Report: CompPoly Evaluation Optimizations

May 25, 2026

Formal Land

Contents

1 Summary

Main contributor: Valerii Huhnin, Formal Land

Here is the description of the grant target from the roadmap:

> The goal of this milestone is to optimize the evaluation of polynomials.
> For that, we must implement an efficient evaluation version for each of the polynomials representation for:
> - The Horner method
> - The evaluation of a batch of random points, when it can be optimized
> - The evaluation of a batch of points generated for the NTT
>
> We do not plan to focus on the evaluation of a batch of polynomials at one point. It can be added if needed.
> The implementations must come with a Lean proof of equivalence with the naive implementation, and a set of tests and profiling showing increased performance.

We completed the milestone in the upstream CompPoly repository. The work was implemented by Valerii Huhnin from the Formal Land team and merged into `Verified-zkEVM/CompPoly`.

The delivered work covers all three requested optimization areas:

- Horner-style evaluation for univariate, bivariate, multivariate, and multilinear polynomial representations.
- Efficient batch evaluation of one univariate polynomial on many points, using a subproduct-tree algorithm and faster monic modular reduction.
- Faster NTT-backed evaluation and multiplication paths, including both additive NTT and root-of-unity NTT pipelines.

Each implementation was accompanied by Lean correctness lemmas or proofs connecting the optimized code back to the existing naive or specification-level implementation. The milestone also delivered and then refactored a benchmark suite that is integrated into GitHub Actions, emits JSON and Markdown reports, and is used to compare naive and optimized variants.

We also completed an additional optimization that the roadmap listed as optional: evaluation of many polynomials at one point. This was merged in PR 230 on 2026-05-22.

2 What we get from it

With this work we hope to make CompPoly more useful as an implementation library for large polynomials in Lean, in addition to being a specification library:

- Polynomial evaluation code now has executable optimized variants that can be used directly by downstream algorithms.
- Optimized code paths are justified by Lean proofs of equivalence with simpler definitions.
- The benchmark suite gives maintainers a way to detect performance regressions and measure future improvements.
- The NTT and field improvements accelerate higher-level algorithms such as multiplication, low-product multiplication, monic remainders, and subproduct-tree evaluation.
- We focused on maintainability, by not merging work that would have been too complex to maintain compared with its observed impact.

3 Merged pull requests

All pull requests listed below are merged in `Verified-zkEVM/CompPoly`.

- PR 208, merged 2026-05-06: *bench: add polynomial evaluation benchmarking*. Initial benchmark executable, CI integration, JSONL/Markdown output, and benchmark artifacts.
- PR 209, merged 2026-05-07: *feat(eval): add Horner evaluation for polynomial representations*. Horner evaluation for univariate, bivariate, multivariate, and multilinear polynomials, plus MLE evaluation and equivalence lemmas.
- PR 210, merged 2026-05-13: *perf(additive-ntt): verified faster implementation*. Faster array-backed additive NTT with equivalence proofs and benchmarks.
- PR 213, merged 2026-05-18: *feat(univariate): efficient evaluation on a batch of points using subproduct-tree algorithm*. Subproduct-tree batch evaluation, faster `modByMonic` variants, multiplication/remainder contexts, and correctness files.
- PR 215, merged 2026-05-15: *feat(ntt): add `BabyBear` domain and benchmark NTT with Baby and Koala bears*. NTT benchmark coverage for BabyBear and KoalaBear domains.
- PR 220, merged 2026-05-19: *perf(ntt): faster NTT implementation*. `NTTFast` pipeline with cached plans, optimized transforms, correctness proofs, and downstream benchmark improvements.
- PR 223, merged 2026-05-20: *feat(bench): various enhancements*. Refactored benchmark suite, grouped reports, CLI filters, output modes, and size presets.
- PR 228, merged 2026-05-21: *feat(fields): fast KoalaBear field*. Fast KoalaBear implementation using machine integers and Montgomery representation, with equivalence proof.
- PR 230, merged 2026-05-22: *feat(univariate, multilinear): faster implementations of evaluation of many polynomials at one point*. Shared-power many-polynomial univariate evaluation and faster many-polynomial multilinear evaluation.

