Union of 2^k Sieves for Accelerating Convergence Verification of the Collatz Conjecture

Gopal Reissenberger

Date: 17 April 2025

Abstract

We present a constant-time residue sieve that strengthens Barina's one-bit $2^{k_{\max}}$ pruning rule for computational verification of Collatz convergence. The baseline sieve marks residues $n_L \bmod 2^{k_{\max}}$ that guarantee a strict descent after $k_{\max}$ iterations of the accelerated Collatz map, allowing the corresponding starting values to be pruned without explicit fallback simulation [1]. Our method replaces this baseline table with a strictly stronger precomputed bitset of the same width, preserving identical runtime cost (one modular index and one memory access) while increasing the density of certified residues. In CPU experiments with $k_{\max} = 20$, the strengthened sieve increases the prune rate from 86.88% to 95.38%, reduces fallback survivors by approximately 65%, and improves throughput by 1.28×–1.32× across dense and anchor sweep benchmarks.

Keywords Collatz Conjecture · Software Optimisation · Computational Verification · Number Theory · High-Performance Computing

1 Introduction

The Collatz conjecture is among the most elementary unsolved problems in mathematics. It concerns the dynamics of the map $T : \mathbb{N} \to \mathbb{N}$ defined by

$$T(n) = \begin{cases} n/2, & n \equiv 0 \pmod 2, \\ (3n+1)/2, & n \equiv 1 \pmod 2, \end{cases}$$

and asks whether for every initial value $n \in \mathbb{N}$, the forward orbit $\{n, T(n), T^2(n), \dots\}$ eventually reaches the trivial cycle $\{1, 2\}$. Despite the simplicity of its definition, the conjecture has resisted all attempts at a complete proof. Mathematician Jeffrey Lagarias famously stated that the Collatz conjecture was "completely out of reach of present-day mathematics" [2].

However, computational verification provides a complementary approach: given a bound B, one checks that every $n \leq B$ reaches a smaller value after finitely many iterations, thereby implying convergence for all values up to B by induction. This programme has been carried out to extremely large limits, most recently by Barina who verified convergence up to 2^{71} by combining distributed GPU computation with several layers of acceleration [1]. At these scales, the dominant cost is not the evaluation of the map T itself, but the number of starting values whose trajectories must be explicitly simulated before a descent can be certified. As a result, large-scale verification depends crucially on *pruning* strategies: fast, sound filters that discard a large fraction of candidates without performing full trajectory simulation.

A particularly effective pruning mechanism arises from a residue-class decomposition. Writing a starting value in the form

$$n = 2^k n_H + n_L, \qquad 0 \le n_L < 2^k,$$

one can express the k-fold iterate $T^k(n)$ in a closed form depending on the low residue n_L and a simple multiplicative factor applied to the high block n_H. This leads to a sieve of width 2^k which marks those residues n_L for which *every* value in the corresponding arithmetic progression is guaranteed to descend below its starting value within k accelerated steps. In Barina's implementation, this criterion becomes a one-bit decision per residue class, yielding a constant-time lookup that prunes the majority of starting values [1].

The purpose of this paper is to strengthen this residue sieve in a way that preserves its essential computational advantages. For a fixed maximum sieve width $2^{k_{\max}}$ (determined by memory constraints and hardware considerations), it is natural to employ the one-bit sieve at $k_{\max}$ as a baseline pruning rule. However, residues that fail the universal descent condition at $k_{\max}$ may still certify descent at a smaller width $k < k_{\max}$, showing that different choices of k capture different parity-vector structure in the accelerated dynamics. We show that this phenomenon can be exploited by composing multiple valid one-bit sieves into a single precomputed table at width $2^{k_{\max}}$. Concretely, given a set of widths $K = \{k_1, \ldots, k_m\}$ with $k_m = k_{\max}$, we construct a *union sieve* that marks a residue class at level $k_{\max}$ as good whenever its projection is good at *any* smaller level. This union sieve is strong and strictly dominates the baseline sieve in pruning power.

2 Barina's One-Bit 2^k Sieve

2.1 Setup and decomposition

For $k \ge 0$, let $T^k(n)$ denote the k-fold iterate of the accelerated Collatz map T, with $T^0(n) = n$. Fix $k \ge 1$ and write

$$n = 2^k n_H + n_L, \qquad 0 \le n_L < 2^k, \quad n_H \ge 0.$$

Let $o_k(n_L)$ denote the number of odd iterates among $n_L, T(n_L), \ldots, T^{k-1}(n_L)$. A key identity used for sieve pruning is

$$T^k(2^k n_H + n_L) = 3^{o_k(n_L)}\, n_H + T^k(n_L), \tag{1}$$

which holds for all $n_H \ge 0$ and $0 \le n_L < 2^k$; see [1]. We say that a starting value n is *certified at level k* if $T^k(n) < n$ [1].

