Hunting Clang Compile-Time Regressions — Midterm update

Ezzeldin Ibrahim (AnonMiraj)

Where we left off

- **One** confirmed regression pinned, zero fixes upstream yet
- The bottleneck: **~45 minutes to build Clang per commit**, 135+ builds on my poor laptop

The scoreboard

15+ regressions pinned to exact first-bad commits, across **8 codebases** (Boost.MPL, Hana, Spirit X3, range-v3, fmt, SQLite, the Linux kernel …), turned into **8 upstream PRs — 6 already merged**.

The PRs so far:

PR	fix	local benchmarks	LLVM tracker	status
#202727	APINotes: early return when no apinotes files are loaded	undoes a +0.4% tax	**-0.11%**	merged
#203710	APINotes: skip per-decl work when no notes are active	(follow-up to above)	**-0.07%**	merged
#204926	only record lexer check-points when colors are enabled	part of a +0.3% step	+0.07%	merged
#209355	fix check-point recording under default colors	fixes a regression **#204926 caused**	**-0.20%**	merged
#208862	skip macro source-location check for names that can't warn	-0.7% kernel, -0.3% MPL	~0%	merged
#208139	compute current instantiation less often in lookupInBases	~99% of a +0.4% step	**-0.03%**	merged
#209694	no typo-correction suggestions for disabled diagnostics	**-1.2% MPL**, -0.2% kernel	**-0.28%**	open
#206363	don't add doc comments to the AST if not requested	-5% on comment-heavy TUs	**-0.24%**	open

tracker = llvm-compile-time-tracker.com, geomean instructions:u

What that adds up to

- On the template-heavy MPL basket, the fixes recover **~1.5%** of frontend time. The whole 18→trunk regression was **+2.3%**. That's most of the fixable half — the rest is the price of some features
- On the **LLVM compile-time tracker** merged PRs net **≈ -0.35%** on the frontend-heavy config the two open PRs add another **≈ -0.5%**, for **≈ -0.9%** total
- The pattern in almost every fix: Clang doing **expensive work for a feature/diagnostic it was never used**.

manyclangs

Remember the 45-minute builds? I found the **manyclangs** project: pre-built Clang binaries for every LLVM commit since 2019, stored as **elfshaker** packs (~150 MB per month of commits).

Before

- build Clang at each commit: **~45 min**
- 9-commit coarse pass: **days**
- laptop unusable while building

After

- `elfshaker extract`: **~2 seconds**
- 9-commit coarse pass: **minutes**
- 30+ bisect passes ran in weeks

The catch

manyclangs binaries are built **with assertions on**. Assertion-only code can fake a regression.

- One commit measured **+2.7%** — in release builds it's actually a **speedup** (the cost was inside `#if !NDEBUG`)
- Another "+1.2%" regression shrank to **+0.03%** in a real release build

So the discipline now:

1. **Localize** with manyclangs
2. **Confirm** with one native, assertions-off build
3. **Audit trunk** — several pins were already reverted or fixed upstream.

New codebase, new regressions — every time

Each codebase stresses a different part of Clang. Templates found Sema regressions, the preprocessor-heavy targets found lexer ones.

This week I Decided to benchmark the Linux kernel and pinned:

- `getPresumedLoc` **on every** `#define` (+0.7%) — kernels define tens of thousands of macros. Found, fixed, **already merged upstream**
- `Stmt` **class bits 8→9** (+0.8%) — every `isa<>` check got 2 instructions more expensive. Required change
- **P1857R3 modules lookahead** (+0.5%) — the same commit already pinned on Boost.MPL and Spirit X3, now confirmed on plain C too

What's next

So far I have only been benchmarking the frontend and totally forgot about the backend so more benchmarks are needed in that area

Thank you! Questions?