4 Details

4.1 Benchmark suite

PR 208 introduced a `CompPolyBench` Lake executable under `bench/`. It records warmup and measured iteration counts, elapsed nanoseconds, average time, checksums, and basic runner metadata. It writes JSONL and Markdown reports, appends Markdown reports to the GitHub Actions summary, and uploads benchmark outputs as an artifact.

PR 215 extended benchmark coverage to NTT-based multiplication over BabyBear and KoalaBear. PR 223 then split the benchmark code into smaller files mirroring the source layout, added grouped reports, printed progress, and introduced CLI options such as `--list`, group selection, `--markdown-only`, `--json-only`, and `--small`/`--medium`/`--large` presets. CI is configured to use the medium preset, which the PR body reports as taking around 7 minutes.

4.2 Horner and multilinear extension evaluation

PR 209 added optimized evaluation functions across the main polynomial representations:

- Public Horner wrappers for `CPolynomial.Raw`, with equivalence lemmas.
- Bivariate Horner evaluation in two evaluation orders: evaluate in `Y` then `X`, or evaluate `X` coefficient polynomials first and then evaluate in `Y`.

- Coefficient-form multilinear polynomial evaluation by Horner, with a lemma showing equivalence to `eval`.
- Hypercube-value multilinear extension evaluation, with a lemma showing `evalMle` agrees with `eval`.
- Multivariate Horner evaluation, with correctness lemmas in the `HornerLemmas.lean` file.

This directly covers the roadmap requirement for Horner evaluation across polynomial representations.

4.3 Batch evaluation on many points

PR 213 added subproduct-tree batch evaluation for univariate polynomials. Pointwise Horner evaluation of a degree n polynomial at m points has complexity $O(nm)$. The subproduct-tree algorithm builds a tree of products for the m linear factors and then descends the tree by reducing the input polynomial modulo subtree products. With fast multiplication and fast monic modular reduction, the PR body describes the expected complexity as roughly $O((n+m)\log(n+m)\log m)$.

The implementation is parameterized by multiplication and monic-reduction contexts. This lets CompPoly select naive or optimized backends while keeping a total, verified interface. The PR also improved `modByMonic` with:

- a remainder-only long-division variant that avoids quotient construction and unnecessary polynomial multiplications;
- a reversal-based monic remainder algorithm using inverse modulo X^k;
- low-product multiplication contexts used by the reversal-based algorithm.

The PR reports that subproduct-tree evaluation is slower on small examples, but faster on sufficiently large examples. In its CI benchmark, a dense degree < 65536 polynomial evaluated at 8192 points went from 25.60 seconds with pointwise Horner to 13.59 seconds with subproduct-tree evaluation using NTT multiplication and reversal-based monic remainder.

4.4 Faster NTT and additive NTT

PR 210 optimized additive NTT. The previous implementation represented stages as functions, which could recompute shared intermediate values. The new `computableAdditiveNTTFast` implementation uses arrays, a recurrence for subspace vanishing polynomial evaluation, and precomputed twiddle factors. The PR moved proofs to `Correctness.lean` and added equivalence lemmas connecting the fast implementation to `computableAdditiveNTT` and `additiveNTT`.

The PR body reports these CI benchmark improvements:

- `additive-ntt-btf3`: average time from 16.77 ms to 1.34 ms.
- `additive-ntt-btf3-l4-r2`: average time from 1.51 s to 8.12 ms.

PR 220 added a faster root-of-unity NTT multiplication pipeline under `CompPoly.Univariate.NTTFast`. It includes cached `NTTFast.Plan` data, a one-shot multiplication wrapper, low-product and multiplication contexts, and correctness proofs connecting the optimized executable path to the existing NTT specification and ordinary multiplication.

