coverage asymmetry is a property of the scheme, not a limitation of this work: signing LESS at category 5 with resident matrices needs more SRAM than the largest mainstream Cortex-M4 provides, and the only way onto the device there is to stop keeping the matrices resident.

For completeness we also examined an intermediate variant that trades stack for static buffers (7.7 KB stack, +58 KB `.bss`). Its total RAM is only ≈3 % below balanced at LESS-252-45; it did not complete signing within the campaign's measurement window (its recomputation lineage signs at roughly twice the reference time); and it cannot load category 5 (322.9 KB of `.bss` leaves 317 KB — less than its 379 KB buffer requirement). It is dominated in the sense that matters, and we omit it from the matrix (one measured

Table 4: Coverage matrix. ✓ = measured on target at 20 MHz (cycles; memory per Section 6.5); DP = with double-pass signing; kg = keygen measured on target, sign/verify exceed SRAM (budget breakdown in text); M = keygen measured on target, sign/verify host-KAT-verified with on-target signing impractical (hours), reported as a time bound; abl = measured, reported in the ablation table; — = out of scope.

	Category 1			Category 3		Category 5	
Config	252-45	252-68	252-192	400-102	400-220	548-137	548-345
ref-sub/ref-norm	✓	✓	✓	✓	✓	kg	kg
mem-max	✓	✓	✓	✓	M	M	M
bal-asm	✓	✓	✓	✓	✓	✓ DP	✓ DP
spd-asm	✓	✓	✓	✓	✓	kg	kg
bal-c/spd-c	abl	abl	abl	abl	abl	—	—

point in the artifact).

6.2 Performance

Table 5 reports the headline numbers. The speed ceiling spd-asm signs 4.5–4.7× faster than ref-norm across every measured parameter set (−77.7 to −78.7 %), verifies 4.0–4.5× faster, and generates keys 5.0–5.6× faster. The balanced knee bal-asm gives up only 8–9 % of that signing speed (Section 6.7) while halving memory: 4.1–4.3× faster signing than the reference. Normalizing the Keccak backend costs the reference itself little — ref-sub signs 1.8–3.3 % slower than ref-norm— so the out-of-the-box gap is dominated by our techniques, not by the hash.

Three observations beyond the headline. First, the memory floor is no longer slow: mem-max signs at +24 to +47 % over the reference (category-dependent) — not the multiples one expects from a streaming-recomputation design — and its elimination-dominated key generation matches the reference at category 1, *beats it by* 37–38 % at category 3, and beats it by about 28 % at category 5 — where key generation is the only operation the reference itself can run on the device — because the row-block SWAR arithmetic outweighs the regeneration overhead. An inline-assembly arithmetic core cuts the LESS-252-45 figure further, to +33.3 % at the same one-kilobyte stack (Section 4.6). Second, category 5 runs: the balanced configuration with double-pass signing completes LESS-548-137 in 39.6 Gcyc and LESS-548-345 in 99.4 Gcyc — 33 and 83 minutes of wall time at the 20 MHz benchmark clock (at the 120 MHz clock, dividing by six gives 5.5 and 14 minutes, though the wait-state inflation of Appendix B makes the realized times somewhat longer) — while generating keys 5.1× faster than the reference's keygen-only capability at the same sets. Whether such signing latencies are usable is application-dependent; the point of this row is that the characterization exists at all, where previously the answer was a fault. Third, the cost of the balanced memory policy is modest and uniform: +8 to +9 % signing time against spd-asm for half the stack.

6.3 Layered Ablation

Table 6 separates the portable layers from the assembly layer, on both memory policies. The portable layers (L1+L2) account for the bulk: −68 to −69 % signing versus ref-norm, *achieved in plain C with the same hash backend* — this is the part that transfers to any platform. The assembly layer adds a further −27 to −31 % on top of its C base, consistently across parameter sets and memory policies.

These gains are the techniques', not an under-tuned baseline's: the reference we compare against is already built at `-O3`, and rebuilding it at `-Ofast` — the most aggressive

Table 5: Cycle counts at 20 MHz (Mcyc, median of N runs: $N = 10$ at categories 1–3 — a few long-running reference and floor entries captured $N = 3$–5 — and $N = 5$ or 3 at category 5 by cost; see Section 3.3 for variance). Δsign is relative to ref-norm of the same parameter set. ref-sub is shown once for context (remaining sets: +1.8–2.1 % sign, artifact). DP = double-pass signing. OOM = exceeds SRAM (Section 6.1). [a]inline-assembly arithmetic core, demonstrated at LESS-252-45 (Section 4.6); other sets use the portable C floor. [b]keygen measured with a keygen-only harness (median of five runs); sign/verify are not run on target in these configurations (Section 6.1). The reference category-5 keygen entries are single runs (key generation is deterministic to 0.05 %).

Set	Config	keygen	sign	verify	Δsign
252-45	ref-sub	681.9	4,037.0	2,616.4	+3.3 %
	ref-norm	680.4	3,908.2	2,558.6	—
	mem-max	678.6	5,732.3	4,267.9	+46.7 %
	mem-max (asm)[a]	610.9	5,208.3	3,948.9	+33.3 %
	bal-asm	139.0	942.5	656.4	−75.9 %
	spd-asm	128.5	871.8	614.3	−77.7 %
252-68	ref-norm	292.5	5,928.1	3,867.4	—
	mem-max	292.8	8,355.9	6,501.2	+41.0 %
	bal-asm	59.6	1,435.0	991.0	−75.8 %
	spd-asm	56.0	1,320.4	946.9	−77.7 %
252-192	ref-norm	98.6	16,705.5	10,726.6	—
	mem-max	96.7	20,713.4	19,046.9	+24.0 %
	bal-asm	19.9	4,040.4	2,980.9	−75.8 %
	spd-asm	19.7	3,709.0	2,706.9	−77.8 %
400-102	ref-norm	1,163.3	29,090.8	19,765.1	—
	mem-max	737.2	41,037.9	25,284.7	+41.1 %
	bal-asm	218.8	6,711.4	4,706.9	−76.9 %
	spd-asm	208.6	6,205.9	4,414.4	−78.7 %
400-220	ref-norm	390.9	62,722.4	42,291.3	—
	mem-max	242.8[b]	host-KAT verified (M)		—
	bal-asm	72.9	14,474.9	10,621.1	−76.9 %
	spd-asm	72.6	13,370.1	9,788.8	−78.7 %
548-137	ref-norm	2,713.2	OOM		—
	mem-max	1,935.8[b]	host-KAT verified (M)		—
	bal-asm DP	536.6	39,604.8	14,735.5	—
	spd-asm	525.7[b]	OOM		—
548-345	ref-norm	918.2	OOM		—
	mem-max	662.6[b]	host-KAT verified (M)		—
	bal-asm DP	178.9	99,356.9	39,042.7	—
	spd-asm	178.4[b]	OOM		—

