

Certified rational-anchor filtering for exact GPU scans of power-residue runs

Brice Pouly*

12 October 2026

Abstract

We study exact longest monochromatic runs of power-residue colourings, including neutral prefixes and least canonical witnesses. A certified rational cover represents every relevant interval inside a short signed character table. A reflection-compatible schedule retains an anchor; conservative filters reject only known contradictions, bilateral completion recovers the full run, and tie-preserving reduction selects its least witness. We prove this exact-success contract and describe its CUDA realization with separate arithmetic domains below 2^{32} and 2^{33} . On one RTX 5090, two reserved cohorts give ratios 24.00 and 10.95 of sums of hot per-prime medians against the identified historical dense engine. A separate batch study gives throughput factors 1.23–2.58 against an already optimized reference. The experiments have different timing boundaries and their factors are reported separately. Short-table multiplicative extraction has close algorithmic antecedents in cryptanalysis; our contribution is its certified organization for exhaustive maxima and witnesses and its measured implementation. Square-root-sized cover parameters do not give a general $O(\sqrt{p})$ running-time bound.

Keywords. Multiplicative characters; monochromatic runs; Rabung construction; rational approximation; CUDA; reproducible computation.

1 Problem and contribution

Rabung’s construction uses power-residue colourings to produce lower bounds for van der Waerden numbers [1]. A prime and a colour count encode the colouring succinctly, but the finite conditions must still be checked. Computing an exact run profile is therefore useful both for searching for candidate certificates and for building an experimental census. Distributed searches by Monroe provide an important computational predecessor [2, 3]. The published certificates paper separates GPU discovery from pointwise certificate verification [4, 5]. The present paper treats the scanner itself as the object of study. Its exactness theorem concerns ordinary maxima, prefixes and canonical witnesses. The coverage proof and controlled measurements establish this computational contract independently of any census classification or limiting probability law.

Reading contract. A successful scan returns (M_r, L_r, K_r, w_r) : the largest ordinary run, the neutral prefix, the Rabung boundary threshold and the least start of a maximizing run. Section 2 fixes these conventions. Sections 3–4 establish coverage and conservative selection; Sections 5–6 explain profile sharing and execution; Sections 7–8 distinguish timings from implementation checks. Capacity refusal is a reported failure, not a statement that long runs are absent.

The central difficulty is to reduce expensive character evaluations without assuming that long runs behave randomly. A probabilistic estimate for a typical maximum can guide experiments, but cannot justify discarding a candidate in an exact scan. We instead construct a geometric

*Prepared with artificial-intelligence assistance; see the declaration at the end.

hitting set: every relevant interval has a rational representation with bounded numerator and denominator; a short monochromatic anchor then identifies a candidate whose maximal extent is reconstructed exactly.

The contributions developed here are an integer certificate for the cover, a reflection-compatible anchor schedule, conservative bit filters, and an implementation that retains exact maxima, prefixes and canonical witnesses. The implementation combines adaptive warp partitioning, character-profile sharing and concurrent streams, with explicit failures for resource or domain violations. The measured gains have two distinct baselines: the historical dense engine, and a later qualified sparse engine. This distinction is essential to assessing the scale of the improvement.

Minkowski’s theorem, multiplicativity, Montgomery reduction and CUDA streams are established tools. Our claims concern their explicit assembly for this finite scanning problem and the archived measurements. The article does not claim priority for rational approximation or for any general-purpose GPU technique. The related work below identifies close algorithmic precedents. We make no claim that the primitives or their use for extracting sequences were invented here, and no exhaustive priority claim for the complete assembly.

Scope. The experiments use every $r \in \{2, \dots, 9\}$ dividing $p - 1$, rather than only a fixed intersection of congruence classes. The optimized workers accept prime inputs $1000 \leq p < 2^{33}$, using distinct 32-bit and wider paths. Type bounds do not promise successful execution for every input: table, seed and queue guards may refuse a case. Small-prime handling belongs to a separate exact preparation path. All measurements discussed here predate the completed atlas campaign. The atlas and new numerical bounds retain their own evidence; dissemination and evidence limits are collected in Section 9.

1.1 Related work and the precise contribution

First occurrences of power-residue runs were studied by Lehmer and Lehmer and by Rabung and Jordan [14, 15]. Buell and Hudson computed longest quadratic-residue and nonresidue runs for primes below 420 000 [13]. Their publisher’s abstract and bibliography establish that the observable and a census of maxima are older. Their full paper and LSU report 83-017 were not obtained; their internal algorithm is therefore not claimed to differ from ours in every respect.

Beullens, Beyne, Udovenko and Vitto [10], Sections 3.2, 3.4 and 7.2, already exploit multiplicativity, arithmetic subsequences, denominators and reversal to extract many normalized sequences from a short table, with extensions to power-residue symbols. Their Section 3.1 also gives a deterministic table-search variant. Kaluderović, Kleinjung and Kostić [11], Sections 3–5 and Appendix A, develop multiplicative extraction, reuse, reversal and bit-matrix transposition. These are close algorithmic precedents, rather than merely the same elementary identity. Their key-recovery contract does not automatically supply exhaustive run maxima and their canonical ties. Conversely, this different output contract does not erase the precedent for the primitives. We cite the revised ePrint text for these details; the related ANTS XIV article has a different title [12].

Small modular representatives also belong to classical Thue and Euclidean methods and rational reconstruction [16, 17]. Reconstruction seeks a small fraction, often unique if it exists in a regime $2BD < p$; it can legitimately report that none exists. Our clipped window condition implies $BD > p$ and ensures existence for every relevant interval. Neither the lattice principle nor rational normalization is new. The contribution claimed here is the cover–hitting-set–conservative-filter–completion composition with a stated output contract, not priority for its ingredients.

Predecessor	Work and output	Relation to this scanner
Buell–Hudson	Quadratic run maxima and census	Same kind of observable; internal algorithm not inspected.
Monroe paper/code	Full power table, shared orders, maximum/prefix/boundary threshold	Dense precedent; no per-position exponentiation is required to fill the table; no canonical witness exposed by the inspected code.
Beullens et al.; Kaluđerović et al.	Short-table extraction, reversal, bit operations for key recovery	Direct precedent for primitives; different output and coverage obligations.
Thue/reconstruction	Bounded modular representatives; uniqueness/conditional reconstruction	Classical foundations; whole-window existence is the role of our explicit certificate.
This work	Certified window cover, complete folded anchor set, safe rejection, exact completion	Exact M_r, L_r, K_r, w_r on successful execution under the stated hypotheses.

Table 1: Algorithmic comparison, not a common-machine timing comparison or an assertion of universal priority.

2 Exact output contract

Fix an odd prime p and an admissible $r \geq 2$. Choose a surjective additive label for a multiplicative character of order r ,

$$c_r : \mathbb{F}_p^\times \longrightarrow \mathbb{Z}/r\mathbb{Z}, \quad c_r(xy) = c_r(x) + c_r(y).$$

For a primitive root g , one choice is $c_r(g^j) = j \bmod r$. The kernel is the subgroup of r th powers; the choice of labels permutes the colours and has no effect on equality of colours. We scan the ordinary integer sequence $1, \dots, p-1$. Zero is excluded, and an interval does not wrap cyclically through it.

Let M_r be the largest length of a constant-colour interval, and let L_r be the longest initial interval with colour zero. Thus $L_r \geq 1$, since $c_r(1) = 0$. For a maximal run $[l, h]$, its reflected partner is $[p-h, p-l]$, because

$$c_r(p-x) = c_r(x) + \sigma_r, \quad \sigma_r = c_r(-1) = \frac{p-1}{2} \bmod r. \quad (1)$$

The canonical witness is the least start among all runs of length M_r ; reflection allows each pair to contribute $\min(l, p-h)$. A tie is part of the output contract, not a dispensable side effect of finding the maximum.