The optimized NTT path:

- caches inverse domains, domain-size inverses, and per-stage twiddle tables;
- uses forward DIF transforms producing bit-reversed evaluation order;
- consumes that order with a compatible inverse DIT transform, avoiding standalone bit-reversal passes;

- fuses adjacent radix-2 stages into mixed radix-4 passes where possible;
- runs two forward transforms together in the multiplication path;
- uses explicit recursive hot loops over butterfly indices.

The PR body reports degree < 1024 multiplication improvements of 13.49 ms to 3.75 ms for BabyBear and 13.36 ms to 3.78 ms for KoalaBear. With a reused plan, the times are 3.49 ms and 3.50 ms respectively. The same PR reports downstream improvements, including subproduct-tree large evaluation with reversal modular reduction going from 14.93 seconds with the baseline NTT backend to 3.78 seconds with the `NTTFast` backend.

4.5 Fast KoalaBear field

PR 228 added a fast KoalaBear field implementation using machine integers and Montgomery representation, plus a proof of equivalence between the fast and basic KoalaBear fields. The benchmark suite was updated so KoalaBear is the default field and both basic and fast versions are tested.

The PR body reports that algorithms using the faster field are about 4x faster on average, with specific examples of about 10x faster univariate Horner evaluation and about 3x faster univariate multiplication using fast NTT.

4.6 Evaluation of many polynomials at one point

Additionally, we added an optimization of evaluation of many polynomials at one point with PR 230:

- shared-power / dot-product based evaluation of many univariate polynomials at one point;
- faster evaluation of many multilinear polynomials at one point;
- correctness proofs establishing equivalence with naive evaluation;
- benchmarks comparing naive and optimized versions.

The PR body reports that shared-power univariate evaluation is about 2x faster on optimized KoalaBear and roughly the same or 10 percent slower on the unoptimized field. For multilinear polynomials, the optimized implementation is about 50 percent faster on optimized KoalaBear and about 30 percent faster on the unoptimized field.

5 Left out

Two investigated optimizations were not included:

- Specialized code for the 3-coset shape. $\mathcal{V}$alerii found that it is technically possible to save some computation, but not in the stage that is the bottleneck. Because the specialization would add complexity and maintenance burden without meaningful end-to-end improvement, it was left out.
- A transposed memory layout for evaluating many polynomials. For many polynomials, the coefficient matrix appears to benefit from transposition because of better cache utilization compared with an array-of-polynomials representation. However, this would require a specialized API and additional complexity, so it was not merged in this milestone.

These decisions were made to keep the verified library maintainable: optimization code was added where the benchmark and proof story justified it, and omitted where the complexity/performance tradeoff was weak.

6 Files of interest

The main files and folders touched by the milestone are:

- `bench/`: benchmark runner, grouped benchmarks, JSON/Markdown report generation, and benchmark documentation.
- `CompPoly/Univariate/Basic.lean`: univariate evaluation APIs.
- `CompPoly/Bivariate/Basic.lean` and `CompPoly/Bivariate/ToPoly.lean`: bivariate evaluation and correctness.
- `CompPoly/Multilinear/Basic.lean`: multilinear evaluation and MLE evaluation.
- `CompPoly/Multivariate/CMvPolynomial.lean`: multivariate evaluation implementation.
- `CompPoly/Multivariate/HornerLemmas.lean`: multivariate Horner correctness lemmas.
- `CompPoly/Univariate/BatchEval/`: subproduct-tree batch evaluation, contexts, and correctness.
- `CompPoly/Fields/Binary/AdditiveNTT/`: additive NTT implementation and correctness.
- `CompPoly/Univariate/NTTFast/`: fast root-of-unity NTT pipeline and proofs.
- `CompPoly/Fields/KoalaBear/`: basic and fast KoalaBear field implementations.
- `CompPoly/Univariate/ManyEval/`: many-polynomial univariate evaluation optimizations and correctness.
- `CompPoly/Multilinear/ManyEval/`: many-polynomial multilinear evaluation optimizations and correctness.

7 Thanks

We thank the Ethereum Foundation and the Verified-zkEVM maintainers for the opportunity to contribute these verified optimizations to CompPoly. We are happy to discuss any details in this report and the project.