Table 6: Layered ablation, 20 MHz (Mcyc), all three operations. ΔC = vs ref-norm (same operation); Δasm = vs the corresponding C layer. The layer structure is near-identical across operations, with key generation showing the largest reductions (its elimination benefits most from the SWAR arithmetic). bal-c's own ΔC, omitted for width, is −66.6 to −67.4 % (sign), −62.6 to −66.9 % (verify), and −70.6 to −72.2 % (keygen).

			speed policy		balanced policy	
Set	Op	ref-norm	spd-c (ΔC)	spd-asm (Δasm)	bal-c	bal-asm (Δasm)
252-45	keygen	680.4	190.7 (−72.0 %)	128.5 (−32.6 %)	200.2	139.0 (−30.6 %)
	sign	3,908.2	1,232.8 (−68.5 %)	871.8 (−29.3 %)	1,304.2	942.5 (−27.7 %)
	verify	2,558.6	848.6 (−66.8 %)	614.3 (−27.6 %)	876.3	656.4 (−25.1 %)
252-68	keygen	292.5	82.6 (−71.8 %)	56.0 (−32.2 %)	85.8	59.6 (−30.5 %)
	sign	5,928.1	1,868.5 (−68.5 %)	1,320.4 (−29.3 %)	1,970.7	1,435.0 (−27.2 %)
	verify	3,867.4	1,286.3 (−66.7 %)	946.9 (−26.4 %)	1,344.8	991.0 (−26.3 %)
252-192	keygen	98.6	28.6 (−71.0 %)	19.7 (−31.1 %)	28.6	19.9 (−30.4 %)
	sign	16,705.5	5,255.3 (−68.5 %)	3,709.0 (−29.4 %)	5,559.5	4,040.4 (−27.3 %)
	verify	10,726.6	3,751.8 (−65.0 %)	2,706.9 (−27.9 %)	4,009.1	2,980.9 (−25.6 %)
400-102	keygen	1,163.3	316.8 (−72.8 %)	208.6 (−34.2 %)	325.8	218.8 (−32.8 %)
	sign	29,090.8	8,992.4 (−69.1 %)	6,205.9 (−31.0 %)	9,478.3	6,711.4 (−29.2 %)
	verify	19,765.1	6,218.2 (−68.5 %)	4,414.4 (−29.0 %)	6,535.5	4,706.9 (−28.0 %)
400-220	keygen	390.9	108.7 (−72.2 %)	72.6 (−33.2 %)	108.6	72.9 (−32.9 %)
	sign	62,722.4	19,500.3 (−68.9 %)	13,370.1 (−31.4 %)	20,472.4	14,474.9 (−29.3 %)
	verify	42,291.3	13,959.9 (−67.0 %)	9,788.8 (−29.9 %)	14,698.1	10,621.1 (−27.7 %)

standard level — changes its deterministic key-generation cycles by under 0.01 % and its signing by under 0.1 % at LESS-252-45, inside the run-to-run band. What separates the reference from the frontier is the algorithmic restructuring of Section 5.1 and the SWAR representation of Section 5.2, which no optimizer flag reaches.

6.4 Per-Kernel Waterfall

Table 7 attributes the assembly layer kernel by kernel, starting from spd-c (the assembly Keccak is already in the base by policy — it is deliberately not a bar). The elimination kernels dominate, as the profile predicts: −25.7 % of signing in one step. The canonical-form kernels contribute steadily smaller slices, in profile order, with one informative exception. The final scatter/gather kernels are a small *regression* on signing at the zero-wait-state benchmark clock (+24.8 Mcyc, the last row) while still improving verification (−7.8 Mcyc); at the development clock, where instruction fetch competes with flash wait states, the same kernels are sign-neutral. We keep them in the full assembly builds for the verification gain and for build simplicity, and report the effect rather than tuning the kernel set per operation — adoption was gated at the exploration clock, where this group is sign-neutral, and the benchmark-clock regression that emerged on re-measurement is the one kernel-level deviation from B1's cycle clause, documented here (Section 3.5). The smallest stage deltas are comparable to the DRBG instance band and should be read as indicative rather than exact.

6.5 Memory

Table 8 reports peak stack. At LESS-252-45, measured on-target watermarks: the balanced knee needs 67,816 B at its worst operation versus 166,852 B for the reference (−59.4 %), and the memory floor needs 1,156 B — −99.3 %, more than two orders of magnitude — with category-1 watermarks of 436–1,156 B across all operations and parameter sets. Two honesty notes. First, total RAM: adding the scheme static (Section 3.3), the floor's

Table 7: Cumulative kernel waterfall, LESS-252-45, 20 MHz (Mcyc), from a single controlled per-kernel run (so absolute endpoints differ from the median-of-N values in Tables 5 and 6 by <3 %; the per-kernel Δs are the object of interest). Each row enables one kernel group on top of the previous row; Δ = vs the previous row. The final group (scatter/gather) regresses signing slightly at this zero-wait-state clock while improving verification — a clock-dependent effect discussed in the text.

Stage	sign	Δsign	verify
base: spd-c (portable C + asm Keccak)	1,250.6	—	826.6
+ RREF elimination (rank-1/rank-2)	929.3	−25.7 %	675.8
+ transpose grid	908.7	−2.2 %	661.0
+ histogram	891.2	−1.9 %	646.8
+ row accumulate	880.0	−1.3 %	632.3
+ row scalings	870.3	−1.1 %	622.6
+ scatter/gather (full spd-asm)	895.1	+2.9 %	614.8

Table 8: Peak memory. Top: on-target stack watermarks at LESS-252-45 (bytes; representative value = max over operations, criterion B1). Bottom: bal-asm's per-operation stack across parameter sets, with the reference peak; Δ is the peak-to-peak reduction (bal-asm signing, its peak, versus the reference's verify peak). Unmarked category-1 rows and the balanced LESS-400-102/220 watermarks are on-target (LESS-252-68 and LESS-252-192 are single runs; input-dependent variation ∼100 B). Markers: ‡ x86-massif cross-check on the structurally identical host build — the reference's 416–836 KB stack plus the pqm4 test buffers overruns the 640 KB part (and category 5 does not run there at all), so stack uses the host cross-check while the on-target cycle counts of Table 5 need no such headroom; § on-target double-pass watermark; ¶ mem-max's dimension-independent streaming-SHAKE call-chain watermark (regeneration buffers are static, not on the stack, Section 4.1) — category-1 mem-max measures 436–1,156 B, so we report ∼1 KB at categories 3 and 5, where signing is impractical to watermark on target (hours).

