A Prime Search That Skipped a Prime

Fixing an OpenMP race condition, re-running a biased benchmark, and testing three prime shortcuts until they broke.

Systems & Number Theory Postmortem · Repositories: `prime-research-v2`, `prime-linear-research`

We started this project trying to speed up multi-threaded probable-prime search on 1,024- to 4,096-bit integers, and to check whether polynomial or binomial identities could filter composites faster than standard Baillie-PSW (BPSW) testing. The first draft of our report claimed a 35% to 52% parallel speedup and floated a complexity-theoretic path toward sub-quadratic primality testing.

An adversarial code audit and a round of counterexample searches overturned both claims. The parallel search had an OpenMP scheduling bug that could skip the true next prime when a strong base-2 pseudoprime sat ahead of it. The 35% to 52% speedup came from comparing a 12-thread shared-tree sieve against a single-threaded baseline that rebuilt its remainder state on every call. And our favorite polynomial filter, which rejected every odd composite up to 5,000, failed at 3,215,031,751.

This report documents the bug, the repaired algorithm, the corrected 12-thread benchmark, and the counterexamples that closed off the mathematical shortcuts.

1. How a base-2 pseudoprime caused OpenMP workers to skip 42,821

Given a starting integer N, `parallel_next_prime` searches upward through odd offsets for the smallest Baillie-PSW probable prime $N + \delta$. Candidates that survive a small-prime wheel and a shared remainder-tree sieve are stored in an ordered array `survivors[0 .. K-1]` and distributed across OpenMP threads. Each candidate undergoes a two-stage BPSW test: a base-2 strong probable-prime (Miller-Rabin) test followed by a strong Lucas probable-prime test.

To avoid wasted Lucas tests once a smaller index looked promising, the original implementation maintained two shared atomic integers: `best_provisional` (set when a candidate passed base-2 Miller-Rabin) and `best_confirmed` (set after the candidate also passed strong Lucas). Before testing index `idx`, each worker checked `if (idx > best_confirmed || idx > best_provisional) continue;`. When the candidate at `best_provisional` is a composite strong base-2 pseudoprime (*SPRP-2*), it passes base-2 Miller-Rabin and fails strong Lucas, but any worker that checks the gate in the meantime abandons its index permanently. At `N = 42,798` (`B = 53`, `W = 64`), four candidates survive the sieve: `42,799`, `42,821`, `42,827`, and `42,829`.

Slot	Candidate	Arithmetic status	Thread action under the buggy gate
`0`	`42,799`	127 × 337 (base-2 pseudoprime)	Worker 0 passes base-2 Miller-Rabin, sets `best_provisional = 0`, then fails strong Lucas.
`1`	`42,821`	**Prime (true next prime)**	**Worker 1 sees `idx 1 > best_provisional (0)` at loop entry and exits without testing.**
`2`	`42,827`	113 × 379 (composite)	Worker 2 sees `idx 2 > best_provisional (0)` at loop entry and exits without testing.
`3`	`42,829`	Prime (returned in error)	Worker 3 entered before Worker 0 wrote `best_provisional`, finishes BPSW, and sets `best_confirmed = 3`.

Table 1. Four-worker execution trace at N = 42,798 (B = 53, W = 64). Because 42,799 is 127 × 337 (a strong base-2 pseudoprime), its temporary provisional bound causes Workers 1 and 2 to abandon slots 1 and 2 while Worker 3 confirms 42,829.

Fixing the gate and verifying the schedule

Removing `best_provisional` entirely would give up early cancellation, while keeping it as an irrevocable skip is unsafe. In `prime-research-v2/src/algorithms.cpp`, we added a per-slot byte array `resolved[K]` initialized to zero. Whenever a worker finishes evaluating slot `idx` (whether it fails base-2 Miller-Rabin, fails strong Lucas, or passes both), it writes `resolved[idx] = 1`. After the OpenMP parallel loop hits its barrier, the calling thread scans every index `0 <= idx < best_confirmed` in increasing order and runs full BPSW on any slot with `resolved[idx] == 0`.

If the lowest-index survivor is prime, `best_confirmed` reaches `0` and the post-barrier repair loop does zero work. If a composite *SPRP-2* temporarily suppresses an intermediate prime slot, the repair pass tests that unresolved slot before returning. We added `tests/scheduled.cpp`, which uses custom synchronization barriers to force the exact four-worker interleaving in Table 1 and verifies that `parallel_next_prime(42798)` returns `42,821`.