The scanner also reports

$$K_r = \max\{M_r + 1, B_r\}, \quad B_r = \begin{cases} 2L_r + 2, & r \mid (p-1)/2, \\ L_r + 2, & r \nmid (p-1)/2. \end{cases} \quad (2)$$

Here B_r is a boundary threshold and is unrelated to the table radius B used below. Criterion (a) for a requested length k checks $p > k$ and $M_r < k$; the combined criterion (a)+(b) checks $p > k$ and $K_r \leq k$. These conditions describe distinct outputs. A fast scan of criterion (a) alone is not automatically a certificate for the combined condition. We use colour-first notation $W(r, k)$, with r colours and progression length k , as in the published certificates paper [5]. That paper proves the boundary-test equivalence and owns the numerical bound comparisons. The present output contract supports those applications; no new lower-bound theorem is announced here.

2.1 Dense reference

The historical CUDA engine evaluates the character across a half interval and uses (1) to recover the full profile. It fuses admissible character orders using their least common multiple. Consecutive segments retain their endpoint colours, prefix, suffix and internal maximum; an ordered associative merge reconstructs boundary-crossing runs. When the central colours match, the central junction is merged explicitly. The baseline therefore already exploits symmetry, parallelism and shared character evaluation. Comparing against it is more informative than comparing with a naive serial scan.

Monroe's inspected `uc2.cpp` builds a common discrete-log table by successive multiplication of a primitive root, then scans its entries for several orders [3]. Filling it takes $O(p)$ modular multiplications and $\Theta(p)$ stored labels after root preparation; the pass across a fixed set of orders is linear. Thus a dense method need not perform one exponentiation per position. The historical CUDA baseline here instead trades dense storage for character evaluation over the half interval. Its small root-image table has at most 2520 entries. Our sparse engine prepares $O(B)$ character values, visits scheduled words and completes survivors. These are different work/memory tradeoffs; no timing factor against Monroe is available. Sharing orders and the boundary formula are also present in that earlier software, not introduced here.

2.2 A root of the required order

Let $\mathcal{R}$ be the requested orders and $m = \text{lcm } \mathcal{R}$. For the admissible subset of $2, \dots, 9$, $m \leq 2520$ and $m \mid p-1$. To tabulate the image of $x \mapsto x^{(p-1)/m}$, it suffices to find an element z of order exactly m . Trial values $z = a^{(p-1)/m}$ are accepted if

$$z^{m/q} \neq 1 \quad \text{for each prime } q \mid m.$$

Only the small integer m must be factored; a primitive root of order $p-1$ is unnecessary. The case $m = 2$ uses $z = -1$. Any two choices of z differ by a power u invertible modulo m . The resulting relabelling preserves colour equality, the zero prefix, subgroup refinements and (1). This optimization does not compute an unrestricted discrete logarithm.

3 Certified rational covers

Multiplicativity gives

$$c_r(a/d) = c_r(a) - c_r(d) \quad (a, d \neq 0 \pmod{p}). \quad (3)$$

Consequently, if an interval admits representatives $a, a+d, \dots, a+(Q-1)d$ in a short signed table, testing its colours requires table comparisons instead of one modular exponentiation per position. The following certificate guarantees such a representation. It is an application of the classical convex-body theorem [6]. The short-table use of (3) has algorithmic precedents in cryptanalysis [10, 11]; the certificate supplies the exhaustive window guarantee needed for our distinct output contract.

Theorem 3.1 (Integer coverage certificate). *Let $3 \leq Q \leq p-1$, $H = Q-1$, and let positive integers B, D satisfy*

$$B < p, \quad D < p, \quad HD \leq 2B, \quad 4BD - HD^2 > 4p. \quad (4)$$

For every interval $x, x+1, \dots, x+H$ in $\{1, \dots, p-1\}$, there are integers a, d with $1 \leq d \leq D$, $\gcd(a, d) = 1$ and

$$a \equiv dx \pmod{p}, \quad |a + jd| \leq B \quad (0 \leq j \leq H).$$

None of these numerators is zero.

Proof. The lattice

$$\Lambda_x = \{(a, d) \in \mathbb{Z}^2 : a \equiv dx \pmod{p}\}$$

has basis $(p, 0), (x, 1)$ and determinant p . Apply the determinant-one shear $(a, d) \mapsto (A, d) = (a + Hd/2, d)$. In these coordinates the centrally symmetric convex body

$$\mathcal{C} = \{(A, d) : |d| \leq D, |A| + H|d|/2 \leq B\}$$

has area

$$\int_{-D}^D (2B - H|d|) dd = 4BD - HD^2 > 4p.$$

Minkowski's theorem supplies a nonzero lattice point. If $d = 0$, then a is a nonzero multiple of p with $|a| \leq B < p$, a contradiction. Change sign so that $d > 0$. The convex-body inequality bounds both endpoints a and $a + Hd$ by B in absolute value, hence every intermediate numerator. If $s = \gcd(a, d) > 1$, divide both coordinates by s . Since $s \leq d < p$, s is invertible modulo p ; the congruence is preserved and the bounds only decrease. Finally, $a + jd \equiv d(x + j) \not\equiv 0 \pmod{p}$, so no numerator can equal zero. $\square$

The shear can have half-integer coordinates when H is odd. It still maps the lattice by a real linear transformation of determinant one, so its covolume is unchanged. There is no rounding step in the proof. For an interval of length $q \leq Q$, extend it to a Q -interval inside $\{1, \dots, p-1\}$, when $Q \leq p-1$, and restrict the representation. Thus the same certificate covers all shorter thresholds.

A complete small example. For $p = 101$, $r = 2$, $Q = q = 5$, $H = 4$, $D = 4$ and $B = 30$, the integer certificate is $HD = 16 \leq 60$ and $4BD - HD^2 = 416 > 404 = 4p$. The window $19, \dots, 23$ has numerator row $-25, -21, -17, -13, -9$, since $-25 + 4j \equiv 4(19 + j) \pmod{101}$. Multiplicativity reads its colours in the signed table after subtracting the common label of 4. With $t = 3$, spacing $s = q - t + 1 = 3$ and phase $\phi = 1$, the folded anchor has $b = 17 = 4 \cdot 4 + 1$: its quotient $n = 4$ satisfies $n \equiv 1 \pmod{3}$. Completion uses $x = -17 \cdot 4^{-1} = 21 \pmod{101}$ and recovers $[19, 25]$. Its reflected partner is $[76, 82]$, so its canonical start is 19. A literal character scan gives $(M_2, L_2, K_2, w_2) = (7, 1, 8, 19)$. This arithmetic example illustrates the cover, selection and completion; it is below the optimized workers' accepted range and is not a GPU timing or an additional qualification input.

3.1 Choosing parameters without trusting floating point

On the branch $HD^2 \leq 4p$, the least table radius satisfying the strict area inequality is

$$B = \left\lceil \frac{4p + HD^2}{4D} \right\rceil + 1. \quad (5)$$

The branch condition also ensures $HD \leq 2B$. The implementation checks all of (4) using integer arithmetic after choosing parameters; a rounded square root can propose a candidate, but cannot certify it. Table capacities and signed filter indices impose additional guards.

Ignoring integer rounding, let $D_0 = 2\sqrt{p/H}$ and choose $D = D_0/n$. Then the area boundary gives

$$B \simeq \sqrt{Hp} \left(\frac{n}{2} + \frac{1}{2n} \right), \quad BD \simeq p(1 + n^{-2}). \quad (6)$$

Changing $n = 2$ to $n = 4$ reduces this rectangle proxy from $1.25p$ to $1.0625p$, while increasing the table radius. Larger tables put additional pressure on cache and shared memory. This is an architectural tradeoff, not a theorem that larger n is always faster.

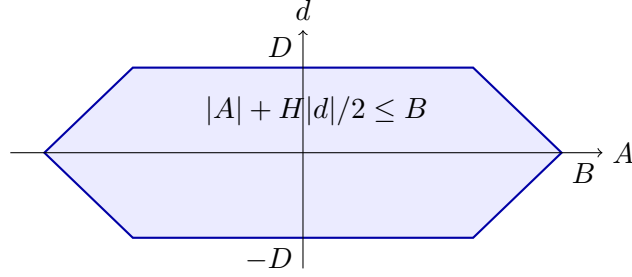


Figure 1: The symmetric clipped cover in sheared coordinates. Its area is $4BD - HD^2$. The diagram illustrates the certificate, not a measured work count.

