Agni: Fast Formal Verification of the Verifier's Range Analysis

Harishankar Vishwanathan (Rutgers U.)

Matan Shachnai (Rutgers U.)

Paul Chaignon (Isovalent)

Santosh Nagarakatte (Rutgers U.)

Srinivas Narayana (Rutgers U.)

Agni

- Agni recap
- Solvers are slow
- Divide-and-conquer
- Agni’s CI
- Conclusion

Agni

Agni Recap

- Goal: Automated formal verification of the verifier's range analysis

- Verifier tracks register and stack slots with 5 abstract domains:
  - 4 interval domains (u32/u64, signed/unsigned)
  - 1 bitwise domain, tums

- Updates on ALU & JMP operations
  - First, each abstract value is independently updated
  - Then, abstract values learn from one another

Agni Recap

1. Extracts the verifier functions of interest

2. Adds some glue code:
   a. To remove writes into global verifier state
   b. To specialize functions for each ALU/JMP operation
   c. To replace LLVM builtins

3. Compiles to LLVM IR

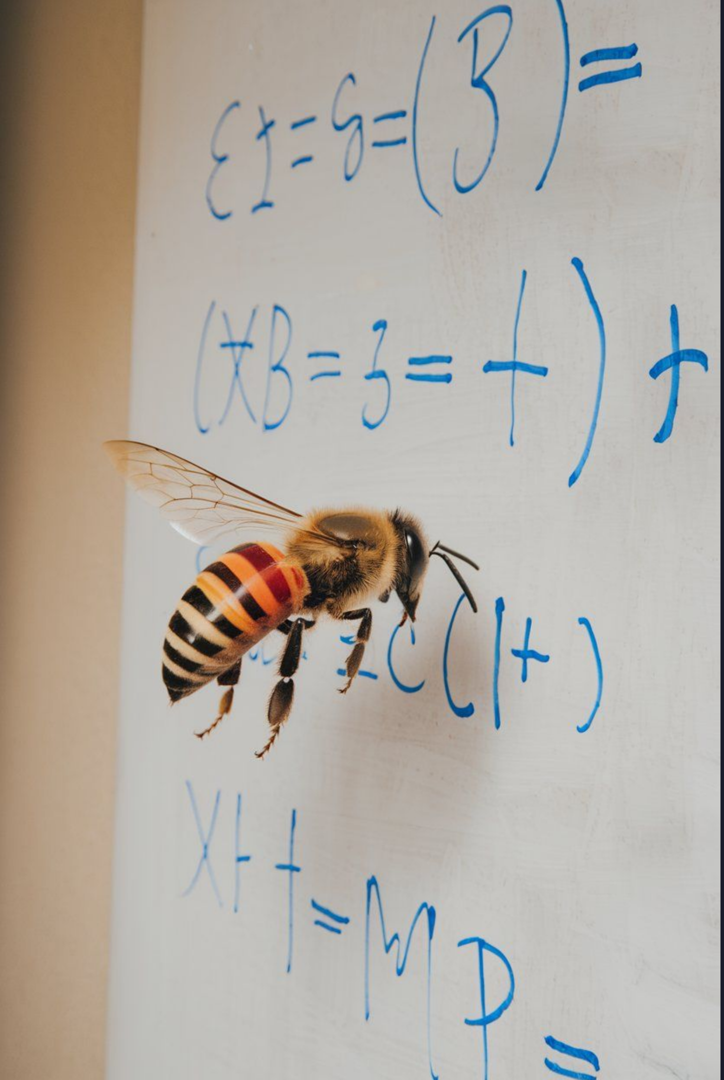
4. Converts the LLVM IR into SMT formula

5. Adds soundness conditions

6. Solve with Z3 solver!

Agni Recap

- Also able to synthetize PoCs for soundness violations
- See Hari’s talk at Linux Plumbers 2023 for details and past results



Agni

Agni: Goal

- Run regularly against latest kernels and patchsets

- Challenges:
  - Needs to be fast: at most a few hours

  - Needs to be maintainable: no need to update Agni for every kernel

Solvers are Slow!