252-45	keygen	sign	verify	max	static	total RAM
ref-norm	82,296	147,356	166,852	166,852	32,116	≈199.0 KB
spd-asm	82,476	147,852	167,580	167,580	32,116	≈199.7 KB
bal-asm	34,556	67,816	55,064	**67,816**	32,120	≈100.0 KB
mem-max	628	1,156	500	**1,156**	29,403	≈30.6 KB

		bal-asm				
Set	ref-norm	keygen	sign	verify	Δ	mem-max
252-45	166,852	34,556	67,816	55,064	−59.4 %	1,156
252-68	167,988	34,428	74,740	56,316	−55.5 %	920
252-192	191,532	34,340	112,124	62,516	−41.5 %	964
400-102	415,992‡	83,820	176,996	134,404	−57.5 %	∼1 KB¶
400-220	433,848‡	83,624	232,828	143,140	−46.3 %	∼1 KB¶
548-137	776,328‡	155,156§	251,748§	247,564§	−67.6 %	∼1 KB¶
548-345	835,992‡	155,004§	271,996§	267,948§	−67.5 %	∼1 KB¶

working set is ≈30.6 KB — its sub-kilobyte stack sits atop a 29.4 KB static workspace — the knee's ≈100 KB, and the reference's ≈199 KB — the ordering is unchanged, but sub-KB stack must not be mistaken for sub-KB RAM. Second, the speed configurations inherit reference-level stack *by design* (resident matrices); their column is there to show what the ceiling costs.

6.6 Code Size

Peak stack is the binding footprint on this platform, but flash matters for portability to smaller-flash parts, and our layers — eight-wide unrolling, the hand-written kernels, and the streaming regeneration code — trade code for cycles. Appendix A.4 (Table 11) reports firmware `.text` at LESS-252-45 (scheme, kernels, platform Keccak, mupq harness, and `libopencm3` runtime; the non-scheme parts are constant across the assembly-Keccak configurations, so the deltas are attributable to our techniques). The speed layers add 13–20 % of code, the in-place balanced policy 36–44 %, and the streaming floor 47–48 %. In absolute terms the growth is immaterial on this device: the largest configuration occupies 49.6 KB of the part's 2 MB flash (2.4 %), so unlike SRAM, code size is not a binding constraint here — we report it per pqm4 convention and for the smaller-flash parts to which the techniques may be carried.

6.7 The Design Space

Tables 5 and 8 place each configuration in a single speed–memory plane — signing time against peak stack, at LESS-252-45 — and its shape is the central empirical claim: the *reference is not on the frontier*. bal-asm strictly dominates ref-norm on both axes simultaneously (−75.9 % cycles at −59.4 % stack at LESS-252-45); spd-asm and mem-max then trade along the frontier in opposite directions — spd-asm buys the last 8–9 % of signing speed at reference-level memory, while mem-max buys a two-orders stack reduction at +24–47 % cycles. Where one should sit on this frontier is an application property; what this section establishes is that the frontier itself is far from where the reference suggested it was.

6.8 Device-Class View

As a characterization (not a deployment recommendation), the total-RAM column maps to MCU SRAM classes as follows at category 1, counting working RAM only and assuming long-term keys reside in flash: mem-max signs and verifies in a 64 KB-class part (≈30.6 KB; key generation, which writes the public key, additionally needs the pk buffer itself, Section 3.3); bal-asm fits a 128 KB-class part (≈100 KB); the reference and spd-asm need a 256 KB-class part (≈199 KB); and category 5 is reached only by the memory-lean policies — bal-asm-DP on a 640 KB-class part (the resident-matrix configurations exceed it), while mem-max's category-5 working set stays dominated by its scheme static rather than its ∼1 KB stack (Table 8).

6.9 Cortex-M4 Context

Post-quantum signatures have been benchmarked across the Cortex-M4, including on the Nucleo-L4R5ZI used here — pqm4's current default target and the board of the additional-signature campaign [KKPY24]. At the zero-wait-state benchmark clock, sign *cycle* counts are core-microarchitecture properties (Section 10.3), so LESS's cost reads against these schemes at the cycle level (Table 9); stack, unlike cycles, is board- and configuration-dependent, so we pin each figure's source. Even at its speed ceiling LESS signs one to two orders of magnitude slower than the lattice and multivariate schemes, and tens of times slower than the code-based CROSS — the price of its short signatures, and the reason its per-signature latency is the operative constraint. Verification shows the same gap or wider — Falcon's verification in particular is three orders cheaper; key generation is the exception — LESS's row-elimination key generation (≈128 Mcyc) is faster than UOV's and within about 2× of Falcon's, the schemes that likewise carry an expensive key-generation step, though it stays one to two orders above ML-DSA and MAYO. This

Table 9: Cortex-M4 context: NIST level-1 key generation, signing, and verification. Cycle counts are clock- and (at zero wait states) board-independent core-microarchitecture properties; stack is board- and configuration-dependent, so sources are pinned. [a]pqm4 [KRSS19] benchmarks on the Nucleo-L4R5ZI [KKPY24] (`m4f` builds, mean cycles; Falcon-512 is the FN-DSA provisional implementation). [b]CROSS speed-optimized R-SDP-1-fast [SGB+25] (Table 2); its stack-optimized variants of the same set trade speed for stack, reaching 7–12 KB at 30–38 Mcyc. [c][SGB+25] reports signing and verification only; CROSS key generation is variant-independent and untouched by those optimizations, so its figures are the v2.1 reference measurements of Hart et al. [HMK+26] (Tables 4–5).

	cycles (Mcyc)			stack (KB)		
Scheme (NIST level 1)	keygen	sign	verify	keygen	sign	verify
LESS-252-45, speed ceiling (this work)	128.5	871.8	614.3	82	148	168
UOV (`ov-Ip`)[a]	139.3	2.5	1.0	19	5	3
ML-DSA-44 (Dilithium2)[a]	1.4	3.9	1.4	38	45	9
MAYO$_1$[a]	9.6	16.9	11.0	179	251	253
CROSS (R-SDP-1-fast, speed)[b]	0.4[c]	14.5	8.6	8[c]	121	61
Falcon-512 (FN-DSA)[a]	67.7	22.5	0.4	14	42	3

scales the systems observation of Section 1: the >4× implementation-maturity swing narrows but does not close the gap.

7 Security Considerations

Nominal versus effective security. The parameter sets carry their NIST category labels (1/3/5) from the v2.0 specification, and we keep those labels for navigation. They should be read together with the second-round cryptanalysis: Budroni et al. [BEFN25] improved attacks on code equivalence and reduced the effective security of the LESS parameter sets by an estimated 12–24 bits depending on category — one of the factors NIST cited [Nat26]. The attack line has continued past the status report: an optimized codeword-search framework by Budroni and Esser [BE26] further improved these attacks, lowering the estimated bit security of the LESS parameter sets by up to an additional 15 bits over the previous best, and a single-codeword-matching approach to linear code equivalence has also appeared [BBO+26]. Nothing in this paper changes that picture in either direction: we implement the v2.0 scheme byte-identically, and our results are about its cost, not its hardness. Re-parameterized successors would inherit our techniques unchanged, since none of them depends on the specific (T, w, s) choices.