Six additional defects caught during the audit

While auditing the concurrency bug, we reviewed the surrounding sieve, modular arithmetic, and certificate code and fixed six other correctness problems:

- **Sieve-only status code:** Calling `test_candidate` with `run_prp = false` returned `DEFINITELY_COMPOSITE` for any survivor that passed the sieve, misclassifying the prime `101`. We added an explicit `UNDETERMINED` status.
- **65-bit Barrett quotient truncation:** In `BarrettNearContext`, near-power-of-two moduli such as $N = 2^{127}$ with $\delta = 2$ overflowed a 64-bit high word during quotient estimation. Retaining the 65th bit fixed the modular reduction.
- **Negative Pocklington targets:** The Pocklington-Lehmer certificate verifier did not check $N > 2$, allowing negative inputs such as `N = -3` to pass verification.
- **Unchecked certificate construction:** The Pocklington builder previously emitted certificates without verifying that the factored part F actually divided $N - 1$ or running self-verification before returning success.
- **Unbounded factoring recursion:** When Pollard rho and trial division failed to split a composite cofactor, `factor_completely` recursed on the same unsplit value until it blew the call stack.
- **Hardcoded wheel primes:** `SegmentSieve` always filtered by primes up to 53 even when the caller requested a smaller sieve bound $B < 53$.

2. Re-running the benchmark against matched 12-thread baselines

Our original benchmark reported that the two-stage search was 35% to 52% faster than a standard GMP baseline. When we inspected the harness, we found two confounding differences: the candidate search ran on 12 OpenMP threads with a prebuilt remainder tree, whereas the baseline ran on a single thread and rebuilt its grouped prime-product remainders inside every call to `next_prime`.

To measure what two-stage cancellation actually contributes, we built three 12-thread baselines in `prime-research-v2/tests/matched_benchmark.cpp` that share the exact same thread count, sieve bounds (B = 65,536 at 1,024 bits; 262,144 at 2,048 bits; 4,194,304 at 4,096 bits), and search windows:

- **Grouped remainder + GMP24:** 12 OpenMP threads using grouped prime-product trial remainders and `mpz_probab_prime_p(..., 24)`. (In GMP 6.3, 24 reps runs a single BPSW pass because the routine loops `reps - 24` additional Miller-Rabin rounds.)
- **Shared tree + Custom BPSW:** 12 OpenMP threads using the same prebuilt `PrimeForest` remainder tree as our algorithm, paired with an atomic `best_idx` cutoff and our custom full BPSW implementation.
- **Shared tree + GMP24:** 12 OpenMP threads using the shared `PrimeForest` remainder tree, an atomic `best_idx` cutoff, and GMP's `mpz_probab_prime_p(..., 24)`.

We generated 12 held-out starting integers per bit width from an independent random seed, and ran all four methods 3 times per input in randomized order on a 12-thread AMD Ryzen 5 4600H (GCC 15.2, GMP 6.3, 432 total measurements).

Bit width	Method (12 OpenMP threads)	Mean (ms)	Median (ms)	Speedup vs. Tree+GMP24
1,024 bits	Grouped remainder + GMP24	4.31	3.10	0.94×
	Shared tree + Custom BPSW	5.14	4.53	0.79×
	Shared tree + GMP24	4.05	3.51	1.00× (baseline)
	Repaired two-stage search	**3.59**	**2.79**	**1.13× [1.05, 1.23]**
2,048 bits	Grouped remainder + GMP24	31.41	30.86	0.94×
	Shared tree + Custom BPSW	33.44	31.00	0.89×
	Shared tree + GMP24	29.63	28.64	1.00× (baseline)
	Repaired two-stage search	**24.85**	**25.37**	**1.19× [1.09, 1.31]**
4,096 bits	Grouped remainder + GMP24	302.21	200.99	0.95×
	Shared tree + Custom BPSW	306.27	203.07	0.94×
	Shared tree + GMP24	288.17	185.72	1.00× (baseline)
	Repaired two-stage search	**261.63**	**160.54**	**1.10× [1.04, 1.19]**