- Solving starting taking days, then weeks

Kernel version	Runtime
v4.14	2.5h
v5.5	2.5h
v5.9	4h
v5.13	10h
v5.19	36h
v6.3	36h
v6.4	several weeks
v6.5	timeout
v6.6	timeout
v6.7	timeout
v6.8	timeout

"Fixing Latent Unsound Abstract Operators in the eBPF Verifier of the Linux Kernel", SAS'24

Why is Solving Slow?

- The logic for one operation is small
  - Ex. ~60 lines of C for BPF_AND (mostly `scalar_min_max_and`)

Why is Solving Slow?

- The logic for one operation is small
  - Ex. ~60 lines of C for BPF_AND (mostly `scalar_min_max_and`)

- But `reg_bounds_sync` is also executed after each per-operation logic
  - It tends to be a bit to a lot more complex than the per-operation logic

  - Runtime increases linked to `reg_bounds_sync` becoming “smarter”

- Solver runtime tends to increase exponentially with size of input formulas

Agni

- Agni recap
- Solvers are slow
- Divide-and-conquer
- Agni's CI
- Conclusion

Divide-and-conquer

- Hari et al. devised a solution: divide-and-conquer
  - That is, verify `reg_bounds_sync`‘s soundness separately

- `reg_bounds_sync` AND per-operation logic are sound ⇒ the whole is sound
  - Otherwise, we can’t deduce anything!

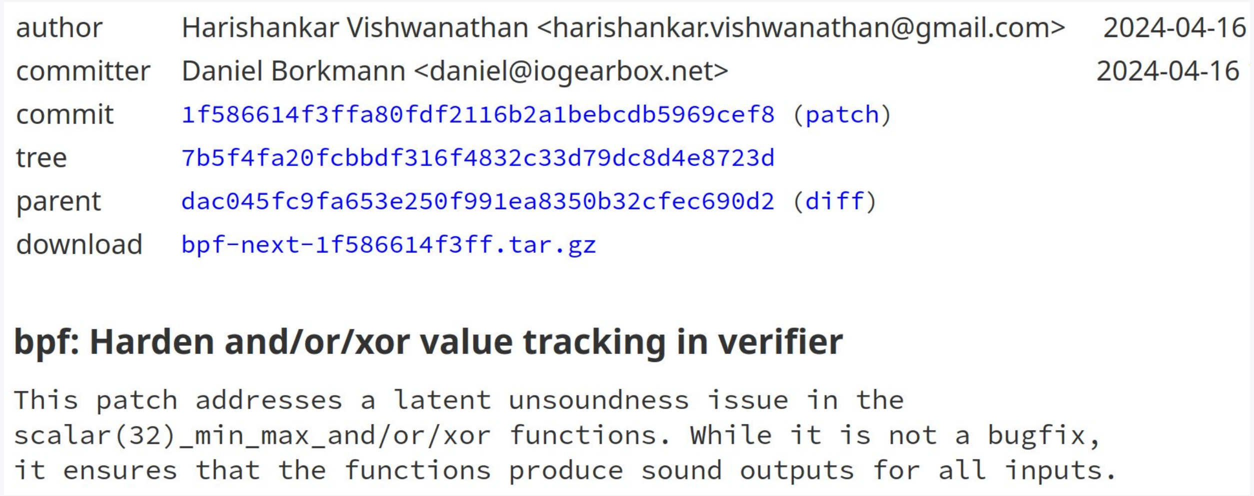
Divide-and-conquer

- Problem: Some per-operation logic is unsound, so can’t deduce anything
    - (Unless we solve the whole, but too long)

Divide-and-conquer

- Problem: Some per-operation logic is unsound, so can’t deduce anything
  - (Unless we solve the whole, but too long)

- Fixed by Hari et al. in [v6.10](v6.10) 👇



author	Harishankar Vishwanathan <harishankar.vishwanathan@gmail.com>	2024-04-16
committer	Daniel Borkmann <daniel@iogearbox.net>	2024-04-16
commit	1f586614f3ffa80fdf2116b2a1bebcdb5969cef8 (patch)	
tree	7b5f4fa20fcbbdf316f4832c33d79dc8d4e8723d	
parent	dac045fc9fa653e250f991ea8350b32cfec690d2 (diff)	
download	bpf-next-1f586614f3ff.tar.gz	