Memory optimizations and the security model. None of our memory techniques changes the public interface or the data an adversary sees. The streaming and on-the-fly configurations recompute values the reference already derives from public inputs — the generator matrix G_0 from the public seed carried in the key, and the GGM path nodes from the signature — rather than storing them, and double-pass signing regenerates ephemerals that the single-pass signer holds in memory. All configurations are byte-identical to the reference on the official known-answer tests (Section 3.2), so the public-key and signature distributions, and hence the security notion, are exactly the reference's; no new seed or intermediate is exposed.

Timing behavior. We claim no constant-time property for any configuration; what we can state precisely is a classification, and the fact that our optimizations preserve the reference's timing envelope:

- *Data-independent by construction*: the SWAR field arithmetic including lazy reduction (folds and bias addition are value-independent; no lane ever branches on data), the row-scaling, accumulate, transpose, scatter/gather, and Keccak kernels, and the GGM/Batcher sampling structure inherited from the reference [BEP+25].

- *Secret-adjacent but bounded*: the signing RREF inherits the reference's pivot-reuse scheme with its constant-time reuse bound ($P = 25$ at category 1), introduced and security-argued by Beckwith et al. [BEP+25]; we did not alter the bound or the pivot-search control flow.

- *Variable-time on blinded or public data*: the canonical-form sorting (histogram contents, quicksort partitioning) branches on data that the reference computes under multiplicative blinding, with the leakage argument given in [BEP+25]; verification additionally uses the reference's variable-time RREF, a documented design decision of the scheme's own optimization literature, which we inherit and cite as precedent rather than re-justify.

Two platform notes. First, on this core, SRAM accesses are uncached and single-cycle-deterministic, so value-indexed SRAM accesses (histogram counters) do not create the cache-timing channel they would on an application processor; flash is behind the ART cache, but our kernels execute from flash with data in SRAM. Second, our hand-written kernels replace like-for-like arithmetic and introduce no new secret-dependent branches or table lookups; the cross-build gate (G2) additionally pins their input–output behavior to the C semantics.

Fault attacks. LESS-specific fault analyses exist — ZKFault [MAKK24] and the forgery-oriented attack of [MKN+25] — and SHA-3-targeting fault key recovery has been demonstrated on pqm4-class targets including this scheme family [SJDN26]. Our implementations contain no fault countermeasures; hardening any of the configurations (in particular the seed-tree and double-pass paths, whose recomputation structure changes the fault surface) is out of scope for this work and flagged as such. We likewise implement no power- or electromagnetic-side-channel countermeasures; masking or shuffling any configuration is orthogonal to this characterization and left to future work.

8 Negative Results and Lessons

8.1 What Did Not Work

We implemented, gated, and rejected the techniques below. We report them because several are *standard advice* — two are flagship techniques of desktop-SIMD and sibling-scheme optimization work — and knowing that they invert on this core is as transferable as the positive results. Outcomes are adoption-gate measurements at the 120 MHz development clock during exploration (sign unless stated); each rejected change is preserved behind a default-off build flag in the artifact for reproduction.

Table-lookup scalar multiplication
(helps verify on AVX-512/NEON [BEP+25]) — +6 % sign: load latency loses to the single-cycle multiply plus fold.

Blocked elimination with `SMLAD` dot products
(the CROSS signature kernel on the M4 [SGB+25]) — +34 % RREF: serial accumulation chains stall; LESS's AXPY update is lane-independent and prefers wide interleaving.

Row padding 126 → 128
(word-aligns every row) — +1.8 % sign: the row-wise absorb needed to preserve the hash stream costs more than the unaligned penalty it removes.

Word-skip early-exit comparisons
(skipping equal prefixes in sorting) — +1–5 % CF: blinded rows diverge in the first bytes and prefixes are short.

16-wide unrolling
(fewer loop branches) — slower than 8-wide: residual-branch and code-size effects dominate.

Dual-row row scaling with a shared inverse
(the rank-2 idea applied to CF scalings) — no effect (< noise): the shared work is too small a fraction.

The pattern across these is consistent and worth stating as the lesson: *on this core, arithmetic is cheap and memory traffic is expensive — but only streaming traffic is affordable: latency-bound lookups, layout shuffling, and comparison shortcuts all lose to straight-line SWAR with deep independent interleaving.* The first two entries sharpen this into a portability warning: a technique's success on AVX-512/NEON (the table lookups above) or even on the *same core for a sibling scheme* (CROSS's `SMLAD` kernels, whose syndrome computations really are dot products) does not survive a change in the inner loop's dependence structure. The same instruction even reverses verdict within this paper — `SMLAD` regresses this lane-independent RREF yet is exactly the right tool for the floor's genuine dot-product regeneration (Section 4.6). Ablation against one's own baseline, under a correctness gate, is the only adjudicator we trust. One caveat, noted above: these adoption measurements were taken at the exploration clock. The two large rejections (+6 % and +34 %) sit far outside the clock-sensitivity band of Appendix B; the sub-2 % entries are exploration-clock calls, and no headline result rests on them.

8.2 An Alignment-Safe Transpose and a Memory Free Lunch

Our builds target `-O3`, and at that level the byte-transpose on the canonical-form path needs care: it moves 8-byte groups at a row stride of $n-k$, which is not a multiple of eight for categories 1 and 5 (strides 126 and 274; category 3's 200 is aligned). We therefore perform these 8-byte loads through `memcpy` rather than a `uint64_t` dereference. The two are byte-identical in output, but only the former is portable across optimization levels: at `-Os` and `-O2` the compiler emits unaligned-tolerant word loads, whereas at `-O3` it is free to lower an aligned-typed dereference to `LDRD`, which faults on an unaligned address on this core. Desktop and AArch64 hosts tolerate the misalignment in hardware, so the difference only shows up on the device (or under a host alignment sanitizer); we apply the alignment-safe form uniformly so that every configuration, including the reference baseline we benchmark, builds and runs at our `-O3` setting.

This one detail carries a system-level lesson worth stating. An earlier iteration of our code hit exactly this `-O3` fault and, having misattributed it to stack pressure on the 640 KB part, worked around it by forcing the working matrices into static storage. Reading the fault-status register settled the true cause — an unaligned usage fault at the offending load, with the memory-management and bus-fault bits clear and static use at only ≈32 KB — and switching to the alignment-safe transpose returned ≈170 KB of `.bss` to the stack in the speed configurations, which is what lets them reach reference-level footprint at maximum speed (Section 6). The lesson: on a constrained target an `-O3` HardFault reads as a memory problem, and it is tempting to pay for it in footprint; the fault-status register — not the memory map — is the right first witness.