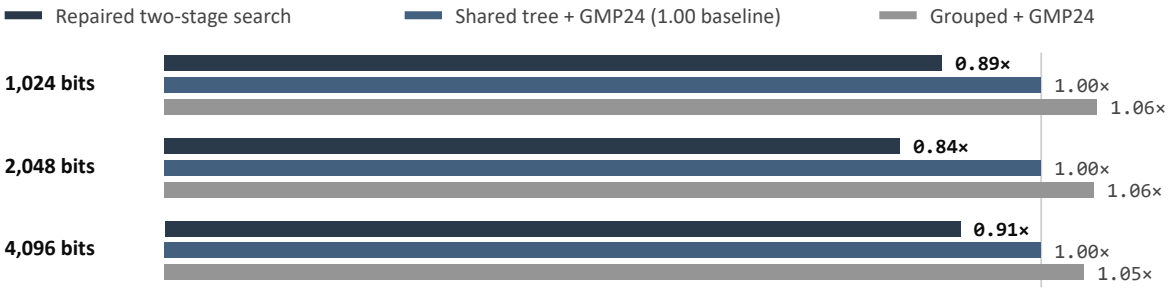

Table 2 & Figure 1. Held-out 12-thread benchmark across 12 inputs × 4 methods × 3 randomized repeats per bit width (432 runs). Bracketed intervals show 95% paired bootstrap confidence intervals for the ratio of mean wall times against Shared tree + GMP24.

What the corrected numbers mean (and their caveats)

Once all four methods get 12 threads and the same remainder tree, the gap narrows from the overstated 35% to 52% claim to a consistent 9.2% to 16.1% reduction in mean wall time (1.10× to 1.19× speedup over Shared tree + GMP24). Two measurement caveats matter when interpreting Table 2:

- **Warm-query vs. cold-start cost:** Table 2 measures steady-state queries after `PrimeForest` is constructed. Building the 4,096-bit remainder tree takes roughly 48.8 ms up front, which pays off across batches of queries rather than one-shot calls.
- **Complete query paths:** The pool baselines construct per-window survivor metadata slightly differently from the integrated two-stage search, so the 1.10× to 1.19× figure reflects the complete query routine rather than Lucas cancellation in isolation.

3. Testing three mathematical prime shortcuts until they broke

Alongside the systems work, we built an executable testbed (`prime-linear-research/experiments.py`) to check whether low-degree polynomial rings, binomial coefficients, or small-prime residue fingerprints could act as ultra-fast deterministic prime filters or reduce the cost of testing record-sized numbers.

The 3-ring polynomial identity fails at 3,215,031,751

The Agrawal-Kayal-Saxena (AKS) theorem proves that N is prime if and only if $(X + a)^N \equiv X^N + a \pmod{N, X^r - 1}$ holds for an order parameter $r = O(\log^5 N)$ and enough linear offsets a. Because working in a ring of degree $\log^5 N$ is much slower than Miller-Rabin, a natural temptation is to fix a tiny set of quotient rings such as r in $\{2, 3, 5\}$ with $a = 1$.

Tested sequentially on all 1,831 odd composites in [3, 5000], the 3-ring check $(X + 1)^N \equiv X^N + 1 \pmod{N, X^r - 1}$ for r in $\{2, 3, 5\}$ produces zero false positives. A test script that only scans small integers makes the 3-ring shortcut look exact. When we fed it known strong pseudoprimes, however, the 14th candidate in our list (`N = 3,215,031,751 = 151 × 751 × 28,351`) passed all three rings as well as strong probable-prime tests to bases 2, 3, 5, and 7.

Why does 3,215,031,751 slip through? In the quotient ring mod $(N, X^r - 1)$ for small r, the polynomial identity only constrains the multiplicative order of N modulo the prime factors of r and cyclotomic cosets of size dividing r. Whenever each prime factor $p \mid N$ satisfies $p \equiv 1 \pmod{\text{lcm}(2, 3, 5)}$ or aligns with the Frobenius action on those three tiny rings, the congruence holds mod every $p \mid N$ and therefore mod N. Here $151 \equiv 751 \equiv 28{,}351 \equiv 1 \pmod{30}$, so multiplication by p acts identically to the prime Frobenius map in all three rings.

Why Pascal-row divisibility is trial division in disguise

A second classic criterion states that N is prime if and only if N divides the binomial coefficient $C(N, k)$ for every $1 <= k <= N - 1$. Since checking all $N - 1$ coefficients is exponential in the bit length of N, we tested whether checking only the first B coefficients could detect composites early.