2.2 Good residues and the one-bit criterion

We call a residue $n_L \in [0, 2^k)$ *good at width* 2^k if it certifies a strict descent for every value in its corresponding arithmetic progression, i.e. if

$$T^k(2^k n_H + n_L) < 2^k n_H + n_L \qquad \text{for all } n_H > 0. \tag{2}$$

Using the decomposition (1), the condition (2) is equivalent to

$$3^{o_k(n_L)}\, n_H + T^k(n_L) < 2^k n_H + n_L \qquad \text{for all } n_H > 0. \tag{3}$$

Rearranging (3) yields

$$(2^k - 3^{o_k(n_L)}) n_H > T^k(n_L) - n_L. \tag{4}$$

The inequality (4) makes clear that the behaviour depends on the sign of the coefficient $2^k - 3^{o_k(n_L)}$. If $2^k \leq 3^{o_k(n_L)}$, then the left-hand side is non-positive for all $n_H > 0$ and the universal condition (2) cannot hold in general. In contrast, when

$$2^k > 3^{o_k(n_L)}, \tag{5}$$

the left-hand side of (4) is strictly increasing in n_H, and therefore it suffices to test the smallest admissible block index $n_H = 1$. This yields the following one-bit criterion.

[One-bit sufficient condition] Let $k \geq 1$ and $n_L \in [0, 2^k)$. Suppose $2^k > 3^{o_k(n_L)}$. Then n_L is good in the sense of (2) if and only if

$$3^{o_k(n_L)} + T^k(n_L) < 2^k + n_L. \tag{6}$$

Assume (5). If (2) holds for all $n_H > 0$, then it holds in particular for $n_H = 1$, giving (6). Conversely, suppose (6) holds. Then (3) holds for $n_H = 1$. Since the difference between the right-hand and left-hand sides of (3) is an affine function of n_H with positive slope $2^k - 3^{o_k(n_L)} > 0$, the inequality remains true for all $n_H \geq 1$, proving (2).

Lemma 2.2 shows that, subject to the necessary positivity condition $2^k > 3^{o_k(n_L)}$, each residue class n_L mod 2^k admits a *binary* classification: either it satisfies the single check (6) and is marked good, or it fails and is marked bad. Barina's sieve stores this classification as a bitset good_k of length 2^k, where $\text{good}_k[n_L] = 1$ if n_L is good and $\text{good}_k[n_L] = 0$ otherwise [1].

Algorithm 1 Build one-bit sieve good_k

Input: Width $k \geq 1$
Output: Bitset $\text{good}_k[0, 2^k)$
Initialize $\text{good}_k[r] \leftarrow 0$ for all $r \in [0, 2^k)$
for $r \leftarrow 0$ **to** $2^k - 1$ **do**
    Compute $(T^k(r), o_k(r))$ by iterating T exactly k times
    if $2^k \leq 3^{o_k(r)}$ **then**
        continue; `// cannot be universally good`
    if $3^{o_k(r)} + T^k(r) < 2^k + r$ **then**
        $\text{good}_k[r] \leftarrow 1$
return good_k

2.3 Runtime pruning and fallback simulation

Given a starting value n, the sieve check reduces to reading a single bit at index n mod 2^k. If $\text{good}_k[n \bmod 2^k] = 1$, then n is certified to satisfy $T^k(n) < n$ and can be discarded from explicit simulation [1]. Otherwise, the verification engine performs a *fallback simulation*: it iterates the map T from n until the trajectory descends below its starting value, or until it reaches a value below a previously verified region. In practice, to avoid pathological behaviour during benchmarking, one may cap fallback simulation after a large number of steps $S_{\max}$, treating such cases as survivors for performance accounting.

The effectiveness of the sieve depends on the chosen width 2^k. Larger values of k increase the number of residue classes that can be marked good, but also increase the memory footprint and may incur higher cache and bandwidth costs. In Barina's large-scale verification, the optimal k depends strongly on hardware characteristics, with CPU implementations favouring very large sieve sizes and GPU implementations favouring smaller sizes due to different memory hierarchies and throughput constraints [1].