4 Anchors, conservative rejection and exact completion

The cover supplies a bounded arithmetic progression of numerators for any run worth retaining. The implementation enumerates short anchors from this representation, filters them in words of 32 candidate bits, and extends the survivors to their true boundaries in the original sequence.

4.1 A reflection-compatible hitting set

Suppose a run contains a q -window and $3 \leq t \leq q$. There are $s = q - t + 1$ possible starts for a t -anchor within that window. Along a rational row these starts have numerators differing by d . A periodic selection with period s therefore hits at least one of them. The signed table is folded using reflection. It is useful to make the selection commute with that fold, rather than relying on a favourable location of the run.

Lemma 4.1 (Symmetric anchor schedule). *For the increasing representation $a, a + d, \dots, a + (t-1)d$, define $b = a + (t-1)d$. The same anchor, in descending order, is $(b, b-d, \dots, b-(t-1)d$. Negating and reversing it sends its first numerator to $(t-1)d - b$. For $d > 2$ and a primitive pair (b, d) , write $b = nd + c$, $0 < c < d$, using Euclidean division even if b is signed. Choose any phase $\phi \in \{0, \dots, s-1\}$, and select b if*

$$n \equiv \begin{cases} \phi, & 2c < d, \\ t-2-\phi, & 2c > d \end{cases} \pmod{s}. \quad (7)$$

Selection is invariant under $b \mapsto (t-1)d - b$. Every q -window in a primitive rational row contains a selected t -anchor, whose orientation may be chosen to satisfy $b \geq (t-1)d/2$. For $d = 1, 2$, retaining all starts has the same hitting property.

Proof. Reflection replaces n by $t-2-n$ and c by $d-c$, which exchanges the two branches of (7). The cases $c = 0$ and $2c = d$ cannot occur in a primitive pair with $d > 2$. Within the q -window the s possible starts have one fixed nonzero residue c and s consecutive quotients n , so exactly one satisfies the appropriate congruence. If its centre is negative, reflection makes it nonnegative and preserves selection. All reflected numerators retain their absolute table bounds. $\square$

After the fold, the enumeration domain is

$$1 \leq d \leq D, \quad \left\lceil \frac{(t-1)d}{2} \right\rceil \leq b \leq B, \quad \gcd(b, d) = 1. \quad (8)$$

The span $(t-1)d \leq HD \leq 2B$ ensures that all anchor numerators lie in the signed table. Zero numerators are excluded. The right endpoint of the unreflected anchor is $bd^{-1} \bmod p$; the completion code uses the reflected increasing anchor beginning at $x = -bd^{-1} \bmod p$, with

numerators $-b + jd$. It checks $x \neq 0$ and $x + t \leq p$, so it never treats a zero-crossing cyclic interval as a run.

In `periodic_bands.hpp`, a band has $b = nd + c$ with $1 \leq c < d/2$ or $d/2 < c \leq d - 1$, clipped to (8). The complementary phase is $(t - 2 - \phi) \bmod s$. The retained kernels use one phase $\phi = \lfloor (t - 1)/2 \rfloor \bmod s$ for the whole denominator row in an epoch, not independent phases for adjacent bands. The wheel $G = \gcd(d, 6)$ restricts to $b \equiv c_G \pmod{G}$: one row for $G = 1, 2$, two rows for $G = 3, 6$. Additional masks can remove some nonprimitive pairs; the exact gcd check occurs in completion. A primitive pair is never removed. For $d = 1, 2$ all primitive starts are retained. Ranks enumerate the first clipped band, the translated interior bands and the last clipped band; `floor_div/ceil_div` explicitly handle signed endpoints. Colour values do not change this partition.

Total policy for anchor length (retained mode 13). The implementation is `anchor_t` in `gpu_pipeline.cuh`. For $q \geq 3$ put

$$h = \lceil q/2 \rceil, \quad s_0 = q - h + 1, \quad u = 2^{\lceil \log_2 s_0 \rceil}, \quad v = q - u + 1.$$

Set $z = v$ only if $h \leq v \leq h + 2$; otherwise set $z = h$. Return $t_{\text{proposed}} = \min(q, \max(3, z))$. The selected 32-bit worker chooses it at seed completion and keeps that t for later epochs; its q_e can still increase. The selected wider worker enables dynamic anchor length: a later epoch uses $t_e = \max(t_{e-1}, t_{\text{proposed}})$. Verified maxima and thus q_e are nondecreasing. Both policies preserve $3 \leq t_e \leq q_e$. For $q = 12$ the proposal $v = 9$ is rejected, yielding $t = 6, s = 7$; for $q = 13$, $v = 10$ is rejected, yielding $t = 7, s = 7$. Power-of-two spacing is an optional optimization with a general-period fallback, not a universal promise within two increments. The finite revision check covers every $3 \leq q \leq 256$ and the monotone epoch invariant without running CUDA. Completion can return a run much longer than Q .

4.2 What a bit filter is allowed to reject

For each character bit plane, aligned table words are compared at the offsets of an anchor. A candidate survives only when all its known anchor colours agree. Quotienting by d changes each label by the same constant, so equality of numerator colours is sufficient. A wheel can omit pairs whose common divisor prevents them being primitive. A skipped residue must be excluded arithmetically, not because it looks unlikely.

Entries outside the table are unknown. They do not count as a mismatch. This convention is essential near clipped boundaries and when the completion extends beyond the representation certified by the short window.

Lemma 4.2 (Bilateral rejection). *Consider a constant-colour anchor of length t . Suppose a different known colour is observed $i \geq 1$ places to the left of its first position and $j \geq 1$ places to the right of its last position. The run containing the anchor has length at most $t + i + j - 2$. In particular, if both mismatches are found within budgets U, V with $U + V \leq q - t + 1$, that run has length less than q .*

Proof. The run can include at most $i - 1$ additional positions on the left and $j - 1$ on the right. For the budget statement, its length is at most $t + U + V - 2 \leq q - 1$. $\square$

This lemma explains a simple two-cut filter. The retained mode is stronger: it considers every placement of a q -window containing the anchor, and keeps a candidate if at least one placement has no known mismatch. An actual run of length at least q supplies such a placement, so rejecting all placements is safe. The implementation performs this screen only when $q - t \leq 32$; otherwise it retains the candidate. Its mode-four guard is `h>32`, with a 33-element array for positions $0, \dots, 32$; this path does not shift a 32-bit word by 32. Neither filter assumes that the anchor starts at the head of a maximal run.

4.3 Thresholds and canonical ties

A pilot establishes a real monochromatic run and hence a lower bound for the maximum. Each denominator epoch takes a snapshot of this bound and caps its geometric threshold by Q . For a coupled pair of profiles, the shared search uses a threshold no larger than either current maximum. The threshold stays immutable throughout the generation and completion kernels of that epoch; it is updated only after ordered completion.

To retain the least witness, the search must preserve anchors in runs whose length equals the current maximum. It may reject runs certified to be strictly shorter. At completion, each surviving anchor is mapped to a position of the original sequence using (3), extended in both directions, and reduced by length and then by least canonical start. The prefix is checked directly using modular powers. If the pilot cannot establish the required threshold, or a queue overflows, the engine produces a failure rather than an approximate maximum.

The geometric certificate, the hitting-set argument and conservative rejection explain why a relevant run is retained at the algorithmic level. Correctness of its CUDA realization additionally depends on the finite partition, synchronization, arithmetic and output checks described below. A test passing does not replace a proof of the cover, and a proof of the cover does not verify an arbitrary CUDA implementation.

4.4 Composition into an exact-success contract

Consider a true global maximum. Its covering denominator is assigned to one epoch. The verified maximum available at that epoch gives a threshold no larger than the true maximum, so a subwindow at that threshold contains a scheduled anchor. Known contradictions cannot eliminate that anchor. Exact completion recovers the run, and the tie rule keeps its least start. Later increases in the threshold do not require revisiting earlier epochs. The theorem states precisely the implementation hypotheses for this argument.