9 Related Work

LESS optimization. The closest work is "LESS is Even More" by Beckwith et al. [BEP+25], which introduced most of the algorithm-level machinery the round-2 scheme ships with — pivot-reuse RREF with a constant-time bound, variable-time verification RREF, canonical-form early aborts and scaling reductions, histogram-based sorting, multiplicative blinding, Batcher-network sampling — and evaluated on AVX2, AVX-512, and NEON; it reports no microcontroller results. We deliberately position this paper as *complementary along the platform axis*: we inherit their algorithm-level optimizations unchanged (and cite their security arguments for blinding and variable-time verification in Section 7), and contribute what those algorithms need on a vector-ISA-less, in-order, load-latency-bound core — the two-lane SWAR representation with lazy reduction, the rank-2 elimination kernel, the TRM audit, and the entire memory direction, none of which appears in their work (a desktop target gives no reason to care). The contrast is occasionally sharp: their table-lookup scalar multiplication, profitable under `vpermi2b`/`vqtbx`, is a measured regression on the M4 (Section 8.1).

Code-based signatures on the Cortex-M4. For CROSS — a code-based (RSDP) scheme over the same field $\mathbb{F}_{127}$— two recent works target the M4: LightCROSS [HMK+26] reports large memory reductions (over 90 % across operations) with just-in-time hashing and DSP-assisted matrix kernels, and Schupp et al. [SGB+25] present paired minimal-stack and DSP-accelerated speed implementations — a two-extremes structure our floor/ceiling axis follows, extended here by an explicitly governed knee (B1/B2) and a full coverage matrix. Technique transfer between the schemes is asymmetric, as Section 8.1 quantifies: their `SMLAD` dot-product kernels do not survive LESS's lane-independent elimination, while our $\mathbb{F}_{127}$ SWAR/lazy-reduction layer is candidate material for CROSS's arithmetic. We tested that direction directly: porting our two-lane $\mathbb{F}_{127}$ SWAR to CROSS's own syndrome kernel (`fq_vec_by_fq_matrix` of `crossv1.2-sha3-r-sdp-1-fast`, an $\mathbb{F}_{127}$ multiply–accumulate over a 76×51 matrix) cut it from 43,886 to 34,079 cycles per call on our board, a 1.29× speedup, byte-identical to CROSS's scalar reduction on target. The gain is smaller than LESS's own (1.29× versus the multiples of Section 6.3) precisely because CROSS's matrices are narrow (≤ 51 wide, against LESS's 126–274) and its signing is dominated by hashing rather than by field arithmetic — so the transfer is real but its scheme-level payoff is workload-bounded, a concrete instance of the same dependence-structure lesson as Section 8.1. CROSS itself was not advanced past round 2, so the durable, surviving-candidate value of our work rests on its field-agnostic memory and methodology layers rather than on this same-field arithmetic transfer (Section 11).

Small-field elimination on the M4. Gaussian elimination over small fields has further M4 precedent in the multivariate line: Rainbow's bitsliced constant-time $\mathbb{F}_{16}$ elimination [CKY21] and MAYO's nibble-packed arithmetic [BCC+24] solve secret-dependent systems where constant-time pivoting is mandatory. LESS's elimination sits in a different regime — byte-packed $\mathbb{F}_{127}$, with the secret-adjacent part covered by the scheme's own pivot-reuse bound and blinding (Section 7) — so our kernels optimize throughput under that inherited discipline rather than re-deriving constant-time pivoting; the two-lane SWAR representation is nonetheless a same-family relative of MAYO's nibble packing, adapted to a 7-bit field.

Benchmarking infrastructure. pqm4 and mupq [KRSS19] define the measurement conventions we follow (cycle counting, stack watermarking, shared platform Keccak). The additional-signature benchmarking campaign on this infrastructure [KKPY24] is our motivating citation: it excluded LESS for memory reasons, leaving the scheme without

microcontroller data — the gap this paper fills.

Stack–speed trade-offs for PQC on microcontrollers. Memory-versus-speed restructuring of PQC schemes on Cortex-M-class parts has precedent in the lattice ecosystem (e.g., the compact, stack-reduced Dilithium implementations of Greconici et al. [GKS21], which trade recomputation for buffers), and in the KpqC lattice-KEM/signature line, where the same vocabulary — on-the-fly matrix expansion, streaming, recomputation, and buffer reuse — recurs. Those patterns are not what we claim as novel; our contribution to this line is the explicitly criterion-governed operating-point family and the coverage-matrix reporting format, which we found necessary for a scheme whose parameter sets straddle the device's memory boundary in both directions.

The LEP line. Beyond LESS itself, linear code equivalence continues to host new constructions — e.g., blind signatures [LP25], alongside threshold, ring, and VOLE-in-the-head variants — whose computational cores (RREF, canonical forms, monomial actions over small prime fields) coincide with what this paper optimizes; we expect the kernel and memory results to apply with minor adaptation.

10 Discussion

10.1 Reading the Design Space

The value of characterizing a design space is that different requirements select different points, and the selection is now a lookup rather than a research question. A workload with cycles to spare and reference-level RAM available takes the speed ceiling (spd-asm). One on a part too small for resident matrices but with speed still at a premium takes the balanced knee (bal-asm), which gives up only 8–9 % of the ceiling's signing speed for 41–59 % less stack and still strictly beats the reference on both axes. One minimizing footprint takes the streaming floor (mem-max), which is nearly free at category 1 (Section 6.2). Category 5 removes the choice on a single device: only the two memory-lean policies fit, and of those only bal-asm-DP is measured end-to-end on target. For scale, the fastest category-1 signing point, 871.8 Mcyc, is about 44 s at the 20 MHz benchmark clock and about 8 s at the 120 MHz development clock (whose flash wait states inflate the count, so the ratio is not simply the clock ratio; Appendix B) — still one to two orders of magnitude above the lattice, multivariate, and code-based signatures benchmarked on the Cortex-M4 (Table 9), consistent with the performance factor in the second-round decision; our claim concerns the size of the implementation-induced swing, not the closing of that gap. These are characterization statements about what the scheme admits on the hardware, not deployment recommendations for a scheme outside standardization; the security caveats of Section 7 bound any operational reading.

10.2 On Studying an Eliminated Scheme

Non-selection is not a break, and here it is a precondition of the paper's central finding rather than an argument against it. Schemes not selected by NIST routinely remain in practical use and standardization elsewhere — NTRU Prime has seen deployment in widely used software, and FrodoKEM and Classic McEliece feature in ISO and national standardization efforts. LESS in particular is unbroken — the reported 12–24-bit margin loss [BEFN25] is of the kind ordinarily addressed by re-parameterization, and the cryptanalytic line remains active into 2026 [BE26, BBO+26]. More to the point, the natural experiment behind our systems observation (Section 1) — measuring the gap between what a constrained-device evidence base built on the reference alone would have reported and what the scheme actually

admits — requires a scheme that was eliminated partly on performance, remained unbroken, and lacked any constrained-device implementation: a surviving scheme (already optimized, no counterfactual) or a broken one (performance moot) admits no such measurement. Consistent with this, four of our five contributions — the design-space shape, the technique layers, the memory playbook, and the evaluation methodology — are scheme-independent by construction, and we ourselves build on the Cortex-M4 line of CROSS [SGB+25], itself not advanced past round 2, because implementation-technique work outlives its host schemes. The artifact is parameter-generic — a single build flag selects the parameter set — so a re-parameterized LESS, or any successor in the code-equivalence family, inherits the entire engineering stack unchanged.