3 Union-of-2^k Sieves

3.1 Motivation

A one-bit sieve at width $2^{k_{\max}}$ provides a fast, constant-time pruning rule: for each starting value n, one reads a single bit at index $n \bmod 2^{k_{\max}}$ and prunes if the residue is marked good. However, the definition of goodness at a fixed k is stringent: it requires that the strict descent condition holds for *all* high blocks $n_H > 0$ within exactly k accelerated steps. As a result, some residues that fail the universal descent test at $k_{\max}$ may nevertheless certify descent at a smaller width $k < k_{\max}$.

From an algorithmic perspective, different choices of k probe different parity-vector structure of the accelerated dynamics (Barina found $k = 24$ worked best for GPU's, and $k = 34$ for CPU's [1]). Smaller values of k correspond to coarser residue information and can certify descent earlier, while larger values provide finer residue classes but impose a stronger universal condition at the same fixed step depth. This suggests that the strongest pruning under a fixed memory budget may be obtained by combining information from multiple valid sieves rather than relying solely on a single width.

A foundational approach would check several sieves at runtime, pruning whenever any sieve certifies descent. While this can improve prune rate, it requires multiple memory lookups per candidate and can reduce throughput on modern hardware. We therefore seek a construction that combines multiple sieves into a single lookup at width $2^{k_{\max}}$, preserving the constant-time behaviour of the baseline implementation.

3.2 Construction

Let $K = \{k_1, k_2, \ldots, k_m\}$ be a finite set of sieve widths with

$$1 \le k_1 < k_2 < \cdots < k_m = k_{\max}.$$

For each $k \in K$, let good_k denote the one-bit sieve of length 2^k defined in Section 3, where $\mathrm{good}_k[r] = 1$ if the residue $r \in [0, 2^k)$ satisfies the descent criterion at level k, and $\mathrm{good}_k[r] = 0$ otherwise.

We define the *union sieve* at maximum width $k_{\max}$ as a bitset $\mathrm{good}_\cup$ of length $2^{k_{\max}}$ by the rule

$$\mathrm{good}_\cup[r] \;:=\; \bigvee_{k \in K} \mathrm{good}_k\Big(r \bmod 2^k\Big), \qquad r \in [0, 2^{k_{\max}}), \tag{7}$$

where $\vee$ denotes logical OR. Equivalently, $\mathrm{good}_\cup[r] = 1$ if and only if there exists some $k \in K$ for which the projection of r to the low k bits is good under the k-sieve. This yields a single precomputed lookup table indexed modulo $2^{k_{\max}}$.

Runtime behaviour. Given a starting value n, the union sieve performs the check

$$\text{if } \mathrm{good}_\cup[n \bmod 2^{k_{\max}}] = 1 \text{ then prune, else simulate.}$$

Thus the runtime cost is identical to the baseline $k_{\max}$ sieve: a single index computation and a single memory access. The difference is that the bitset encodes pruning certificates from multiple widths simultaneously.

Algorithm 2 Build union sieve $\text{good}_\cup$ at width $2^{k_{\max}}$

Input: Width set $K = \{k_1, \ldots, k_m\}$ with $k_{\max} = \max K$
Output: Bitset $\text{good}_\cup[0, 2^{k_{\max}})$

foreach $k \in K$ **do**
  Build $\text{good}_k[0, 2^k)$ using Algorithm 1
Initialize $\text{good}_\cup[r] \leftarrow 0$ for all $r \in [0, 2^{k_{\max}})$
for $r \leftarrow 0$ **to** $2^{k_{\max}} - 1$ **do**
  foreach $k \in K$ **do**
    if $\text{good}_k[r \bmod 2^k] = 1$ **then**
      $\text{good}_\cup[r] \leftarrow 1$ **break**
return $\text{good}_\cup$

Precomputation cost. The union sieve can be built once by first generating all component sieves good_k for $k \in K$, and then filling $\text{good}_\cup$ using (7). The additional precomputation overhead is linear in the number of widths $|K|$ times the maximum table size $2^{k_{\max}}$, and does not affect runtime verification throughput.

3.3 Correctness and domination

We now record two basic properties of the union sieve: soundness (no false pruning) and domination (never weaker than the baseline).

[Soundness of union pruning] If $\text{good}_\cup[n \bmod 2^{k_{\max}}] = 1$, then there exists some $k \in K$ such that

$$T^k(n) < n.$$

In particular, pruning based on $\text{good}_\cup$ introduces no false certificates.

By definition (7), $\text{good}_\cup[n \bmod 2^{k_{\max}}] = 1$ implies there exists $k \in K$ such that $\text{good}_k[n \bmod 2^k] = 1$. Writing $n = 2^k n_H + n_L$ with $n_L = n \bmod 2^k$, the meaning of $\text{good}_k[n_L] = 1$ is precisely that $T^k(2^k n_H + n_L) < 2^k n_H + n_L$, i.e. $T^k(n) < n$.