Theorem 4.3 (Exactness on successful termination). *For each profile r , fix (B, D, Q) satisfying Theorem 3.1, with $3 \leq Q \leq p - 1$. Suppose disjoint denominator epochs exhaust $\{1, \dots, D\}$. In epoch e , use a fixed threshold*

$$3 \leq q_e \leq \min(Q, m_{r,e}) \leq \min(Q, M_r),$$

where $m_{r,e}$ is the length of a run verified before that epoch, and use $3 \leq t_e \leq q_e$. Suppose enumeration preserves the complete scheduled primitive anchor set, modulo reflection; rejection is conservative for each profile; completion and colour arithmetic are exact; and each surviving obligation is completed to its true boundaries without passing zero. Assume no lost task, capacity overflow, failed arithmetic guard or unfinished epoch is accepted as success. Then reduction by length and least canonical start returns M_r, w_r . An exact prefix check returns L_r , and (2) returns K_r . For coupled profiles these assumptions apply to each profile separately; candidate routing must preserve their union of obligations.

Proof. Verified seed and completed runs have length at most M_r . Fix any maximum run and a subwindow J of length $\min(M_r, Q)$. Extend J to a Q -window within $1, \dots, p - 1$ if necessary. The cover gives a primitive representation with some $d \leq D$; restriction preserves its primitivity and table bounds. That denominator belongs to an epoch e . Because $q_e \leq |J|$, a q_e -subwindow contains a scheduled t_e -anchor. Reflection may place it in the reflected run of the same length. An actual monochromatic window has no observed contradiction, so a conservative profile filter retains its obligation. Exact completion finds the maximum run or its partner. This holds for every maximum run, so the reduction also finds the least start among ties. Completion produces only real runs, excluding an overestimate. Direct prefix verification and the integer formula give the remaining outputs. $\square$

Algorithm 1 (logical specification; each failure is visible).

1. Validate input, arithmetic and capacities; certify fixed group covers; prepare signed character planes and inverses.
2. Establish real seed runs and their canonicalized witnesses. Refuse if the required threshold cannot be established.
3. For each denominator epoch, snapshot per-profile maxima, choose q_e, t_e , and build whole-row folded plans.
4. Generate primitive anchor words; test coarse anchors and maintain separate profile obligations. Reject only under the appropriate profile’s known contradictions. Queue a word if any obligation remains.
5. Complete queued obligations and reduce maxima/ties; check queue counts, errors, dependencies and epoch completion before advancing.
6. Verify witnesses and prefixes, form K_r , and report success only after all epochs for all requested groups have completed.

The theorem specifies the algorithm’s mathematical contract. It is not a formal verification of CUDA instructions, binary identity or the absence of every software defect. Evidence for those implementation bridges is recorded separately in Section 8 and Appendix B.

5 Sharing character profiles

Subgroup refinements provide opportunities to reuse table bits and plans. For orders $2 \mid 4$, an ordinary binary encoding of the order-four label has its low bit equal to the order-two label. Only one residual plane is needed. For orders $3 \mid 6$, encode an order-six label c as

$$(c \bmod 2) + 2(c \bmod 3).$$

Its two high planes are the order-three representation and its low plane is the residual parity. Signed-table access must also apply the appropriate reflection shifts and strides. Reusing an equal-looking array without these transformations would not preserve the character contract.

The encoding is a bijective storage representation, not an additive character label in $\mathbb{Z}/6\mathbb{Z}$. Colour equalities and the documented reflection transformations, rather than integer addition of these codes, are used.

For a coarse/fine pair, a fine-constant anchor is coarse-constant, but the reverse need not hold: $(0,3,0)$ is constant modulo three and not modulo six. Thus accepting only the fine equality would lose coarse maxima. `profile_masks.hpp::split` first tests the coarse anchor and keeps two masks. Coarse-kept anchors are completed as entire coarse runs and their interior fine subruns are scanned by `coupled_device.hpp::complete`. Fine-only bits refine the residual planes and are completed directly. A word is queued if either mask is nonzero. Fine work already covered by a coarse completion is not queued twice. The logical obligation is the union, not the intersection, of the profiles’ relevant candidates. A shared low threshold alone would not suffice without this routing rule.

Table preparation, plan preparation and candidate completion have different costs. It can be faster to share the table while scanning profiles separately, because the coarser character then has a stronger threshold. The retained 32-bit path separates order two from order four, but leaves orders three and six coupled. Each group retains its own B, D, Q and epochs. This policy follows measurements, not an assertion that every subgroup pair should be coupled.

The later wider worker also shares two planes of the order-eight encoding with order four, retaining one residual plane when $p \equiv 1 \pmod{8}$. It improved the measured natural batch throughput by a further 3.70–4.16% relative to streams alone in the two late wide validation

windows. The same specialization was not retained in the 32-bit worker, where exploratory gains were too small. Generic couplings and the order-three/order-nine pairing were tried and not retained.

6 Arithmetic and CUDA execution

6.1 A separate wider arithmetic path

The 32-bit and wider workers have different type requirements. For $1000 \leq p < 2^{33}$ and $r \geq 2$, every colour class has $(p-1)/r$ elements, so

$$M_r \leq (p-1)/r < 2^{32}.$$

Reflection gives a canonical witness no larger than $(p-1)/2 < 2^{32}$. Length and witness can therefore still be packed into a 64-bit reduction key with 32 bits each. A key of the form

$$(\text{length} \ll 32) \mid (2^{32} - 1 - \text{witness})$$

implements the desired lexicographic order by maximization. However, p , positions and intermediate endpoints must be 64-bit, and K_r must be formed in 64-bit arithmetic. In particular, a witness above $2^{31} - 1$ is valid and must not become a negative signed integer.

The wider path uses Montgomery multiplication [7] with $R = 2^{64}$ and $n' = -p^{-1} \bmod R$. For reduced operands $a, b < p$, write $T = ab$ and $m = (T \bmod R)n' \bmod R$. Then

$$u = (T + mp)/R < 2p < 2^{34}.$$

The low halves cancel modulo R ; their carry equals one precisely when the low half of T is nonzero. Hence u can be formed from the two high products and that carry, using `__umul64hi`, without overflowing the 64-bit high sum in this domain. One conditional subtraction of p reduces the result. Inverses stored for completion use the same Montgomery representation. The case $R = 2^{32}$ for a modulus greater than R requires different bounds; a tested variant did not improve the complete scan and was not retained.

These arguments do not authorize a parser-only extension beyond 2^{33} . Wider maxima and witnesses, candidate tags, indices, product carries and independent verification would all need review. The separate wider worker is qualified only on its recorded finite inputs.

6.2 Bounds on integer intermediates

The selected covers guard $3 \leq Q \leq 256$, $B, D \leq 2^{20}$ and $B, D < p$. With $H \leq 255$, $HD < 2^{28}$, $HD^2 < 2^{48}$, $4BD \leq 2^{42}$ and $4p < 2^{35}$ in the wider domain. The guards use unsigned 64-bit products; $HD \leq 2B$ is checked before subtracting HD^2 , preventing unsigned underflow in the area test. The lower-branch constructor also checks $HD^2 \leq 4p$. These bounds justify the implemented 64-bit certificate, not an unrestricted multiplication of unbounded parameters.

Band endpoints and translations use 64-bit products before checked conversion to bounded indices. A filter word can extend 31 entries beyond its clipped band. With $t, q \leq 256$ and $d/G \leq 2^{20}$, signed filter offsets have magnitude below 2^{29} and fit a signed 32-bit index. The 30-bit denominator field and two wheel-tag bits are sufficient for $D \leq 2^{20}$. Candidate counts and atomic reservation use unsigned 64-bit integers; queue capacity is checked before completion. Its configured 2^{21} words of 16 bytes occupy 32 MiB per worker slot, excluding other arrays. During scalar completion an outside-table numerator can have magnitude below $pd + B < 2^{54}$, so its signed 64-bit representation is safe. Positions, character moduli, and K_r have the separately stated wider types. None of these bounds is inferred from the size of the final answer alone.

6.3 Word partitioning and shared plans

Rows with small denominators contain more work. An adaptive decomposition assigns them more warp chunks while preserving the partition

$$\text{rank} = 32 \text{ chunk} + \text{warp lane} + 32C \text{ round}, \quad 0 \leq \text{chunk} < C, \quad 0 \leq \text{lane} < 32. \quad (9)$$

Every nonnegative rank has exactly one such decomposition: divide by $32C$ to obtain the round, then divide the remainder by 32. Clipping the rank to the row length therefore omits no valid task and duplicates none. This changes the scheduling of work, not the coverage set.