10.3 Limitations and Threats to Validity

Single device. All on-target numbers come from one board, the STM32L4R5ZI, chosen for its 640 KB of SRAM so that the resident-matrix configurations have the best possible chance to run — which makes the *coverage* gaps properties of the scheme rather than of an unusually small part. Absolute stack and RAM figures are device-class quantities in the obvious ways; what transfers across Cortex-M4 parts is the relative ordering, the coverage structure, and the cycle counts (which, at the zero-wait-state benchmark clock, are core-microarchitecture properties). One caveat to that last point on this specific part: the STM32L4R5ZI's SRAM is three blocks (192+64+384 KiB), of which only the first is measured at the fastest timing [KKPY24], so a configuration whose stack spills past the first 192 KiB block — `bal-asm` at LESS-400-220 (233 KB), and the resident-matrix category-3/5 configurations — may carry a block-crossing component in its on-target cycles that we did not isolate.

Single toolchain. All builds use one compiler (`arm-none-eabi-gcc` 10.3.1) — the version pinned by the pqm4 release we build against, chosen for comparability with published pqm4 numbers rather than for absolute codegen quality. This matters chiefly for the codegen-dependent transpose observation of Section 8.2, whose `-O3` fault manifestation is a property of how a given compiler lowers an aligned-typed load; the alignment-safe form we adopt is portable regardless. The performance results, being cycle counts of hand-scheduled kernels, are far less compiler-sensitive.

Cycle-count variation. The schemes are deterministic given the DRBG seed, so run-to-run variation is algorithmic (the rejection-sampling path each random instance takes), not measurement noise. We report the median of N instances (N=10 at categories 1 and 3, N=5 or 3 at category 5 by per-signature cost; Section 3.3) and characterized the DRBG-induced spread over N=20 instances (deterministic-scheme key generation within ±0.05 %; signing within ±1.0 %; verification within ±1.6 % as the challenge changes the number of full canonical-form evaluations). The streaming floor's key generation is the exception — its own rejection-sampled regeneration makes it vary by a few percent rather than being deterministic. The headline ratios are well outside these bands, but individual verification figures should be read with them.

Clock. Cycle counts are taken at the 20 MHz benchmark clock, which serves flash with zero wait states and isolates compute cost (the pqm4 convention); at realistic clocks the ART-cached flash adds a few to about twenty percent for the resident and in-place configurations, growing with memory-traffic intensity — and far more for the flash-fetch-bound streaming floor (Appendix B).

Energy: a datasheet-based estimate. We do not measure energy, but the STM32L4R5 datasheet [STM20] parameterizes a first-order model. Active run current is nearly code-independent — 16.5–22.5 mA at 120 MHz across a while-loop, Coremark, and Dhrystone, and about 3.85 mA at 24 MHz — so at a fixed clock the power $P = V_{dd}\, I_{DD}$ (≈71 mW at 120 MHz and 3.3 V) is essentially configuration-independent, and energy per operation $E = P\,(\text{cycles}/f)$ tracks the cycle count. The signing speedups are therefore, to first order, matched by proportional reductions in energy per signature: at the 120 MHz realistic clock, LESS-252-45 signing falls from ≈2.4 J for the reference to ≈0.58 J for the speed ceiling, a ≈4× energy reduction. The streaming floor's higher energy comes not from higher current — the datasheet's run current executing from Flash and from SRAM differs by under 3 % — but from its extra cycles: at the zero-wait-state benchmark clock its energy penalty equals its 1.2–1.5× cycle penalty at category 1, while at higher clocks the flash-wait-state cycle inflation of Appendix B widens it. Direct measurement on a PowerShield-class rig is the natural refinement (Section 11).

Category-5 measurement. For mem-max at category 5 we report correctness (host KAT, byte-identical) and an hours-scale on-target time bound rather than a measured cycle count: the cost is the super-linear recomputation of the streaming design (not a memory limit), and since the regeneration is flash-bound the faster clock does not shorten it, so we report a bound rather than extrapolating. At these sets mem-max is a memory/correctness floor, not a usable speed point — the only practical-speed category-5 configuration is bal-asm-DP, whose category-5 double-pass stack we measure on target (Table 8); the reference category-5 stack, which cannot run on the device, is the x86-massif cross-check on the structurally identical host build.

Scope. The study targets LESS v2.0 as submitted; we make no claim about re-parameterized successors beyond noting that the techniques do not depend on the specific (T, w, s) choices (Section 7).

11 Conclusion

We set out to answer a question the benchmarking record left open: what does LESS actually cost on a microcontroller? The answer is a design space, not a number. On a 640 KB Cortex-M4, the scheme runs at all seven parameter sets. A single memory wall at category 5 separates the resident-matrix configurations from the two memory-lean policies that clear it; of those, the balanced knee — the one we measure on target end to end — simultaneously signs 4.1–4.3× faster than the platform-normalized reference and uses 41–59 % less peak stack, flanked by a speed ceiling at 4.5–4.7× and a memory floor that signs with a one-kilobyte stack watermark at 1.2–1.5× reference time — the last of these far from a slow fallback, and nearly free at category 1. All configurations are byte-identical to the official known-answer tests, and every adoption decision was governed by explicit Pareto criteria. The reference implementation, we found, sits well off its own scheme's frontier — a data point worth weighing wherever implementation-derived performance figures feed scheme-level decisions.

The techniques were built to outlive the scheme, and they transfer along two distinct axes. The arithmetic layer — two-lane $\mathbb{F}_{127}$ SWAR with lazy reduction and the carry-free detectors — is field-specific: it applies directly to other matrix-centric schemes over the same small prime field, CROSS among them. The memory direction (streaming regeneration with row blocking, on-the-fly seed trees, double-pass signing, in-place computation), the rank-2 kernel and TRM audit, and the evaluation methodology (B1/B2 adoption, byte-identical gates, coverage and waterfall reporting) are field-agnostic: they apply to the matrix-centric third-round survivors — the UOV family (UOV, MAYO, SNOVA, QR-UOV)

— for which NIST has explicitly called for constrained-device evaluation, and whose inner loops (rejection-sampled expansion, structured Gaussian elimination) rhyme with the ones optimized here. Carrying them there is our next step, and the negative results of Section 8.1 travel as warnings in both directions.