[Domination] Let K contain $k_{\max}$. Then for all residues $r \in [0, 2^{k_{\max}})$,

$$\text{good}_{k_{\max}}[r] \le \text{good}_\cup[r].$$

Consequently, the union sieve prunes at least as many starting values as the baseline $k_{\max}$ one-bit sieve.

If $\text{good}_{k_{\max}}[r] = 1$, then taking $k = k_{\max} \in K$ in (7) gives $\text{good}_\cup[r] = 1$. The inequality follows.

3.4 Comparison with adaptive multi-sieve checking

For completeness, we note that an alternative combination strategy is to perform *adaptive* pruning: given n, check $\text{good}_k[n \bmod 2^k]$ for several $k \in K$ and prune if any check succeeds. While this can certify the same set of starting values as a union table built from the same set of widths, it requires multiple memory lookups per candidate. In contrast, the union sieve precomputes this combined decision into a single bitset indexed modulo $2^{k_{\max}}$, preserving the baseline constant-time access pattern (one index computation and one table read). This distinction is important in memory-bound regimes, where throughput is primarily constrained by cache behaviour and memory bandwidth.

4 Correctness

This section formalises the safety of sieve-based pruning and proves that the union-of-sieves construction does not introduce any false certificates. Throughout, T denotes the accelerated Collatz map, and T^k its k-fold iterate.

4.1 Certificates of strict descent

The verification approach considered in this paper relies on *strict descent* below the starting value.

[Descent certificate] Let $n \in \mathbb{N}$ and $k \geq 1$. We say that n is *certified at level* k if

$$T^k(n) < n.$$

The following observation explains why strict descent certificates are sufficient for convergence verification up to a bound.

[Inductive reduction] Fix $B \geq 1$. Suppose that for every n with $1 < n \leq B$ there exists $k \geq 1$ such that $T^k(n) < n$. Then every $n \leq B$ reaches 1 under iteration of T.

Proceed by strong induction on n. The base case $n = 1$ is trivial. Let $2 \leq n \leq B$ and assume the claim holds for all positive integers $< n$. By hypothesis there exists $k \geq 1$ such that $T^k(n) < n$. Since $T^k(n)$ is a positive integer strictly smaller than n, the induction hypothesis implies that the orbit of $T^k(n)$ reaches 1. Therefore the orbit of n also reaches 1.

Lemma 4.1 reduces convergence verification to exhibiting descent certificates for all starting values up to a given bound. In practice, the certificate is either obtained directly from a sieve lookup, or discovered by fallback simulation.

4.2 Correctness of Barina's one-bit sieve

We restate the sieve property in the language of certificates.

[Good residue at width 2^k] Fix $k \geq 1$. A residue $n_L \in [0, 2^k)$ is *good* if

$$T^k(2^k n_H + n_L) < 2^k n_H + n_L \qquad \text{for all } n_H > 0.$$

Equivalently, every integer $n \equiv n_L \pmod{2^k}$ with $n \geq 2^k + n_L$ is certified at level k.

The one-bit sieve good_k stores precisely this predicate. The next proposition records that pruning based on good_k is sound.

[Soundness of one-bit pruning] Let $k \geq 1$ and let good_k be the one-bit sieve. If $\text{good}_k[n \bmod 2^k] = 1$, then $T^k(n) < n$. In particular, pruning a value n when $\text{good}_k[n \bmod 2^k] = 1$ introduces no false certificates.

Write $n = 2^k n_H + n_L$ with $n_L = n \bmod 2^k$ and $n_H \geq 0$. If $\text{good}_k[n_L] = 1$, then n_L is good in the sense of Definition 4.2, and hence $T^k(2^k n_H + n_L) < 2^k n_H + n_L$, i.e. $T^k(n) < n$ [1].

4.3 Correctness of the union sieve

We now prove that the union sieve preserves the same safety guarantees, while strictly extending the set of residues that are pruned.

Recall that the union sieve at width $k_{\max}$ is defined by

$$\text{good}_\cup[r] := \bigvee_{k \in K} \text{good}_k(r \bmod 2^k), \qquad r \in [0, 2^{k_{\max}}),$$

where K is a chosen set of widths with $k_{\max} \in K$.

[Soundness of union pruning] Let K be a set of widths with $k_{\max} \in K$, and let $\text{good}_\cup$ be the corresponding union sieve. If $\text{good}_\cup[n \bmod 2^{k_{\max}}] = 1$, then there exists $k \in K$ such that $T^k(n) < n$. Consequently, union pruning introduces no false certificates.