The retained 32-bit path caches a denominator/residue plan of 1472 bytes in shared memory. One leader prepares it and other warps reuse it. Physical threads whose warp has no final task must still visit every block barrier. The plan is copied into registers before the next iteration overwrites the shared cache. Both static and dynamic shared allocations count towards the per-block budget and the opt-in decision.

The experiments were conducted on an RTX 5090 with 170 SMs, compute capability 12.0, a Ryzen 7 9800X3D, driver 615.71.09 and CUDA compiler 13.3.73. The SM120 limits differ from those of Blackwell data-centre parts: NVIDIA documents 48 resident warps, 128 KiB shared memory per SM and 99 KiB shared memory per block for capability 12.0 [9]. The selected compilation caps registers at 64 per thread. This cap is an empirical compromise among occupancy, spills and additional instructions; it is not an architecture-independent prescription.

6.4 Independent workspaces and bounded batches

Multiple primes are independent. The persistent workers give each active worker slot a private stream, character tables, candidate queue, events, host buffers and device state. Kernel attributes are initialized before concurrent work rather than mutated for individual primes. Results are stored by input index and published in input order after all slots finish. A failure refuses the batch.

The 32-bit worker uses eight worker slots when the largest prime in a batch is below $5 \cdot 10^8$, four slots below $3 \cdot 10^9$, and two slots thereafter. The wider worker uses four slots. Batches contain at most 128 primes and are governed by a 30-second deadline in the experiment driver. The denominator domain is partitioned into disjoint half-open epochs covering $[1, D + 1)$; successful completion must account for every epoch. Queues and table capacities are bounded and checked. These contracts provide an exact success result or a visible failure, rather than silent loss of candidates.

6.5 Cost interpretation

For fixed Q, n and a fixed number of profiles, both B and D scale like $\sqrt{p}$, while the rectangle proxy BD scales like p . The enumeration of table words is bounded by a constant multiple of BD for this fixed geometry; packing 32 candidates changes that constant. Let W count visited anchor words, C_{tab} count scalar positions inspected during bilateral completion whose colours are resolved in the short table, and E count outside-table colour evaluations. Let F count additional positions inspected for fine subruns within completed coarse runs. A conditional cost decomposition is

$$\begin{aligned} T = & T_{\text{root/table/plan}} + O(W + C_{\text{tab}}) + O(E \log p) + O(F) \\ & + T_{\text{prefix/verify}} + T_{\text{transfer/scheduling}}. \end{aligned}$$

Here $O(E \log p)$ counts modular multiplications for binary powering, with fixed profile counts and word size. In-table completion steps are cheaper, but still require scalar work. Distinct anchors can complete the same run. The selected queues pack candidate bits without establishing a global deduplication or amortization bound: none of C_{tab}, E, F is silently absorbed into $O(W)$

or proved distribution-free sublinear. The archived timings are unchanged; no new completion counter is attributed to them. Completion cost depends on the number and extent of surviving anchors. We do not derive a distribution-free square-root bound for the complete scanner, nor a universal speedup from an assumed random character. The empirical rarity of long surviving anchors is useful for performance but is not required by the rejection proofs.

7 Experimental comparisons

Two archived experiments answer different questions. We identify the engines, input selection, timing boundary and aggregation before presenting each factor. The packet retains the source snapshots and underlying JSON records; the offline checker regenerates the tables from those records.

7.1 Historical engine versus the rational-anchor prototype

The historical source is linked to the bounds repository at commit 704b4125b274bab3df45454adc9c421f8068b8a1. The archived comparison records that the engine prefix preceding `main` is identical; its SHA-256 is

b5ba2f527539d2bcd1d0f5ceaa6010afb9588e9655baf591047fdd5da723f64b.

The full source files differ in the diagnostic policy of the entry point. The adapter calls the same engine function for the full admissible mask. This establishes the identity of the scanner being compared, not the identity of an executable or hardware configuration used weeks earlier.

The final 32-bit prototype was frozen before reserved validation. The natural cohort consists of 32 consecutive primes near each of 0.50, 1.51, 2.71 and 4.20 billion, with no congruence filter. A separate stress cohort uses eight primes congruent to one modulo 2520 in eight other ranges, so all eight requested orders occur. Another 15-prime cohort was familiar from tuning and is labelled exploratory control. All runs compute exact M_r, L_r, K_r for the same admissible profiles. Historical engine comparisons do not establish equality of canonical witnesses, which its adapter does not expose; that check uses the later reference and independent oracles.

For each engine pair, the schedule is ABBA followed by BAAB. Each command has six iterations; iteration zero is excluded, leaving 20 hot samples per prime and engine. The statistic is

$$S_e = \sum_{p \in \mathcal{P}} \text{median}_{i \in \text{hot}} T_e(p, i), \quad G = S_{\text{historical}} / S_{\text{optimized}}. \quad (10)$$

It includes CPU preparation, transfers, kernels, waiting and the checks inside the timing boundary. Process/GPU initialization and input validation are excluded. The optimized path also checks its witness; the historical adapter computes the prefix by a direct power loop. The adapter and wrapper are part of the benchmark specification. S_e is a sum of per-prime medians, not the elapsed wall time of one whole campaign or batch.

Cohort	Primes	Historical S (ms)	Optimized S (ms)	G
Natural reserved	128	2534.1976	105.5793	24.003
Profile-rich reserved	8	186.4518	17.0243	10.952
Familiar control	15	318.3642	20.7789	15.321

Table 2: Direct comparison with the historical 32-bit engine on one GPU. The control row is not a reserved validation result. The common compared output is M_r, L_r, K_r ; the historical adapter does not expose w_r . The two reserved cohorts are structured samples.

The same archive records intermediate baselines and a direct comparison between two wider prototypes. They do not create a chain of multiplicative speedups: each ratio has its own runs

and timing conditions. The wider prototype’s reserved comparison yields 1.754 on 16 primes, relative to its first functional wider version. The historical 32-bit engine cannot accept those wider inputs.

7.2 Later batch throughput versus an optimized reference

The later study asks how much remains after the rational-anchor engine is already qualified. Its baseline is that previous optimized worker, running under the same persistent communication protocol. Exploration and selection were separated from the recorded validation cohorts. Four final 32-bit windows and two late wider windows each contain 1024 consecutive primes; all admissible $r = 2, \dots, 9$ are requested. Inputs are grouped into eight batches of 128 primes, with six repetitions and shuffled engine order by batch. The wider order-four/order-eight specialization was selected before the two late wider windows; two earlier wider validation windows use the preceding stream version and are not attributed to the final specialization.

For each repetition, the eight batch wall times are summed. The reported time is the mean of those six sums, and the dispersion is their sample standard deviation. This dispersion describes timing variability on fixed inputs, not uncertainty over the population of primes. This boundary includes worker calculations and their communication to the driver, and excludes process startup, the initial sieve, durable storage and campaign audits. CUDA event-time sums for overlapping primes are not substituted for batch wall time.

Window start / 10^9	Path	Reference (s)	Selected (s)	Factor
0.257	32	0.3769 ± 0.0009	0.1461 ± 0.0009	2.580
1.337	32	0.7148 ± 0.0013	0.4461 ± 0.0009	1.602
2.971	32	1.1106 ± 0.0007	0.8447 ± 0.0016	1.315
4.237	32	1.3660 ± 0.0007	1.1118 ± 0.0018	1.229
6.023	33	1.6345 ± 0.0021	1.1628 ± 0.0010	1.406
8.041	33	1.9084 ± 0.0028	1.4750 ± 0.0031	1.294

Table 3: Six final-engine validation windows, 1024 primes per window. Times are means and sample standard deviations of six repeated batch sums. “Path 33” denotes the wider arithmetic worker, not the active atlas scope.

All six repetitions in each window favour the selected worker. Table 3 quantifies an additional improvement over an already optimized reference, whereas Table 2 quantifies the earlier larger change. The cohorts, timing boundaries and aggregation differ, so multiplying a 24.00 factor by a 2.58 factor would not produce a measured end-to-end comparison. A throughput factor G corresponds to a time reduction $1 - 1/G$, not $G - 1$.