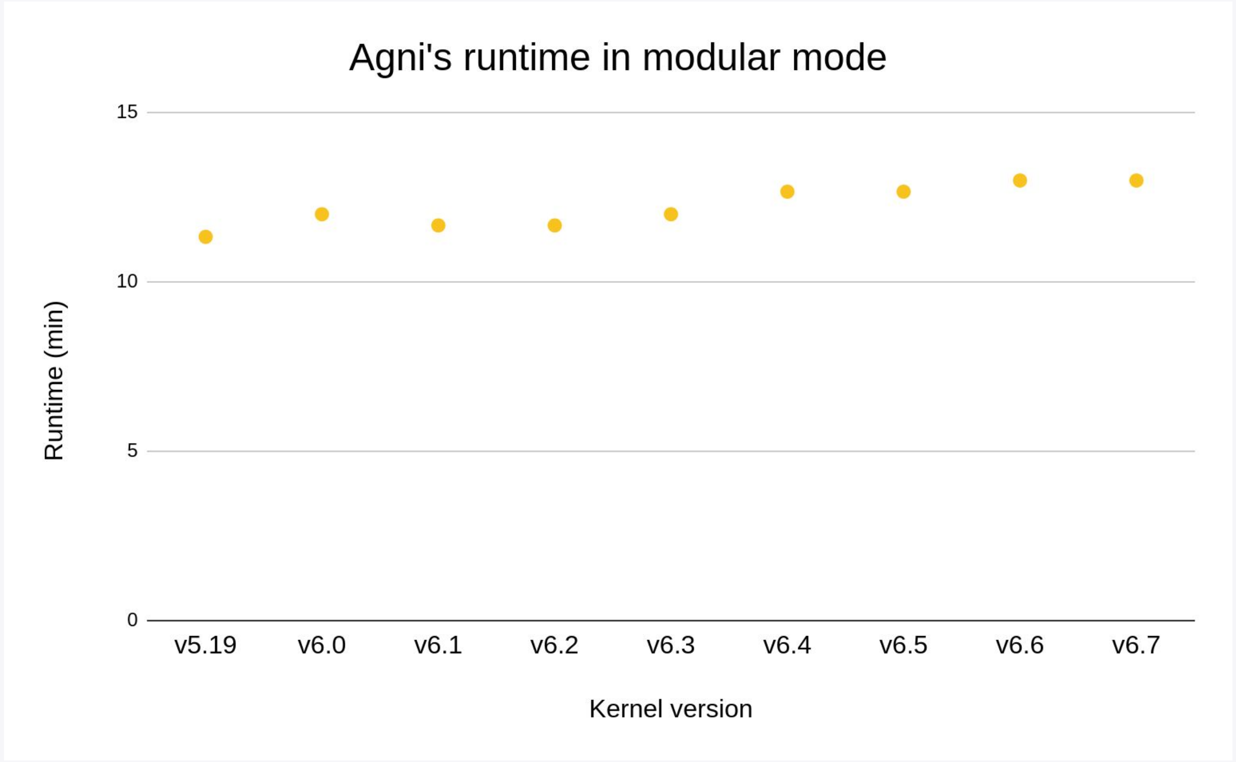
bpf: Harden and/or/xor value tracking in verifier

```
This patch addresses a latent unsoundness issue in the
scalar(32)_min_max_and/or/xor functions. While it is not a bugfix,
it ensures that the functions produce sound outputs for all inputs.
```

Divide-and-conquer

- Back in business!
- New –modular mode to verify `reg_bounds_sync` separately
- All explained in new SAS’24 paper!