If $\text{good}_\cup[n \bmod 2^{k_{\max}}] = 1$, then by definition there exists $k \in K$ with $\text{good}_k[n \bmod 2^k] = 1$. Proposition 4.2 applied to that k yields $T^k(n) < n$.

[No loss relative to the baseline] Assume $k_{\max} \in K$. Then for every residue $r \in [0, 2^{k_{\max}})$,

$$\text{good}_{k_{\max}}[r] \leq \text{good}_\cup[r].$$

In particular, the union sieve prunes at least as many starting values as the baseline $k_{\max}$ one-bit sieve.

If $\text{good}_{k_{\max}}[r] = 1$, then the term $k = k_{\max}$ appears in the OR defining $\text{good}_\cup[r]$, so $\text{good}_\cup[r] = 1$.

4.4 Interaction with fallback simulation

In the full verification engine, sieve pruning is used only as a sufficient condition for descent; all values not pruned by the sieve are handled by fallback simulation. The soundness results above imply that all pruned values already possess a valid descent certificate, and therefore skipping them cannot invalidate convergence verification. Any remaining values are explicitly simulated until they reach a smaller value, at which point a descent certificate is also obtained. Thus, combining union pruning with fallback simulation preserves the correctness of the overall verification procedure.

5 Experimental Setup and Methodology

This section describes the experimental framework used to compare Barina's baseline one-bit sieve at a fixed width $2^{k_{\max}}$ against the proposed union sieve at the *same* width. Since both methods reduce pruning to a single bit lookup indexed by $n \bmod 2^{k_{\max}}$, the comparison isolates the effect of increased pruning power rather than changes in asymptotic runtime cost [1].

5.1 Hardware and implementation

All experiments were performed on a CPU system with:

Intel Core i5-13500 (13th Gen), 14 cores, 20 logical processors, base 2.5 GHz.

The benchmark is implemented in `C++`. Each benchmark point is repeated $R = 3$ times and the median runtime is reported.

5.2 Tested ranges

Fix a maximum sieve width $k_{\max}$ and write all tested integers in block form

$$n = 2^{k_{\max}} n_H + n_L, \qquad 0 \leq n_L < 2^{k_{\max}}.$$

For each chosen value of n_H, we test a contiguous segment of N starting values:

$$\{2^{k_{\max}} n_H,\ 2^{k_{\max}} n_H + 1,\ \ldots,\ 2^{k_{\max}} n_H + (N-1)\}.$$

Two sweep schedules are used:

Anchor sweep:

$$n_H \in \left\{ \begin{matrix} 1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128, 192, 256, 384, 512, 768, \\ 1024, 2048, 4096 \end{matrix} \right\}$$

with $N_{\text{anchor}} = 800000$ per block.

This design provides both fine-scale sampling (dense sweep) and coverage across larger scales (anchor sweep), while keeping each block locally contiguous in n_L.

5.3 Baseline sieve

The baseline pruning rule is Barina's one-bit sieve $\text{good}_{k_{\max}}$ at width $2^{k_{\max}}$. Once the lookup table has been generated, runtime pruning proceeds by a single check:

$$\text{prune } n \quad \Longleftrightarrow \quad \text{good}_{k_{\max}}[n \bmod 2^{k_{\max}}] = 1.$$

If the sieve rejects n, simulation is invoked.

5.4 Union sieve construction

Let $K = \{k_1, \ldots, k_m\}$ be a set of widths with $k_m = k_{\max}$. For each $k \in K$ we build the corresponding one-bit sieve good_k. The union sieve is then precomputed as a single bitset of size $2^{k_{\max}}$:

$$\text{good}_{\cup}[r] \;:=\; \bigvee_{k \in K} \text{good}_k[r \bmod 2^k], \qquad r \in [0, 2^{k_{\max}}).$$

After precomputation, runtime pruning again reduces to one lookup:

$$\text{prune } n \quad \Longleftrightarrow \quad \text{good}_{\cup}[n \bmod 2^{k_{\max}}] = 1.$$

Thus the baseline and union methods have identical runtime structure, differing only in which residues are marked good.

5.5 Fallback simulation policy

If a starting value is not pruned by the sieve (baseline or union), it is explicitly simulated under the accelerated map T until a strict descent occurs or a fixed step budget is exceeded. Concretely, the fallback routine iterates T until either:

1. $T^i(n) < n$ for some $i \geq 1$, or
2. $i = S_{\max}$.

The maximum fallback budget is fixed to

$$S_{\max} = 20000$$

accelerated steps. The benchmark records the number of fallback steps used per simulated value and aggregates these across each sweep point.