Replacing repeated trial division by deterministic Miller–Rabin tests to the first twelve prime bases is another retained change. The smallest composite passing those bases is above 2^{64} [8], so this is an exact primality test within the worker’s integer input domain. The isolated validation component improves by about 57 times in its archived microbenchmark; the complete worker gains only about 5–7% in the 32-bit windows and about 2% in the earlier wider windows. This example illustrates why component gains must not be substituted for whole-worker gains. Primality coverage to 64 bits does not enlarge the CUDA arithmetic domain beyond the separately checked bound.

7.3 Effective configurations and explanatory diagnostics

Per-prime B, D, t_e, q_e depend on the input and epoch; the constructors and total policy specify them. The compact throughput records do not retain their full traces, visited-word counts, survivor counts, outside evaluations or queue/memory peaks. We do not invent those diagnostics or an incremental ablation of every primitive.

Parameter	Selected 32-bit	Selected wider
Base Q ; denominator divisor n	48; 4	24; 4
Group Q : order 4; pair 3/6; other high orders	20; 32; 32	pair 4/8: 16; 12; 12
Main generator threads/block	1024	1024
Completion threads/block	128	128
Epoch denominators	8192	16384
Generator grid cap / SM count	8	4
Completion grid / SM count	12	12
Low/mid/high warp chunks	64 / 8 / 8	32 / 8 / 8
Register cap; queue words/slot	64; 2^{21}	64; 2^{21}
Concurrent worker slots	8, 4, 2 by batch range	4
Anchor mode; dynamic t ; graphs	13; disabled; disabled	13; enabled; disabled
Additional pilot	table seed	1536 positions, 512 threads

Table 4: Runtime and build parameters in `source/CONFIGURATION.json` and worker build records. Group covers remain fixed while verified epoch thresholds adapt. The grid entries count launched blocks divided by the SM count; the generator can launch fewer blocks for a short epoch. They do not count simultaneously resident blocks. The wider pilot’s 512-thread generator is separate. Capacities are not observed memory/queue peaks.

The archived batch records do preserve preparation, setup, queue and verification counters. For the selected window near 2.971 billion, mean sums over eight batches are 181.53, 3082.42 ms (GPU preparation / queue) and 2.14, 8.71, 5.32 ms (CPU preparation / setup / verification); near 8.041 billion they are 258.28, 5502.64 ms (GPU preparation / queue) and 2.07, 8.53, 10.16 ms (CPU preparation / setup / verification). The preparation and queue fields sum per-prime CUDA event times (queue includes generation, completion and prefix kernels); CPU preparation/setup/verification fields sum per-prime host times. Concurrent streams overlap: these fields are diagnostics of work and must not be added as a partition of batch wall time or interpreted as percentages of it. They show why generation and completion deserve attention, without isolating their individual costs.

All six retained final windows contain 1024 distinct input primes and 1024 completed profile sets, with no recorded refusal or timeout in the retained successful validation schedules. The historical reserved cohorts contain 128 and 8 completed distinct primes, and the familiar control 15; their comparison summaries do not provide a complete separate failure ledger. These statements concern the retained cohorts, not every tuning attempt. The selected GPU-preparation path has no automatic dense retry; an unsupported cover or seed refuses the case. A host-preparation fallback for small primes is a different path, not used by the selected timing configuration. Domain guards alone do not promise success for every prime.

7.4 Profiling and unsuccessful variants

Profiling was used diagnostically, in runs separate from the unprofiled timing comparisons. As one example, an attempted shared-memory swizzle raised a measured kernel duration from 59.392 to 68.128 microseconds, shared-load bank conflicts from 1 129 585 to 1 433 934, and registers from 56 to 59 per thread. The issue-active metric increased from 67.36% to 70.90% despite the slowdown. This is one profiled case and one specific variant, not a statement about all swizzles or all inputs.

Other rejected changes included a separately completed boundary seed, a 32-bit-radix wider Montgomery variant, extra wider plan caching, and generic subgroup pairing. In particular, a faster arithmetic microbenchmark did not imply a faster complete scanner. Reporting these negative results limits the interpretation of the retained policy: it is best among the tested

configurations on this machine, not an established optimum.

8 Verification and independence

There are three different levels of evidence: proofs of the mathematical invariants, finite implementation checks, and cross-implementation agreement. They address different failures and must not be conflated.

Geometry and partition tests. Archived tests check 3 469 424 partitioned tasks, 9 988 256 windows and 137 654 606 anchor phases. A separate shared-plan audit covers 10 800 configurations and 103 781 760 reads, including inactive warps and multiple iterations per block. These are finite tests of the scheduling realization. The general coverage argument is Theorem 3.1 and Lemma 4.1, rather than the assumption that the tested character values are representative.

Final engines and reference agreement. The last release configuration agrees with the reference on 6144 fresh primes, covering 20 512 admissible profiles. By order the counts are

r	2	3	4	5	6	7	8	9
profiles	6144	3076	3094	1519	3076	1044	1529	1030.

The comparison checks M_r, L_r, K_r and the canonical witness. The larger study has 8192 validation primes and 27 338 profiles, but the extra 2048 earlier wider cases are not evidence for the final wider specialization. The 209 previously sealed controls cover 867 profiles. Since the reference shares parts of the implementation lineage, this agreement alone is not an independent proof of completeness.

Independent dense scans. The earlier prototype qualification checked 152 new reserved primes with full dense scans, including 16 wider primes. The last study records 25 additional full-oracle calls on 23 distinct primes, up to 8 589 931 561. The dense oracle avoids rational-anchor filtering and uses a different primitive root and natural-residue arithmetic. Its floating-point quotient estimate is corrected using an exact small signed remainder. Section 8.1 proves the actual precomputed-reciprocal algorithm, including low-product wraparound and signed interpretation. CPU checks use 128-bit integer products. Its segment merge retains both boundaries and the least start, so agreement includes runs crossing segment boundaries and the reflected central junction.

This independence targets coverage, label preparation, Montgomery arithmetic and reduction errors. It is stronger than comparing two parameterizations of the same sparse engine. It still certifies only the recorded finite inputs, under the oracle’s own arithmetic proof and implementation checks.

8.1 The oracle’s actual multiplication algorithm

The imported `historical-source/prototype33/dense_oracle.cu` calls `src/math33.hpp::float_mod` in that same snapshot. It uses a precomputed reciprocal, not a division after each product. Its mathematical interpretation is:

1. $v = \text{RN}(1/p)$ (host binary64, round to nearest);
2. $z = \text{RN}(\text{RN}(ab) v)$, $\hat{q} = \lfloor z \rfloor$ (positive binary64 to unsigned-64 conversion);
3. $\rho = ab - \hat{q}p \pmod{2^{64}}$, using unsigned products; interpret ρ as a signed two’s-complement integer r ;
4. if $r < 0$, add p ; if $r \geq p$, subtract p ; return r .

Lemma 8.1 (Corrected reciprocal quotient). *For $0 \leq a, b < p < 2^{33}$, the above algorithm returns $ab \bmod p$, provided both multiplications and reciprocal preparation use correctly rounded binary64 and the stated unsigned/signed conversion semantics.*

Proof. The operands are exactly representable. If both are positive, all floating intermediates are normal and finite. With $\epsilon = 2^{-53}$, three roundings give

$$\left| z - \frac{ab}{p} \right| \leq (3\epsilon + 3\epsilon^2 + \epsilon^3) \frac{ab}{p} < (3\epsilon + 3\epsilon^2 + \epsilon^3) 2^{33} < 2^{-18} < 1.$$

The quotient estimate differs from $\lfloor ab/p \rfloor$ by at most one, so the exact remainder before correction lies in $[-p, 2p)$, well inside $(-2^{63}, 2^{63})$. The unsigned low product difference retains its value modulo 2^{64} even when either full product requires up to 66 bits. Two’s-complement interpretation therefore recovers that exact small signed remainder. The conditional corrections return its representative in $[0, p)$. Zero operands give zero directly. $\square$

The source explicitly multiplies unsigned 64-bit operands and then casts their difference to `int64_t`; it does not overflow a signed product. In C++17, conversion of an out-of-range unsigned value to signed is implementation-defined: this bridge relies on the recorded CUDA/GCC two’s-complement convention. A portable implementation must specify a bit reinterpretation or equivalent conversion rather than silently claim all C++ implementations behave alike. The recorded oracle compilation is `nvcc -O3 -std=c++17 -lineinfo` for SM120, without `-use_fast_math`. No approximate reciprocal instruction is substituted in this proof. Options changing binary64 rounding or arithmetic semantics require a new bridge check. The source hashes and command record are given in the revision source map. The earlier recorded source and the archived oracle snapshot differ only by relocation of the `wide_host.hpp` include; the multiplication function is unchanged. This revision executes no new oracle scan.