Energy measurement is deliberate future work: our recomputation-heavy floor and our resident-matrix ceiling should sit at measurably different energy-per-operation points even at equal cycle counts, and quantifying the energy price of memory minimization on a PowerShield-class setup is the natural completion of this characterization.

References

[Ado24] Alexandre Adomnicai. An update on Keccak performance on ARMv7-M. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):1–24, 2024. ePrint 2023/773.

[Arm15] Arm Ltd. *Cortex-M4 Processor Technical Reference Manual, Revision r0p1*, 2015. ARM DDI 0439B; load/store timing rules in §3.3.3.

[BBB+25] Marco Baldi, Alessandro Barenghi, Luke Beckwith, Jean-François Biasse, Tung Chou, André Esser, Kris Gaj, Patrick Karl, Kamyar Mohajerani, Gerardo Pelosi, Edoardo Persichetti, Markku-Juhani O. Saarinen, Paolo Santini, Robert Wallace, and Floyd Zweydinger. LESS: Linear equivalence signature scheme — specification document, version 2.0. NIST PQC Additional Digital Signature Schemes, Round 2 submission, 2025.

[BBO+26] Magali Bardet, Charles Brion, Ayoub Otmani, Mohamed Saeed, and Nicolas Sendrier. Solving the linear code equivalence problem from single codeword matching. Cryptology ePrint Archive, Paper 2026/550, 2026.

[BCC+24] Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer. Nibbling MAYO: Optimized implementations for AVX2 and Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2024(2):252–275, 2024. ePrint 2023/1683.

[BE26] Alessandro Budroni and André Esser. One pair to rule them all: Towards an optimal algorithm for solving code equivalence via codeword search. In *Advances in Cryptology – CRYPTO 2026*, LNCS. Springer, 2026. ePrint 2026/268; to appear.

[BEFN25] Alessandro Budroni, André Esser, Ermes Franch, and Andrea Natale. Two is all it takes: Asymptotic and concrete improvements for solving code equivalence. Cryptology ePrint Archive, Paper 2025/227, 2025.

[BEP+25] Luke Beckwith, André Esser, Edoardo Persichetti, Paolo Santini, and Floyd Zweydinger. LESS is even more: Optimizing digital signatures from code equivalence. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2025(4):386–408, 2025. ePrint 2025/1424.

[BMPS20] Jean-François Biasse, Giacomo Micheli, Edoardo Persichetti, and Paolo Santini. LESS is more: Code-based signatures without syndromes. In *Progress in Cryptology – AFRICACRYPT 2020*, volume 12174 of *LNCS*, pages 45–65. Springer, 2020.

[CKY21] Tung Chou, Matthias J. Kannwischer, and Bo-Yin Yang. Rainbow on Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2021(4):650–675, 2021. ePrint 2021/532.

[CPS25] Tung Chou, Edoardo Persichetti, and Paolo Santini. On linear equivalence, canonical forms, and digital signatures. *Designs, Codes and Cryptography*, 93:2415–2457, 2025. ePrint 2023/1533.

[GKS21] Denisa O. C. Greconici, Matthias J. Kannwischer, and Daan Sprenkels. Compact Dilithium implementations on Cortex-M3 and Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2021(1):1–24, 2021.

[HMK+26] Harry Hart, Puja Mondal, Suparna Kundu, Supriya Adhikary, Angshuman Karmakar, and Chaoyun Li. Lightening CROSS: Memory optimized implementations of CROSS. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2026(2):630–658, 2026. ePrint 2024/1929.

[KKPY24] Matthias J. Kannwischer, Markus Krausz, Richard Petri, and Shang-Yi Yang. pqm4: Benchmarking NIST additional post-quantum signature schemes on microcontrollers. Cryptology ePrint Archive, Paper 2024/112, 2024.

[KRSS19] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4. Cryptology ePrint Archive, Paper 2019/844, 2019.

[LP25] Yi-Fu Lai and Edoardo Persichetti. LEAF: Compact and efficient blind signature from code-based assumptions. Cryptology ePrint Archive, Paper 2025/1585, 2025.

[MAKK24] Puja Mondal, Supriya Adhikary, Suparna Kundu, and Angshuman Karmakar. ZKFault: Fault attack analysis on zero-knowledge based post-quantum digital signature schemes. In *Advances in Cryptology – ASIACRYPT 2024*, volume 15491 of *LNCS*. Springer, 2024. ePrint 2024/1422.

[MKN+25] Puja Mondal, Suparna Kundu, Hikaru Nishiyama, Supriya Adhikary, Daisuke Fujimoto, Yuichi Hayashi, and Angshuman Karmakar. Fault to forge: Fault assisted forging attacks on LESS signature scheme. Cryptology ePrint Archive, Paper 2025/1838, 2025.

[Nat26] 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. NIST Internal Report NIST IR 8610, NIST, May 2026.

[SGB+25] Jonas Schupp, Marco Gianvecchio, Alessandro Barenghi, Patrick Karl, Gerardo Pelosi, and Georg Sigl. Optimizing the post quantum signature scheme CROSS for resource constrained devices. Cryptology ePrint Archive, Paper 2025/1928, 2025.

[SJDN26] Mona Sobhani, Sönke Jendral, Elena Dubrova, and Mats Näslund. Breaking the one-way property of a SHA-3 implementation via fault injection: Key recovery attacks on post-quantum digital signatures. Cryptology ePrint Archive, Paper 2026/619, 2026.

[STM20] STMicroelectronics. STM32L4R5xx datasheet. DS12023 Rev 5, 2020. Run-mode supply current in Tables 26, 34, 36.

[STM21] STMicroelectronics. *RM0432: STM32L4+ Series Advanced Arm-Based 32-Bit MCUs Reference Manual*, 2021.

A Reproducibility

This appendix documents how to reproduce the results; the supporting artifact — source trees, build scripts, and raw campaign logs — will be released publicly with the final version. The logs are the record from which every table derives: the headline cycle tables are medians of N repeated on-target runs (`bench/20260614_nfinal/`, Section 3.3), including a dedicated keygen-only harness (`kg_results.txt`) for the category-5 and LESS-400-220 keygen entries marked ^{b} in Table 5; the memory, waterfall, reference category-5 keygen single runs, and CROSS-transfer (Section 9) results come from the frozen single-run campaign (tag `paper-freeze-v1-20260611`, `bench/20260611/`). The build state is preserved per configuration under the mapping below.

A.1 Repository Layout

One source tree per configuration, with the paper-name mapping kept as a machine-checkable table in the repository (`CONFIGURATIONS.md`):

Table 10: Configuration-to-source mapping in the artifact.