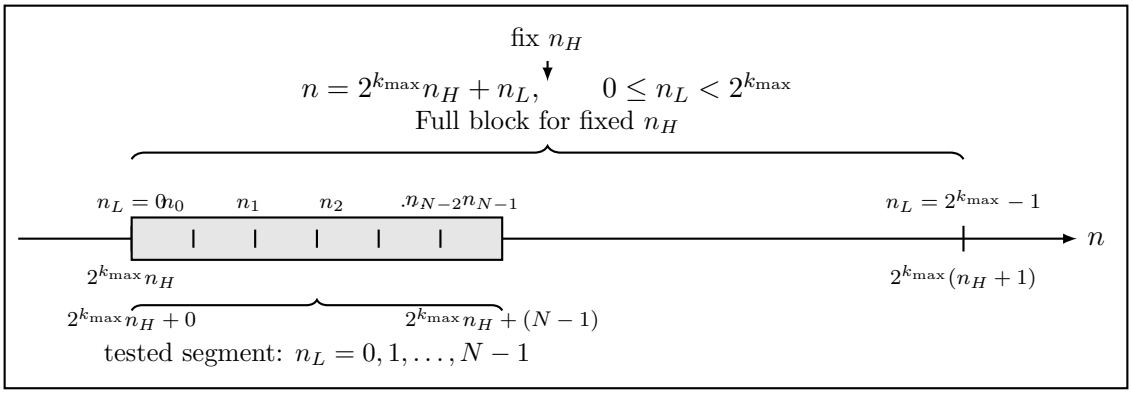

Figure 1: Benchmark sweep structure. For fixed block index n_H, we test values $n = 2^{k_{\max}} n_H + n_L$ by sweeping consecutive residues $n_L = 0, 1, \ldots, N-1$ within the block.

6 Results

All experiments fix the maximum sieve width to $k_{\max} = 20$, so both the baseline one-bit sieve and the proposed union sieve reduce at runtime to a single constant-time lookup in a bitset of length 2^{20}. The methods differ only in how the bitset is populated during precomputation. Performance is evaluated under two sweep regimes: (i) a dense sweep over $n_H \in \{1, \ldots, 256\}$ with $N_{\text{dense}} = 200000$ values tested per block, and (ii) an anchor sweep over sparse n_H values with $N_{\text{anchor}} = 800000$ values tested per block. Each benchmark point reports the median runtime over three repeats.

6.1 Baseline vs. best union sieve at $k_{\max} = 20$

Under $k_{\max} = 20$, the baseline one-bit sieve achieves a prune rate of 86.88% on the dense sweep and 86.85% on the anchor sweep, leaving approximately 13.1% of tested integers to be handled by fallback simulation.

The best-performing union configuration was

$$K = \{8, 10, 12, 14, 16, 18, 20\}.$$

This increases pruning to 95.38% (dense) and 95.34% (anchor), a gain of +8.49 percentage points in both regimes. Equivalently, the fraction of integers requiring fallback simulation decreases from $\approx$ 13.1% to $\approx$ 4.6%, corresponding to an approximate 65% reduction in fallback survivors. Since fallback simulation dominates runtime cost, this reduction translates into a throughput improvement of 1.28× on the dense sweep and 1.32× on the anchor sweep.

Table 1: Baseline one-bit sieve vs. best union sieve at fixed $k_{\max} = 20$. "Fallback survivors" refers to the fraction of tested integers not pruned by the sieve (and therefore requiring fallback simulation).

Method	Prune % (dense)	Prune % (anchor)
Baseline ($K = \{20\}$)	86.88	86.85
Union ($K = \{8, 10, 12, 14, 16, 18, 20\}$)	**95.38**	**95.34**

6.2 Performance across union configurations

To test whether the gains depend on a specific choice of K, several nested unions were benchmarked under the same $k_{\max} = 20$. In general, adding more sieve widths increases pruning and reduces fallback survivors. The configuration $K = \{8, 10, 12, 14, 16, 18, 20\}$ remains the strongest across both sweep regimes.

Table 2: Aggregate performance for $k_{\max} = 20$ under different union configurations (K). Results are weighted by the number of tested integers per block.