Witness and prefix checks. Direct modular-power checks reconstruct the reported interval’s colour and the zero prefix. They confirm that a displayed run exists and that the prefix is correct. A valid witness alone cannot prove that no longer run was missed; that requires coverage or the dense comparison. Conversely, matching the maximum alone does not establish the least canonical witness.

Memory and synchronization tools. The archives contain passing CUDA memcheck, racecheck and synccheck runs and selected-worker memcheck qualifications. Host shared-view checks were run with address and undefined-behaviour sanitizers; leak detection was disabled in the recorded ptrace environment. Sanitizers cover the executions they observe and are not a formal verification of all device schedules.

9 Reproducibility and limits

The frozen packet containing all measured source and data bytes is identified by commit

`f2d515c383776d3f915b87423ea101029d041bc5`.

The selected source ledger is pinned at `f00cdcb95f12d7578bdb37a87db97ff233c86a22`. The article includes a small worked example and separates in-table completion work from character evaluations. Measured source and data snapshots retain their original identities. The accompanying article archive contains the selected reproduction files and their source map. `reproducibility/gpu_method/SOURCE_MAP.json` links engine versions, hashes, qualification records and source functions. The source map distinguishes files included in the accompanying archive from additional upstream material. Upstream commit identifiers establish versions; they do not by themselves establish public accessibility or a release date.

The preceding repository packet separates the editorial source `paper-gpu/` from imported scientific evidence under `results/gpu-algorithms/2026-10-02/`. It retains per-prime medians for the historical comparison, repeated batch times and profile outputs for the later experiment, frozen plans, selections, source snapshots, compilation commands and verification reports. Imported bytes are bound by SHA-256 in `PROVENANCE.json`; their original paths or tar members are recorded. The packet is a compact import of pre-existing archives, not a new benchmark execution.

The local archive has additional historical raw hot samples and exploratory variants. This compact packet does not preserve all of those samples; Table 2 is recomputed from the retained per-prime medians, while Table 3 is recomputed from the retained six repeated batch sums. The measured and rebuilt binaries can have different hashes because of compilation paths or toolchain details; the archive records functional reconstruction checks. A source hash identifies bytes and cannot establish bitwise binary reproducibility on another machine.

Offline checks. The original packet’s offline checker validates all 272 import hashes. The selected reproduction package in the atlas repository checks its own source imports and explicit publication copies, cohort sizes, admissible profile counts, repeated-time statistics and every printed benchmark cell. It makes no CUDA call. The LaTeX source is standalone, with inline bibliography and vector drawing. From the root of the accompanying article archive, run

```
python3 -B scripts/verify_gpu_method.py
```

Its `reproducibility/gpu_method/README.md` distinguishes locally retained measurements and source extracts from the full frozen packet. Source extracts remain byte-identical. Selected JSON publication copies replace historical host paths and omit disk-only metadata; timings, profiles and mathematical values are unchanged and original hashes remain in the provenance map. The full packet separately supplies `paper-gpu/rebuild_workers.py--print-only` for portable build commands. On a separate idle SM120 host with CUDA 13.3, `-execute -smoke` builds the imported workers and compares two archived functional cases with recorded profiles. These GPU commands are supplied for replication and were not executed during this revision. Standalone two-pass `pdflatex` instructions are in the packet README. Binaries rebuilt elsewhere need not have the measured binary hashes; functional agreement on two smoke cases does not reproduce the timing study or inherit its whole qualification.

Comparisons that the evidence does not support. The elapsed weeks of an earlier search and the elapsed hours of a current census differ in scope, overheads, interruptions and hardware history. They suggest a practical change of scale, but do not define a controlled speedup. No timed baseline against Monroe’s implementation is available in these experiments, and no such factor is stated. Threshold-only rejection, one-order scans and short-pattern counting compute different observables from the full M_r, L_r, K_r contract; their attractive timings cannot serve as equivalent full-scan comparisons.

The implementation is qualified on one GPU architecture and compiler configuration, on bounded prime samples. There is no second-device study, energy comparison or guarantee of the same factor over an entire interval. All throughput statements retain their timing boundaries and input domains. Selection-time freezes concern these algorithm experiments; they do not reopen the earlier paper’s closed primary statistical tests. There is no refit of those tests or assertion of a universal law of run maxima.

Short patterns and the output contract. Exact counts of short patterns can sometimes be computed by character sums or elliptic-curve point counts without reconstructing the longest run. The completed research archives contain prototypes for such tasks, including quadratic

quadruplets. They are separate observables with their own arithmetic and certification requirements. The controlled comparisons here concern the full maximum-and-witness contract. They supply no measured speedup for a character-sum or elliptic short-pattern algorithm.

10 Conclusion

Exact sparse scanning is possible because exclusion is justified geometrically and locally: a certified rational cover supplies an anchor, a reflection-compatible schedule hits it, and only observed contradictions can remove it. Exact completion and tie-preserving reduction then reconstruct the profile. GPU scheduling improves how this guaranteed work is executed, while private streams improve throughput across independent primes.

The controlled historical comparison establishes a substantially larger improvement than the final round of optimization alone suggests. Both results remain meaningful when reported with their distinct baselines and aggregation rules. The practical lesson is that algorithmic reduction, arithmetic specialization and hardware scheduling must be evaluated under one explicit output contract. The square-root scale of the tables does not change the demonstrated enumeration bound into a square-root algorithm for the complete maximum.

Declaration of assistance and status. Artificial-intelligence tools assisted with implementation, analysis, literature and source checking, and writing and revision. Brice Pouly is the human author and assumes responsibility for the code, data, references and conclusions. Author-controlled CPU and GPU checks do not constitute independently administered external replication or peer review. The source-access limits of the historical comparisons are stated above.

A Reader’s map of the evidence

This table maps the full upstream packet: paths start at `results/gpu-algorithms/2026-10-02/`, whose `PROVENANCE.json` records their identities. The accompanying article archive retains the selected measurements and source bridges under `reproducibility/gpu_method/`; its `SOURCE_MAP.json` and `README.md` identify those accessible files and distinguish the additional upstream qualification records.

Claim or check	Primary imported record
Historical identity	<code>night/HISTORICAL_ENGINE_IDENTITY.json</code> ; archived adapter and historical source under <code>source/engine32/</code> .
Table 2	<code>night/MESURES_FINALES_32_BITS.json</code> ; inputs in
Table 3	<code>night/FINAL_HOLDOUT.json</code> , freeze in <code>night/FINAL_FREEZE.json</code> .
	<code>throughput/ANALYSE.json</code> ,
	<code>throughput/holdout-32-mr_selected.json</code> ,
	<code>throughput/holdout-33-selected_selected_binary48.json</code> .
Earlier wider reference	<code>throughput/holdout-33-mr_selected.json</code> ; kept separately from the final wider cohort.
Geometry, partition, cache	<code>night/ALL_ANCHOR_LENGTHS_PROOF.json</code> ,
	<code>night/CHUNK_PARAMETER_PROOF.json</code> ,
	<code>night/PARTITIONS_PLANS_VERIFIEES.json</code> ,
	<code>night/SHARED_PLAN_PROOF.json</code> .
Independent checks	<code>night/ORACLES_NOUVEAUX.json</code> ;
	<code>throughput/CONTROLES_INDEPENDANTS.json</code> ; corresponding verification summaries.
Later selections	<code>throughput/PLAN.json</code> , <code>throughput/SELECTION.json</code> ,
	<code>throughput/PLAN_COMPLEMENTAIRE.json</code> ,
	<code>throughput/SELECTION_COMPLEMENTAIRE.json</code> .
Compiler and configurations	<code>source/CONFIGURATION.json</code> , <code>source/worker32.build.json</code> ,
	<code>source/worker33.build.json</code> ; <code>throughput/CODE_IDENTITYES.json</code> .
Profile diagnostic	<code>throughput/ANALYSE.json</code> , field <code>nsight_metrics</code> ; not a batch speedup.