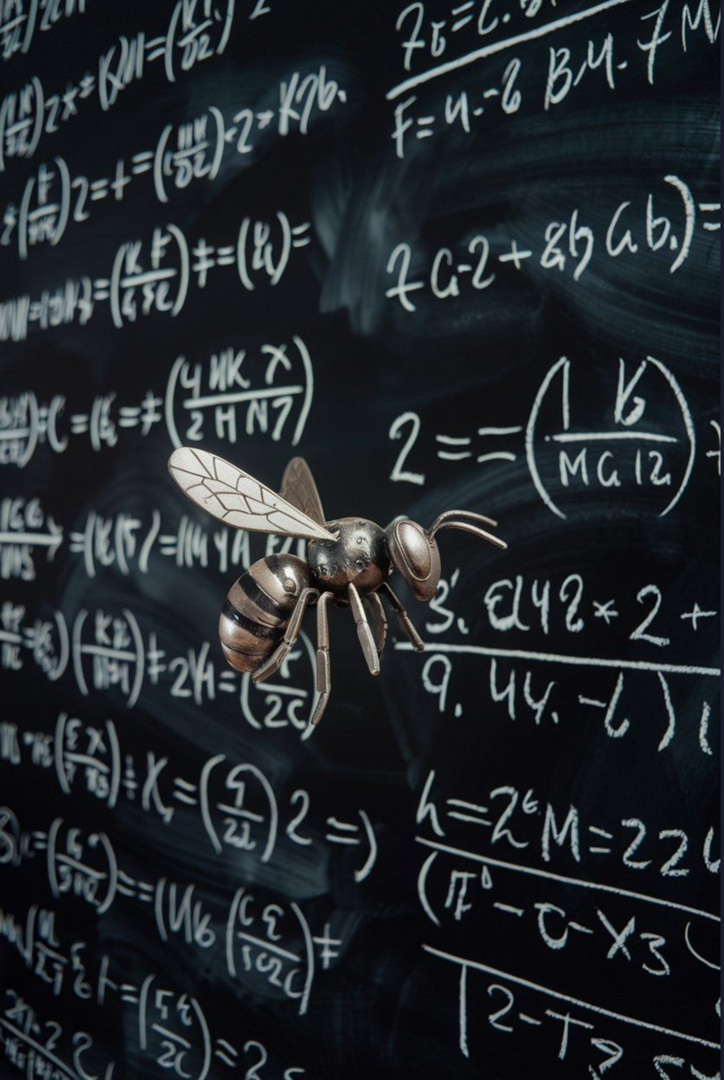
Divide-and-conquer



Divide-and-conquer

- Per-operation OR `reg_bounds_sync` unsound ⇒ can’t deduce anything

- Not an issue as long as:
  - Per-operation functions (ex. `scalar_min_max_and`) stay sound
    AND
  - `reg_bounds_sync` stays sound