K-set	Baseline prune (%)	Union prune (%)	Δ prune (pp)	Survivor drop (%)
8,10,12,14,16,18,20	**86.8830**	**95.3755**	**8.4925**	**64.74**
10,12,14,16,18,20	86.8830	94.0605	7.1775	54.72
12,14,16,18,20	86.8830	92.9495	6.0665	46.25
16,18,20	86.8830	92.1970	5.3140	40.51
14,16,18,20	86.8830	92.1970	5.3140	40.51

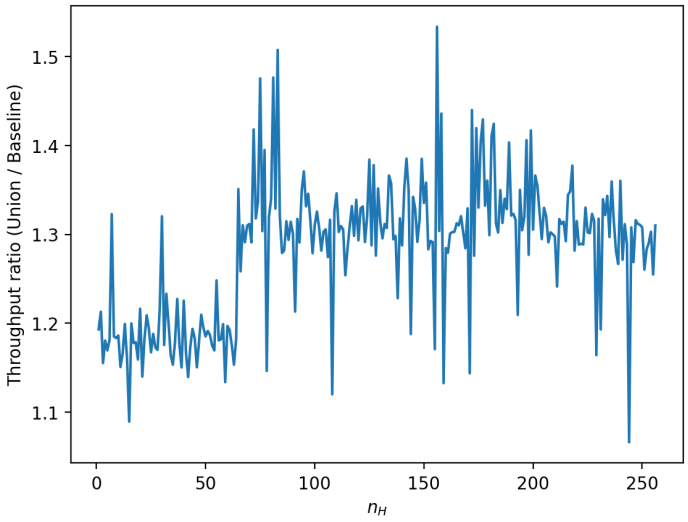

Figure 2: Dense sweep throughput ratio as a function of block index n_H, comparing the baseline one-bit sieve ($K = \{20\}$) against the best union configuration $K = \{8, 10, 12, 14, 16, 18, 20\}$. Ratios above 1 indicate speedup from reduced fallback simulation load.

Figure 2 exhibits noticeable block-to-block variability, but the throughput ratio overall maintains above unity, reaching a peak of above 1.5×.

6.3 Summary of observed gains

At fixed memory footprint ($k_{\max} = 20$), union-of-2^k sieves produces a consistent improvement over the baseline one-bit sieve. Across both dense and anchor sweeps, pruning increases from approximately 86.9% to approximately 95.4%, fallback survivors decrease by approximately 65%, and throughput increases by 1.28× to 1.32× (reaching peaks of 1.5×. These results support the conclusion that multi-width residue certification strengthens sieve pruning while preserving constant-time lookup at runtime.

7 Discussion

7.1 Why the union sieve improves pruning

The union-of-sieves construction strengthens Barina's baseline one-bit sieve by admitting any residue class that certifies a descent at *some* level $k \in K$, rather than requiring universal goodness at a fixed $k_{\max}$. Empirically, this effect is substantial even in the memory-constrained regime $k_{\max} = 20$. In Table 1, the baseline prune rate of 86.88% increases to 95.38% under the full union configuration $K = \{8, 10, 12, 14, 16, 18, 20\}$, corresponding to a gain of 8.4925 percentage points. Interpreted operationally, this reduces the fraction of values requiring fallback simulation from approximately 13.12% to 4.62%, i.e. a 64.74% reduction in simulated survivors.

A useful way to view this result is that different sieve widths capture different parity-vector structure in the accelerated dynamics. A residue class may fail the universal condition at $k_{\max}$ because the affine inequality is not uniformly satisfied for all $n_H > 0$, while still certifying an early descent at a smaller width $k < k_{\max}$. The union sieve recovers these certificates in a way that remains sound and preserves the constant-time lookup structure.

7.2 Throughput gains and the role of fallback simulation

The observed speedups are consistent with the runtime cost structure of Collatz verification engines. A single sieve lookup is negligible compared to explicit fallback simulation, especially when fallback requires iterating T repeatedly until the trajectory descends below the starting value. Consequently, throughput improvements are driven primarily by reducing the number of fallback calls and the total fallback step count, rather than by micro-optimising the sieve lookup itself.

Under the best union configuration $K = \{8, 10, 12, 14, 16, 18, 20\}$, the overall throughput improves by approximately 1.28×–1.32× depending on sweep regime. Figure 2 shows that the speedup persists across the entire dense sweep range $n_H \in \{1, \ldots, 256\}$, with the ratio fluctuating but remaining consistently above unity. This supports the interpretation that union pruning provides a genuine reduction in verification workload rather than a narrow improvement restricted to a small subset of blocks.

7.3 Why the speedup curve is volatile across n_H

The throughput ratio in Figure 2 exhibits visible volatility as a function of n_H. This behaviour is expected for two reasons. First, fallback simulation cost is heavy-tailed: a small number of "difficult" starting values can require disproportionately many iterations before descending, so the per-block runtime can be dominated by rare outliers rather than by typical behaviour. Second, since throughput is computed as a ratio of two noisy measurements (union versus baseline), modest absolute timing variation can produce noticeable oscillations in the ratio even when the underlying work reduction is stable. Importantly, the volatility does not obscure the main trend: the union sieve consistently improves throughput over baseline across the tested range.