B Implementation bridge and validation versions

Paths start at `results/gpu-algorithms/2026-10-02/`, except `paper-gpu/revision/`. The revision source map stores full SHA-256 hashes, symbols and line locations; hashes identify bytes, not correctness or the date of invention.

Obligation	Implementation bridge	Evidence / limit
Certified fixed cover	<code>clipped_cover.hpp</code> , <code>prime_cover</code> , <code>group_cover</code>	Integer area/domain guards; group cover fixed across its epochs.
Folded complete bands	<code>periodic_bands.hpp::</code> <code>make_shapes</code> , <code>rank/select</code>	Complementary phases, Euclidean division, clipped boundary and translated middle bands.
Total t policy	<code>gpu_pipeline.cuh::anchor_t</code> , <code>epoch_snapshot</code>	Mode 13 fallback; fixed/adaptive update as configured; revision check $q = 3, \dots, 256$.
Coarse/fine union	<code>profile_masks.hpp::split</code> ; <code>coupled_device.hpp::complete</code>	Coarse masks keep coarse completion and fine interior scanning; fine-only residuals retain separate obligations.
Ties and exact extent	<code>completion_core.hpp::complete</code> , <code>take_snapshot</code>	Packed lexicographic reduction, exact completion, zero/domain and epoch guards.
Different dense arithmetic	<code>dense_oracle.cu</code> ; <code>src/math33.hpp::float_mod</code>	Reciprocal lemma and unsigned-low/signed bridge; full scans on recorded inputs only.

Version / domain	Inputs and compared outputs	Authority and independence
Historical adapter, 32 bits	128 natural + 8 stress; 15 familiar. M, L, K .	Historical source/prefix identity and hot-median records; no w output.
Earlier rational prototypes, 32/wider	152 distinct reserved primes, including 16 wider. M, L, K, w .	<code>night/ORACLES_NOUVEAUX.json</code> ; independent complete dense coverage, not final-worker qualification.
Final selected 32 / binary48 wider	6144 distinct fresh primes; 20 512 order profiles. M, L, K, w .	Two final holdout files; same-lineage reference. Order profiles are not distinct independent prime trials.
Preceding wider stream selection	2048 earlier wider primes. M, L, K, w .	Separate wider file; contributes to the 8192-prime whole study, not the final specialization.
Later independent dense checks	25 calls on 23 distinct primes. M, L, K, w .	<code>CONTROLES_INDEPENDANTS.json</code> and final wider controls; different coverage/arithmetic.
Offline v0.2 checks	Anchor policy, source hashes, logged diagnostics, printed tables.	New CPU-only editorial checks; no additional CUDA qualification or full-run oracle count.

References

- [1] J. R. Rabung, Some progression-free partitions constructed using Folkman’s method, *Canadian Mathematical Bulletin* 22(1) (1979), 87–91. doi:10.4153/CMB-1979-012-1.
- [2] D. Monroe, New lower bounds for van der Waerden numbers using distributed computing, *Journal of Combinatorial Mathematics and Combinatorial Computing* 128 (2026), 305–315. doi:10.61091/jcmcc128-19. Publisher PDF dated 4 December 2025; preprint arXiv:1603.03301 (2016; version 7, 2022). <https://arxiv.org/abs/1603.03301>.

- [3] *hmonroe/vdw*, software repository, `uc2.cpp` at commit 267e2b8e09c58af0f7ca9959fedbf9594e87060c (17 February 2019); Git blob ac82f1604958471b7bfab4ff1c8edbd76ffdc73. File attribution: main program by Daniel Monroe, arithmetic routines based on Mark Lotts; repository/commit identity is distinct from paper authorship. <https://github.com/hmonroe/vdw/blob/267e2b8e09c58af0f7ca9959fedbf9594e87060c/uc2.cpp>.
- [4] B. Pouly, *Succinct Rabung certificates and recurrence-closed lower bounds for van der Waerden numbers*, published edition v1.0.0, 2026. doi:10.5281/zenodo.22980965. Scanner source snapshot 704b4125b274bab3df45454adc9c421f8068b8a1. <https://github.com/Vulkin-prog/vdw-rabung-g-bounds/tree/704b4125b274bab3df45454adc9c421f8068b8a1>.
- [5] B. Pouly, *Succinct Rabung certificates and recurrence-closed lower bounds for van der Waerden numbers*, published edition v2.0, 2026. doi:10.5281/zenodo.23337824.
- [6] D. Micciancio, *Minkowski's theorem*, CSE206A: Lattice Algorithms and Applications, UC San Diego, Fall 2025, Corollary 8. <https://cseweb.ucsd.edu/classes/fa25/cse206A-a/Lec2-Minkowski.pdf>.
- [7] P. L. Montgomery, Modular multiplication without trial division, *Mathematics of Computation* 44(170) (1985), 519–521. doi:10.1090/S0025-5718-1985-0777282-X.
- [8] J. P. Sorenson and J. Webster, Strong pseudoprimes to twelve prime bases, *Mathematics of Computation* 86(304) (2017), 985–1003. doi:10.1090/mcom/3134; preprint <https://arxiv.org/abs/1509.00864>.
- [9] NVIDIA, *Blackwell Tuning Guide*, sections on compute capability 12.0, consulted 3 October 2026. <https://docs.nvidia.com/cuda/blackwell-tuning-guide/>.
- [10] W. Beullens, T. Beyne, A. Udovenko and G. Vitto, Cryptanalysis of the Legendre PRF and generalizations, *IACR Transactions on Symmetric Cryptology* 2020(1), 313–330. doi:10.13154/tosc.v2020.i1.313-330. ePrint 2019/1357, revised 30 October 2020. <https://eprint.iacr.org/2019/1357>.
- [11] N. Kaluderović, T. Kleinjung and D. Kostić, *Improved key recovery on the Legendre PRF*, ePrint 2020/098, version 23 June 2021, Sections 3–5 and Appendix A. <https://eprint.iacr.org/2020/098>.
- [12] N. Kaluderović, T. Kleinjung and D. Kostić, Cryptanalysis of the generalised Legendre pseudorandom function, *The Open Book Series* 4(1) (2020), 267–282, ANTS XIV. doi:10.2140/obs.2020.4.267. This separate bibliographic entry is not substituted for the revised ePrint.
- [13] D. A. Buell and R. H. Hudson, On runs of consecutive quadratic residues and quadratic nonresidues, *BIT* 24 (1984), 243–247. doi:10.1007/BF01937490. Publisher abstract and bibliography consulted; full text and cited LSU Computer Science Technical Report 83-017 not obtained.
- [14] D. H. Lehmer and E. Lehmer, On runs of residues, *Proceedings of the American Mathematical Society* 13 (1962), 102–106. doi:10.1090/S0002-9939-1962-0138582-6.
- [15] J. R. Rabung and J. H. Jordan, Consecutive power residues or nonresidues, *Mathematics of Computation* 24 (1970), 737–740. <https://www.cs.umd.edu/~gasarch/TOPICS/res/resRabungJordan.pdf>.
- [16] V. Shoup, *A Computational Introduction to Number Theory and Algebra*, 2nd edition, Cambridge University Press (2008), Sections 4.5–4.6. <https://shoup.net/ntb/ntb-v2.pdf>.
- [17] P. S. Wang, M. J. T. Guy and J. H. Davenport, P-adic reconstruction of rational numbers, *ACM SIGSAM Bulletin* 16(2) (1982), 2–3. doi:10.1145/1089292.1089293. <https://www.cecm.sfu.ca/~monagan/teaching/TopicsinCA15/GuyDavenWang.pdf>.