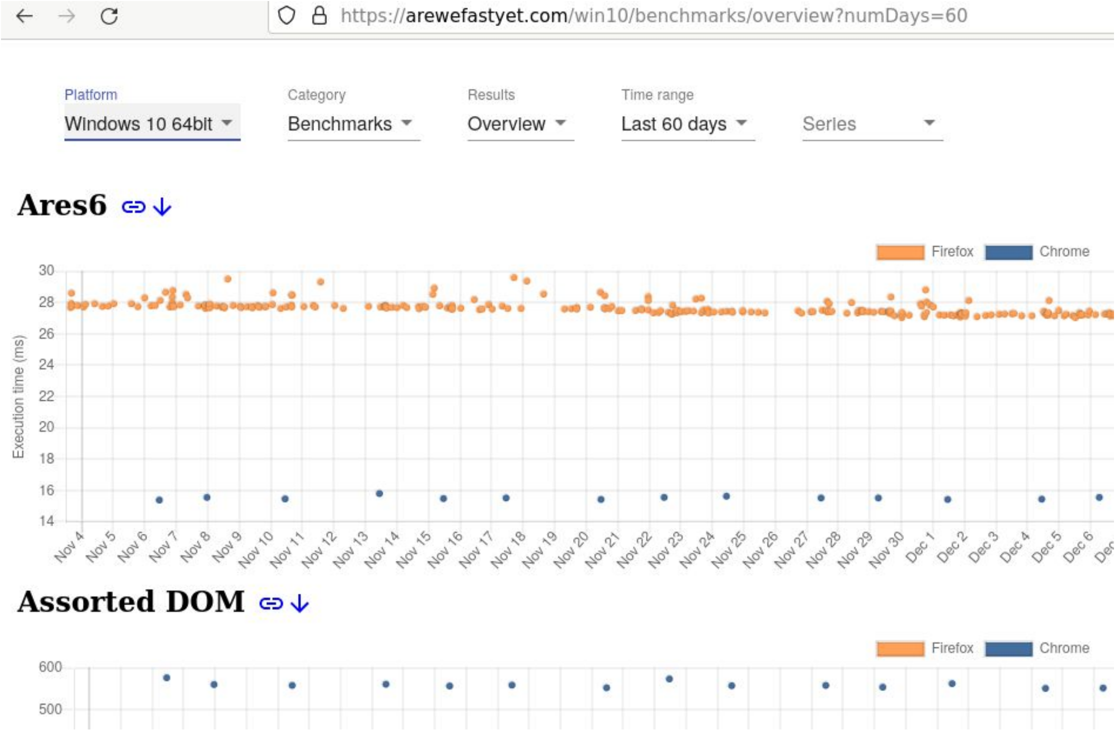
Are We Fast Yet

Sebastian Ullrich (Lean FRO)

LEAN
THEOREM PROVER

lean-lang.org/publications/

Inspiration

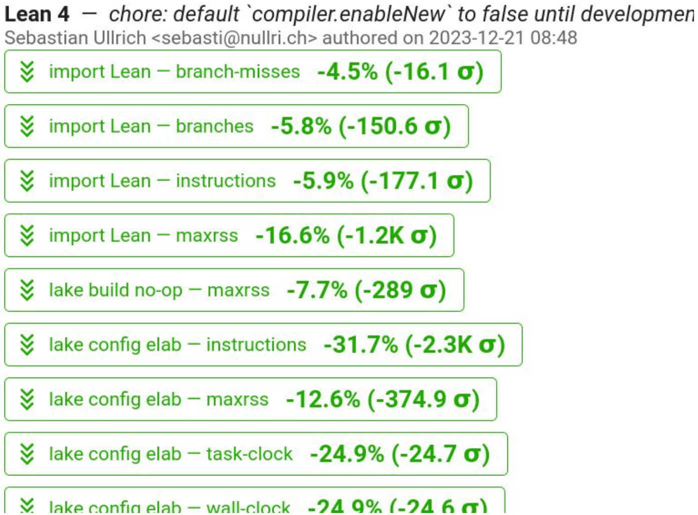


What to Measure

1. Full rebuild
   - easy to measure, continously
   - CI worst-case
   - time (+ parallelism), memory, disk
2. Incremental build
   - local/CI common case
   - average over last N commits?
3. UI latency of specific action
   - load Mathlib, edit single proof, …
   - specific benchmark per action