7.4 Apparent lack of diminishing returns in the tested K-sets

Across the tested configurations, expanding K by adding smaller sieve widths continued to increase pruning power, and no clear diminishing marginal returns were observed. This is not contradictory to the expectation that union gains should eventually saturate; rather, it simply indicates that the specific tested range of K did not reach the saturation regime.

There are two practical reasons for this. First, the experiment focused on a structured sequence of widths $\{8, 10, 12, 14, 16, 18, 20\}$ that are especially effective at covering multiple parity-vector lengths while maintaining clean power-of-two residue projections. In such a setting, each added width can contribute a non-negligible set of new descent certificates not already captured by larger k, particularly when $k_{\max}$ is relatively small. Second, the study targeted a fixed-memory benchmark regime ($k_{\max} = 20$) and compared only a limited set of configurations for tractability. A more exhaustive search over K (including redundant intermediate widths, irregular spacings, and larger $k_{\max}$) would likely reveal a plateau where additional sieves contribute increasingly few new good residues.

From an engineering perspective, the absence of observed diminishing returns within this tested regime is favourable: it suggests that in small-$k_{\max}$ environments, multi-width composition remains beneficial and does not quickly saturate.

7.5 Limitations and implications for large-scale verification

The present experiments are CPU-only and focus on a memory-constrained sieve width of $k_{\max} = 20$, chosen to keep the union bitset small and cache-friendly. Barina's full verification pipeline operates with much larger sieve widths on CPU and hardware-dependent choices on GPU, where the dominant constraints include memory bandwidth, cache behaviour, and massive parallelism [1]. The union method is structurally compatible with these environments

because it preserves a single-table lookup at runtime, but its performance impact may differ as $k_{\max}$ increases and the sieve transitions from cache-resident to memory-bound.
In distributed verification pipelines, runtime is approximately proportional to the number of candidates tested, so throughput improvements translate directly into faster completion of fixed-size work units. With $k_{\max} = 20$, the union sieve yields a 1.28×–1.32× throughput gain over the baseline sieve, corresponding to roughly 30% more tested values per unit time. Under linear scaling, this would extend a verified range by a factor of 1.3 (an exponent shift of $\log_2(1.3) \approx 0.38$), or reduce time-to-solution by about 23% for the same bound. The practical impact is therefore best interpreted as a consistent efficiency multiplier that can be compounded with larger compute deployments and wider sieve regimes. The main implication is clear: union composition is a strictly sound optimisation that increases pruning density without increasing runtime lookup cost. This makes it an attractive drop-in enhancement for verification engines in regimes where fallback simulation is a bottleneck and sieve width is constrained by memory or cache considerations.

8 Conclusion

We introduced a *union-of-2^k sieves* optimisation for accelerating computational verification of Collatz convergence under a fixed maximum sieve width $2^{k_{\max}}$. The construction forms a single bitset at width $2^{k_{\max}}$ by precomputing a single lookup table that marks a residue as prunable whenever it is certified at any selected width $k \in K$. This preserves the constant-time lookup of Barina's baseline $k_{\max}$-sieve while strictly enlarging the set of residue classes that are certified to satisfy a descent condition [1].

In CPU benchmarks with $k_{\max} = 20$, the best-performing configuration $K = \{8, 10, 12, 14, 16, 18, 20\}$ increases the prune rate from 86.88% to 95.38% (a gain of 8.4925 percentage points). Equivalently, the fraction of candidates requiring fallback simulation drops by 64.74%, and the resulting reduction in simulation workload produces an end-to-end throughput improvement of approximately 1.28×–1.32× across the tested sweep regimes. These results show that multi-width residue certificates can be compressed into a single lookup table that yields substantial practical gains without weakening correctness.

A natural direction for future work is to evaluate the union sieve at larger $k_{\max}$, where cache behaviour and memory bandwidth constraints become more significant, and to test integration into GPU-scale verification pipelines. It would also be valuable to study the marginal contribution of additional widths to K in larger regimes, and to develop principled selection rules for near-optimal K under fixed hardware and memory constraints.

References

[1] David Barina. "Improved verification limit for the convergence of the Collatz conjecture". In: *The Journal of Supercomputing* (2025). DOI: 10.1007/s11227-025-07337-0.

[2] Jeffrey C. Lagarias, ed. *The Ultimate Challenge: The 3x+1 Problem*. Providence, RI: American Mathematical Society, 2010. ISBN: 978-0-8218-4933-8.