Across all 4,330 composites in [4, 5000], the first index k where $C(N, k) \neq 0 \pmod{N}$ was always the exact least prime factor of N. That happens because of the elementary identity:

$$C(N, p) = (N / p) \cdot C(N - 1, p - 1) \text{ with } C(N - 1, p - 1) \equiv (-1)^{p-1} \equiv 1 \pmod{p}.$$

When p is the smallest prime dividing N, every integer $1 <= j < p$ is invertible mod N, so N divides $C(N, k)$ for all $k < p$ and fails at $k = p$ because the factor of p in the denominator cancels a factor of p in N. Checking $C(N, k) \bmod N$ for $1 <= k <= B$ therefore catches the exact same composites as trial division up to B, while doing more arithmetic per step.

Residue collisions and the 40-million-digit squaring wall

We also checked whether combining small-prime residues with binary representation features (bit length and Hamming weight) could separate primes from composites without full modular exponentiation. By the Chinese Remainder Theorem, residues mod 2, 3, 5, 7, 11, and 13 only determine N mod 30,030. In our scan, the prime `1,500,007` and the 3-prime composite `1,740,247 = 31 × 73 × 769` share identical residues `[1, 1, 2, 5, 3, 2]`, identical bit length (21 bits), and identical popcount (13 one-bits).

Finally, we estimated the arithmetic wall for testing a 40-million-decimal-digit general integer (roughly $b \approx 1.33 \times 10^8$ bits, comparable to the largest known Mersenne primes). General Fermat or Miller-Rabin testing requires $\Theta(b)$ modular squarings of b-bit numbers. Even with FFT multiplication running at ~0.1 seconds per 133M-bit squaring, executing 1.33×10^8 sequential squarings takes roughly 1.33×10^7 seconds (~154 days) on a single core. Record Mersenne searches avoid this only because $N = 2^p - 1$ has a fixed weight-2 binary structure and an explicit group order structure enabling the Lucas-Lehmer test and fast mod-$(2^p - 1)$ folding.

Proposed shortcut	Small-range behavior	Why it fails in general
Quotient rings $(X+1)^N \equiv X^N + 1 \bmod (N, X^r - 1)$ for r in {2, 3, 5}	0 false positives across all 1,831 odd composites in [3, 5000].	Composite `3,215,031,751 = 151 × 751 × 28,351` passes all three rings (and strong PRP bases 2, 3, 5, 7).
Pascal row divisibility: check C(N, k) ≡ 0 mod N for 1 <= k <= B	Rejects every composite in [4, 5000] at its exact least prime factor k = lpf(N).	Since C(N, p) = (N/p) C(N-1, p-1) with C(N-1, p-1) ≡ 1 mod p, checking k <= B is trial division up to B.
Small-prime residue and bit-shape fingerprint	Filters candidates divisible by 2, 3, 5, 7, 11, or 13.	Prime `1,500,007` and composite `1,740,247 = 31 × 73 × 769` share residues [1, 1, 2, 5, 3, 2], bit length (21), and popcount (13).
40M-digit exponentiation via $\log_2(N)$ squarings	Fast at 1,024 to 4,096 bits.	At 133M bits, 133M modular squarings times ~0.1 s per FFT multiply takes ~154 days on one core.

Table 3. Summary of the four mathematical hypotheses tested in prime-linear-research/experiments.py and the exact counterexamples or complexity bounds that ruled them out.

4. Reproducing the tests and benchmarks

Both research directories in the repository are self-contained and reproducible from the command line:

- **Run the unit, adversarial, and deterministic barrier schedule tests:** in `prime-research-v2`, build with CMake and run `test_correctness`, `regression`, and `scheduled_regression`.
- **Re-run the 432-run held-out 12-thread ablation benchmark:** run `matched_benchmark` in `prime-research-v2`, which writes raw timings to `results/holdout-ablation.csv` and paired bootstrap intervals to `results/summary.txt`.
- **Re-generate the mathematical counterexamples:** run `python prime-linear-research/experiments.py`, which reproduces the `3,215,031,751` 3-ring failure, the Pascal-row least-prime-factor identity, and the `1,500,007` / `1,740,247` collision.