Status Quo

Continuous benchmarking of each lean4 & mathlib4 commit



Lean 4 — *chore: default `compiler.enableNew` to false until developmen*
Sebastian Ullrich <sebasti@nullri.ch> authored on 2023-12-21 08:48

⇊ import Lean — branch-misses **-4.5% (-16.1 σ)**

⇊ import Lean — branches **-5.8% (-150.6 σ)**

⇊ import Lean — instructions **-5.9% (-177.1 σ)**

⇊ import Lean — maxrss **-16.6% (-1.2K σ)**

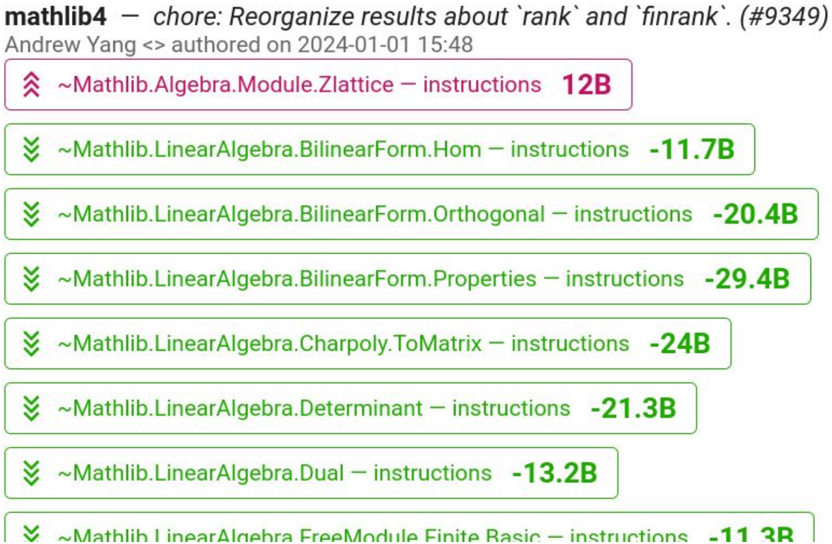
⇊ lake build no-op — maxrss **-7.7% (-289 σ)**

⇊ lake config elab — instructions **-31.7% (-2.3K σ)**

⇊ lake config elab — maxrss **-12.6% (-374.9 σ)**

⇊ lake config elab — task-clock **-24.9% (-24.7 σ)**

⇊ lake config elab — wall-clock **-24.9% (-24.6 σ)**



mathlib4 — *chore: Reorganize results about `rank` and `finrank`. (#9349)*
Andrew Yang <> authored on 2024-01-01 15:48