Paper name	Source tree	Key build flags
ref-sub	`clean/`	C Keccak (bundled)
ref-norm	`clean/`	asm Keccak (pqm4)
mem-max	`mem_opt/*-maxopt`	`-O3`
bal-c	`balanced_opt/`	`BAL_SP_ALL, BAL_ME_INPLACE, LESS_LAZY_RREF`
bal-asm	`balanced_opt/`	above + `BAL_USE_ASM, LESS_CF_ASM*` (+ `BAL_ME_DOUBLEPASS` at cat. 5)
spd-c	`speed_opt/`	`LESS_LAZY_RREF`
spd-asm	`speed_opt_asm/`	`LESS_USE_ASM, LESS_LAZY_RREF, LESS_CF_ASM*`

Rejected techniques (Section 8.1) remain in the tree behind default-off flags (e.g., `LESS_PAD_ROWS`, `LESS_BLOCK_RREF`) so each negative result is reproducible.

A.2 Reproducing the Results

1. *Correctness gates.* (i) x86: build the pure-C configuration and run the NIST KAT generator; diff the produced `.rsp` against the official vectors — the repository includes a 16-way parallel KAT runner that re-derives the per-vector DRBG streams and reassembles a byte-identical `.rsp` (LESS-548-345 in under six minutes on a desktop). (ii) M4: flash the testvector binaries for the C and assembly builds and byte-compare the transcripts; run the roundtrip test.

2. *Cycle measurements.* Build via the pinned pqm4/mupq harness (board: Nucleo-L4R5ZI; toolchain and flags pinned in the repository, `arm-none-eabi-gcc` 10.3.1); the campaign script reproduces the full matrix of Table 5 at 20 MHz (`CLOCK_BENCHMARK`), including the waterfall sequence of Table 7, writing one log per configuration–parameter–operation.

3. *Memory.* Stack watermarks via the mupq stack-watermarking method (campaign final phase); x86 cross-checks via `valgrind massif -stacks=yes` on the structurally identical host build (labeled *C-Keccak, x86-64*).

4. *Tables and figures.* A single script converts raw logs to CSV and emits every table and figure in this paper; manual edits are prohibited by policy, so any regeneration either matches the paper or flags a discrepancy.

A.3 Operational Notes

Long campaigns on ST-Link-attached boards occasionally wedge the debug/VCP path; the artifact documents the recovery ladder we used (USB re-enumeration, then a debug-access-port halt and reflash for HardFault states), which kept the full multi-hour campaign autonomous.

A.4 Firmware Code Size

Flash is not the binding footprint on this platform (Section 6.6), but we report code size per pqm4 convention and for the smaller-flash parts to which the techniques may be carried. Table 11 gives firmware `.text` at LESS-252-45.

Table 11: Firmware code size (`.text`) at LESS-252-45, bytes (rows in ascending `.text`). Δ is versus ref-norm (both use the assembly Keccak, so the constant Keccak/harness/runtime code cancels). ref-sub links the bundled C Keccak and is shown for context.

Config	`.text` (B)	Δ vs ref-norm
ref-sub	30,128	(C Keccak)
ref-norm	33,520	—
spd-c	37,872	+13.0 %
spd-asm	40,176	+19.9 %
bal-c	45,552	+35.9 %
bal-asm	48,112	+43.5 %
mem-max	49,136	+46.6 %
mem-max (asm)	49,648	+48.1 %

B Measurements at the 120 MHz Development Clock

The body reports cycles at the 20 MHz benchmark clock, where flash serves with zero wait states so that counts reflect pure compute cost (the pqm4 convention, Section 3.3). At the 120 MHz development clock the flash incurs wait states, mitigated but not eliminated by the ART cache, so cycle counts rise. Table 12 gives a representative comparison at LESS-252-45 (single instance per configuration; the DRBG-induced spread of Section 3.3 applies).

Two observations. First, the inflation grows with a configuration's memory-traffic intensity: the compute-bound reference rises only 3.2 %, whereas the in-place balanced-assembly build, whose in-place-then-recompute structure issues more loads and instruction fetches, rises 20.2 %; key generation (+0.3 to +8.8 %) and verification (+3.9 to +17.8 %) follow the same ordering. Instruction fetch competes with flash wait states, so the clock that isolates our techniques from this platform effect is the zero-wait-state one; this is why the body's attribution uses 20 MHz. Second, the effect is most extreme for the streaming floor's key generation, whose recomputation re-executes the SHAKE core millions of times and is therefore flash-fetch-bound: its LESS-252-45 key generation measures 3,074 Mcyc at 120 MHz against its 678.6 Mcyc median at 20 MHz (Table 5), an inflation of roughly 4.5× — a platform artifact rather than a compute cost, and one more reason the floor's headline numbers are reported at 20 MHz.

Table 12: LESS-252-45 key generation, signing, and verification at 120 MHz versus 20 MHz (Mcyc). The 120 MHz figures are single instances; Δ is the inflation over the corresponding 20 MHz N=10 median (the ref-norm/spd-asm/bal-asm medians appear in Table 5 and the spd-c/bal-c signing medians in Table 6; the remaining C-layer key-generation and verification medians are from the artifact campaign).

Op	Config	@120 MHz	@20 MHz	Δ
keygen	ref-norm	682.5	680.4	+0.3 %
	spd-c	193.0	190.7	+1.2 %
	spd-asm	131.2	128.5	+2.1 %
	bal-c	212.0	200.2	+5.9 %
	bal-asm	151.2	139.0	+8.8 %
sign	ref-norm	4,034.9	3,908.2	+3.2 %
	spd-c	1,321.6	1,232.8	+7.2 %
	spd-asm	989.4	871.8	+13.5 %
	bal-c	1,484.0	1,304.2	+13.8 %
	bal-asm	1,132.8	942.5	+20.2 %
verify	ref-norm	2,659.4	2,558.6	+3.9 %
	spd-c	922.2	848.6	+8.7 %
	spd-asm	717.7	614.3	+16.8 %
	bal-c	963.2	876.3	+9.9 %
	bal-asm	773.4	656.4	+17.8 %

Table 13: Sign inflation at 120 MHz across parameter sets (Mcyc, single instance versus the 20 MHz N-median of Table 5). The bal-asm > spd-asm ordering holds at every set.

	spd-asm			bal-asm		
Set	@120	@20	Δ	@120	@20	Δ
252-45	989.4	871.8	+13.5 %	1,132.8	942.5	+20.2 %
252-68	1,485.5	1,320.4	+12.5 %	1,699.8	1,435.0	+18.4 %
252-192	4,174.2	3,709.0	+12.5 %	4,777.2	4,040.4	+18.2 %
400-102	6,929.0	6,205.9	+11.6 %	7,915.5	6,711.4	+17.9 %
400-220	14,962.9	13,370.1	+11.9 %	17,151.9	14,474.9	+18.5 %

The memory-traffic ordering is systematic rather than a single-set artifact: across all five measured parameter sets the in-place balanced build inflates more than the resident-matrix speed build at every set (Table 13), with LESS-252-45 — the set shown in the table above — the largest-inflation point.