⇈ ~Mathlib.Algebra.Module.Zlattice — instructions **12B**

⇊ ~Mathlib.LinearAlgebra.BilinearForm.Hom — instructions **-11.7B**

⇊ ~Mathlib.LinearAlgebra.BilinearForm.Orthogonal — instructions **-20.4B**

⇊ ~Mathlib.LinearAlgebra.BilinearForm.Properties — instructions **-29.4B**

⇊ ~Mathlib.LinearAlgebra.Charpoly.ToMatrix — instructions **-24B**

⇊ ~Mathlib.LinearAlgebra.Determinant — instructions **-21.3B**

⇊ ~Mathlib.LinearAlgebra.Dual — instructions **-13.2B**

⇊ ~Mathlib.LinearAlgebra.FreeModule.Finite.Basic — instructions **-11.3B**

speed.lean-lang.org

Status Quo

Continuous benchmarking of each lean4 & mathlib4 commit

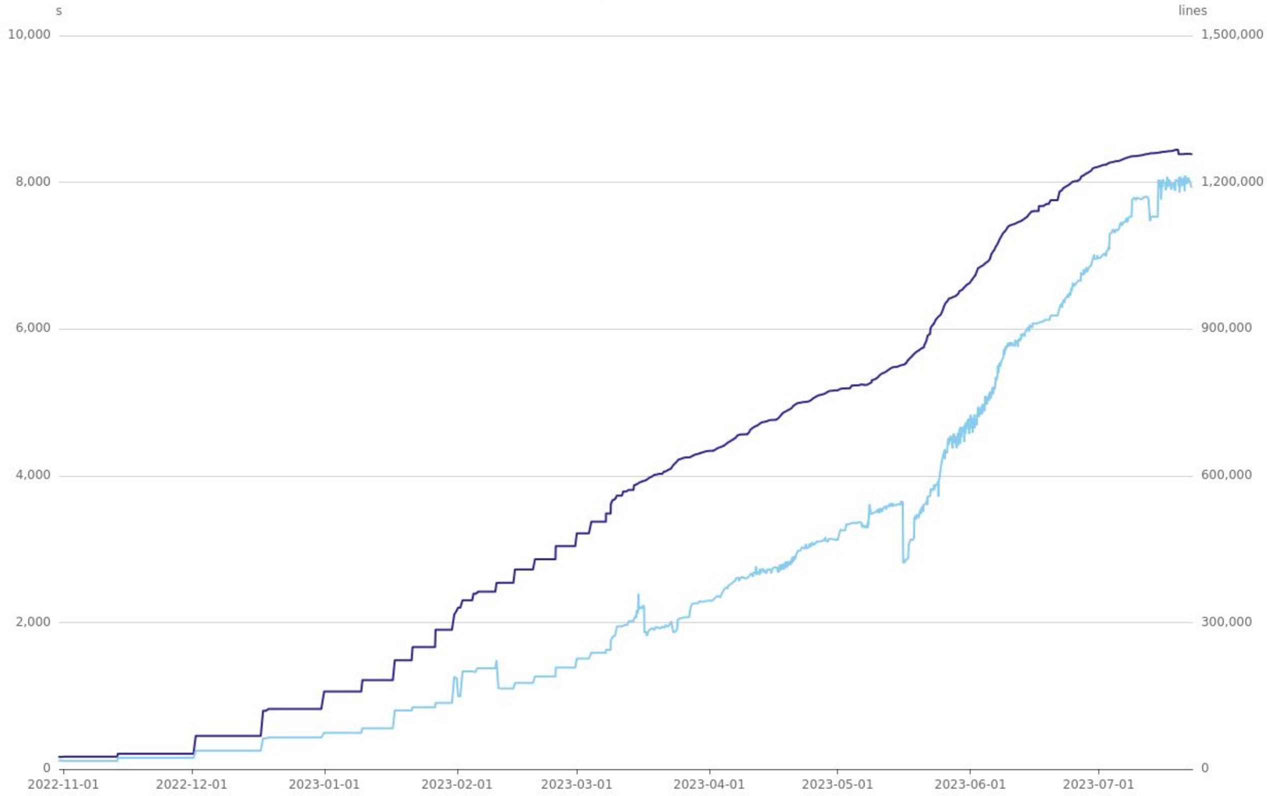
- full build with `--profile`
- compiler microbenchmarks
- `import Lean`, `lake env`, `match` reduction, workspace symbols

- full build with `--profile`
- per-file instruction counts (8B instrs ≈ 1s)
- `import Mathlib`

Benchmarking Pipeline

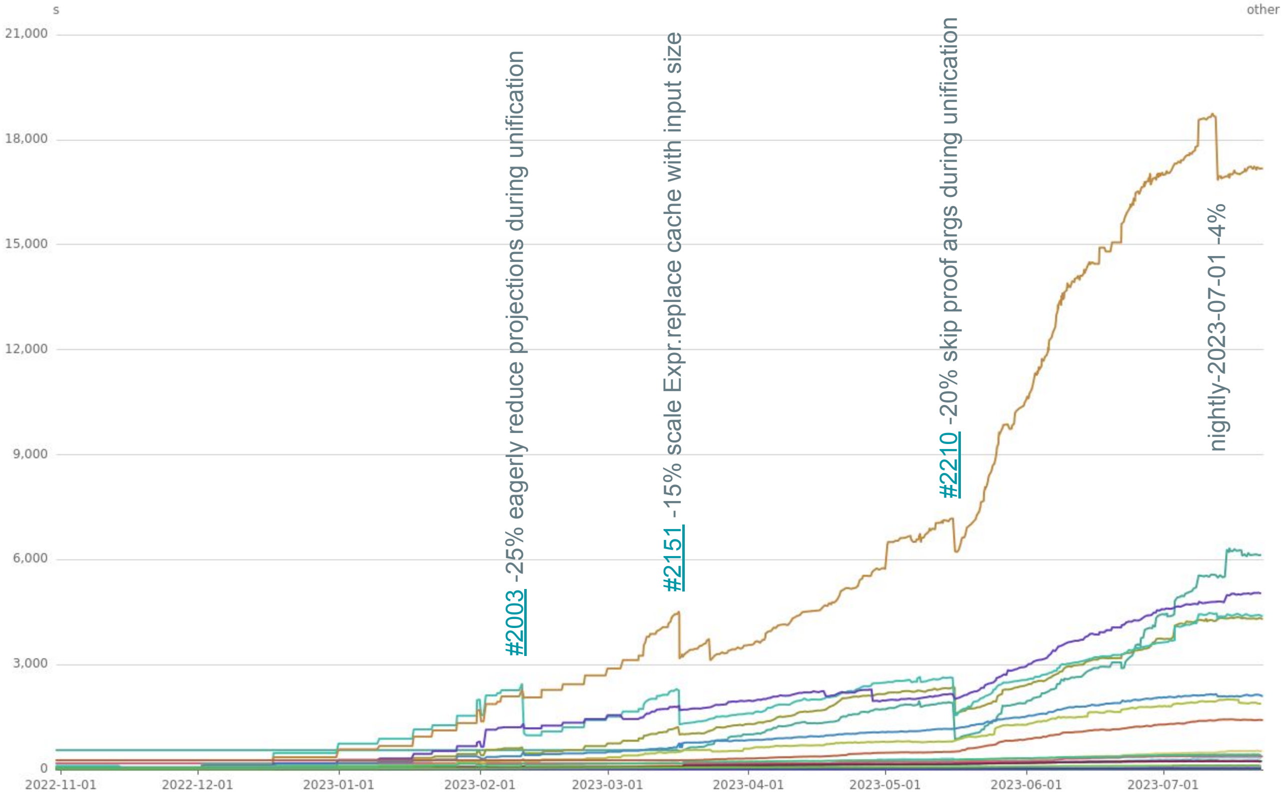
1. Temci, an unmaintained benchmarking cmdline tool (Python)
    - lean4/tests/bench/speedcenter.exec.velcom.yaml
    - mathlib4/scripts/bench/temci-config.run.yml
2. VelCom, an unmaintained benchmark runner & visualizer (Java/TypeScript)
3. A bit of shell glue code
    - Kha/lean-bench
    - Kha/mathlib4-bench

The Mathlib Port: lines & build time



speed.lean-fro.org/mathlib4

The Mathlib Port: Breakdown into Categories



#2003 -25% eagerly reduce projections during unification

#2151 -15% scale Expr.replace cache with input size

#2210 -20% skip proof args during unification

nightly-2023-07-01 -4%

45% typeclass inference

16% other tactics

13% interpretation

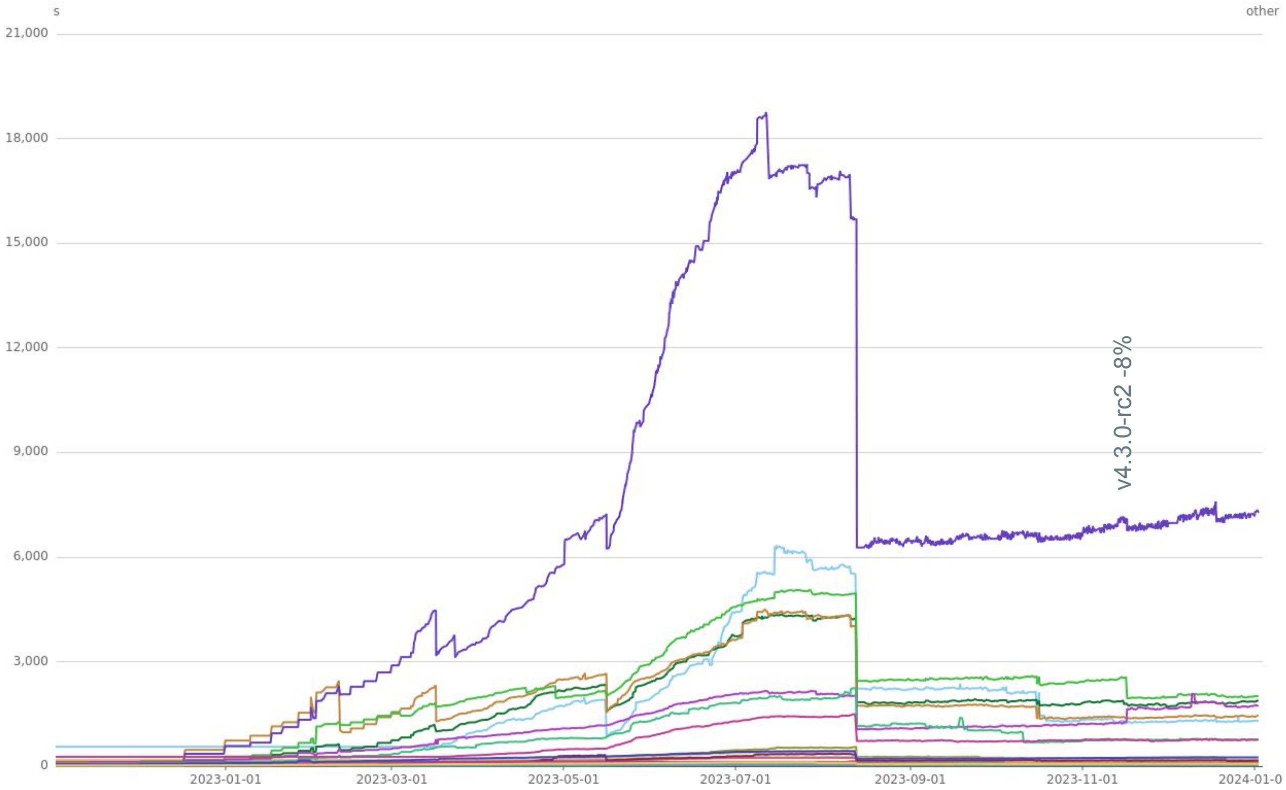
11% simplifier, other elab

5% import, compilation

4% kernel

0.2% parsing

State Jan 2024



i5-8500 | Ryzen 9 7950X3D

Performance: Before (Lean 3) and After (Lean 4)

On a Ryzen 9 (32 threads):

Total build time: 48 min ~> 21 min (-55%)
Single-core time: 23 hours ~> 5 hours (-77%)
Typeclass inference: 3 hours ~> 1 hour 46 min (-42%)

Performance: Importing Mathlib

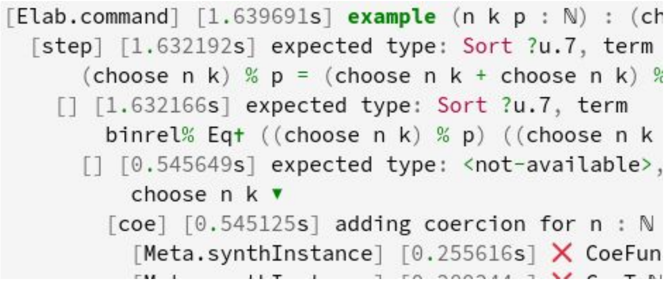
disk: 436 MB ~> 3.1 GB (+711%)
time: 10.6 s ~> 1.5 s (-86%)
allocations: 4.6 GB ~> 243 MB (-95%)

due to zero-cost deserialization via memory mapping

3.1 GB disk compressed down to 200 MB on the wire via digama0/leangz

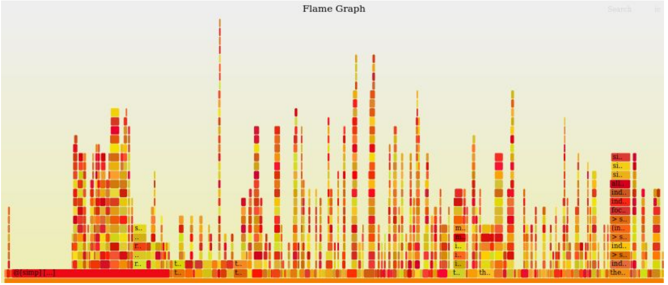
New profiling tools

`trace.profiler` prints a *structured* profile



```
[Elab.command] [1.639691s] example (n k p : ℕ) : (ch
  [step] [1.632192s] expected type: Sort ?u.7, term
      (choose n k) % p = (choose n k + choose n k) %
    [] [1.632166s] expected type: Sort ?u.7, term
        binrel% Eq† ((choose n k) % p) ((choose n k
      [] [0.545649s] expected type: <not-available>,
          choose n k ▼
        [coe] [0.545125s] adding coercion for n : ℕ
          [Meta.synthInstance] [0.255616s] ❌ CoeFun
```

[hargoniX/Flame](#) converts it into a *flame graph*



On the FRO Roadmap

1. Full rebuild
   - #5 Elaboration efficiency, especially parallelism
2. Incremental build
   - #9 Reservoir Package Registry
   - #11 Module System
3. UI latency of specific action
   - #5 Elaboration efficiency: incrementality

lean-fro.org/about/roadmap

[demo]

Conclusion

We have a solid benchmarking setup in some areas, less so in others. Help welcome!

The move to Lean 4 has yielded unprecedented performance improvements, and we will continue to work both on improvements to existing components and on exciting new possibilities.