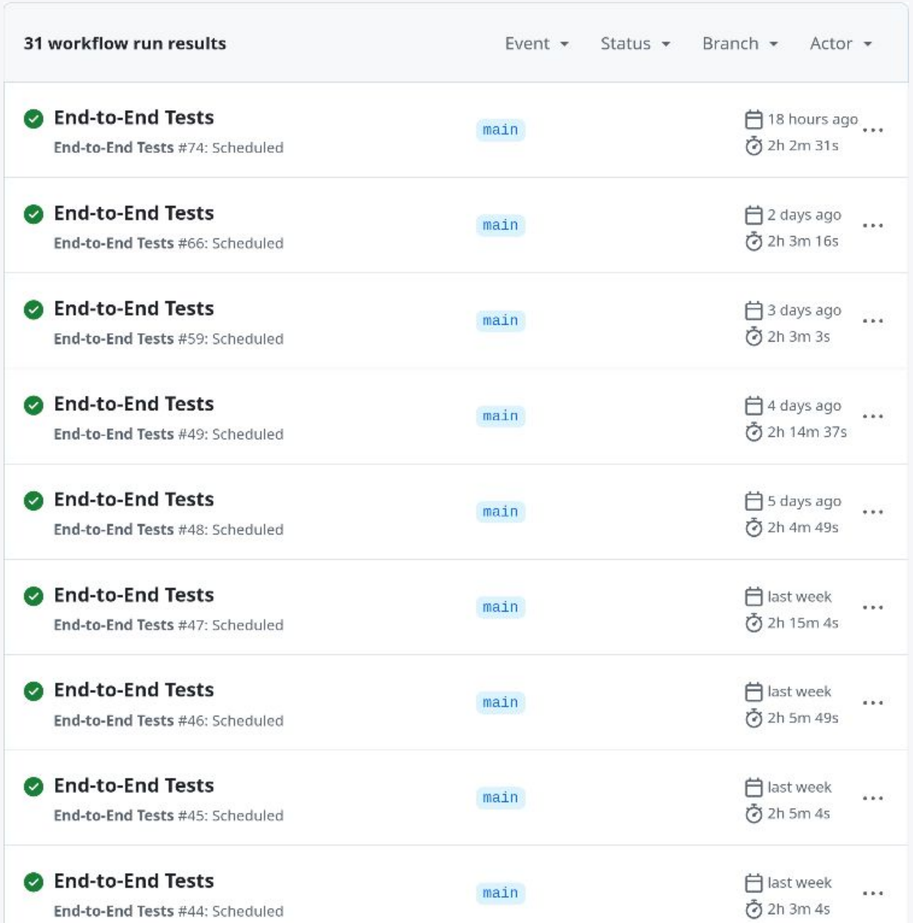


Agni

- Agni recap
- Solvers are slow
- Divide-and-conquer
- Agni's CI
- Conclusion

Agni's CI

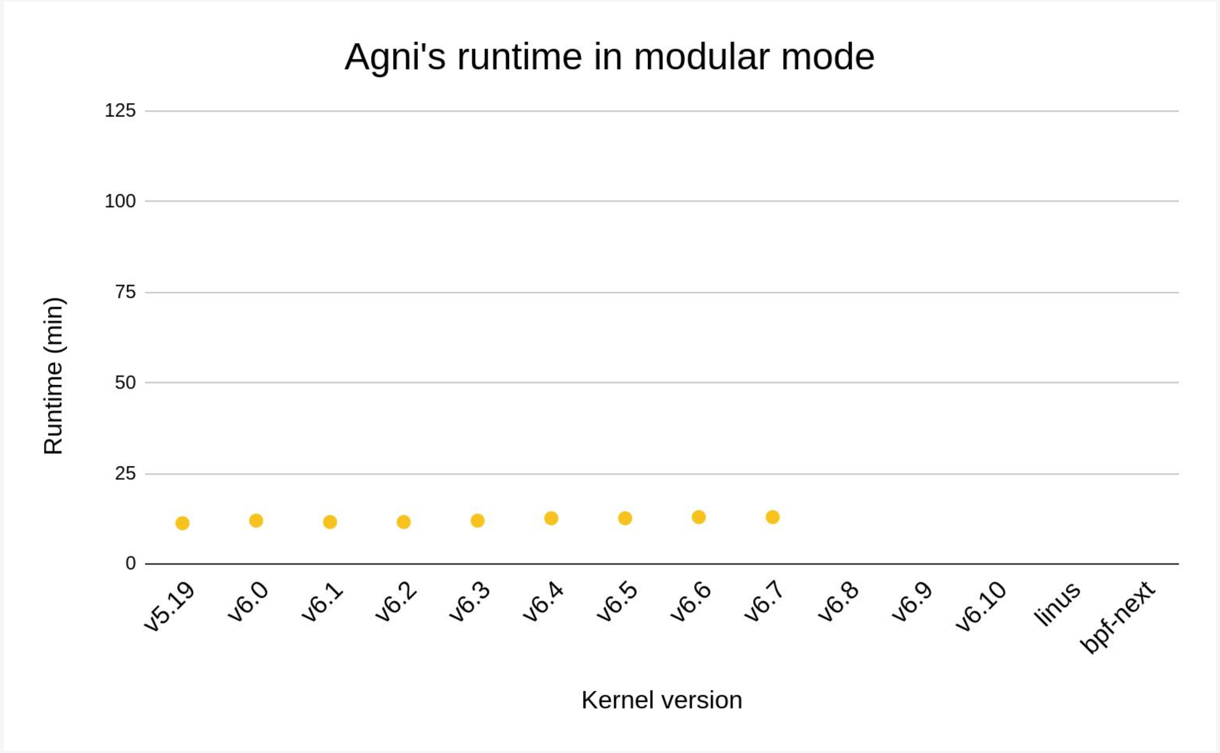
- Building a CI for Agni
  - Test Agni itself
  - Test the kernel
- Covers bpf, bpf-next, and linus's trees
- Runs once a day
- Has been running for a month



31 workflow run results

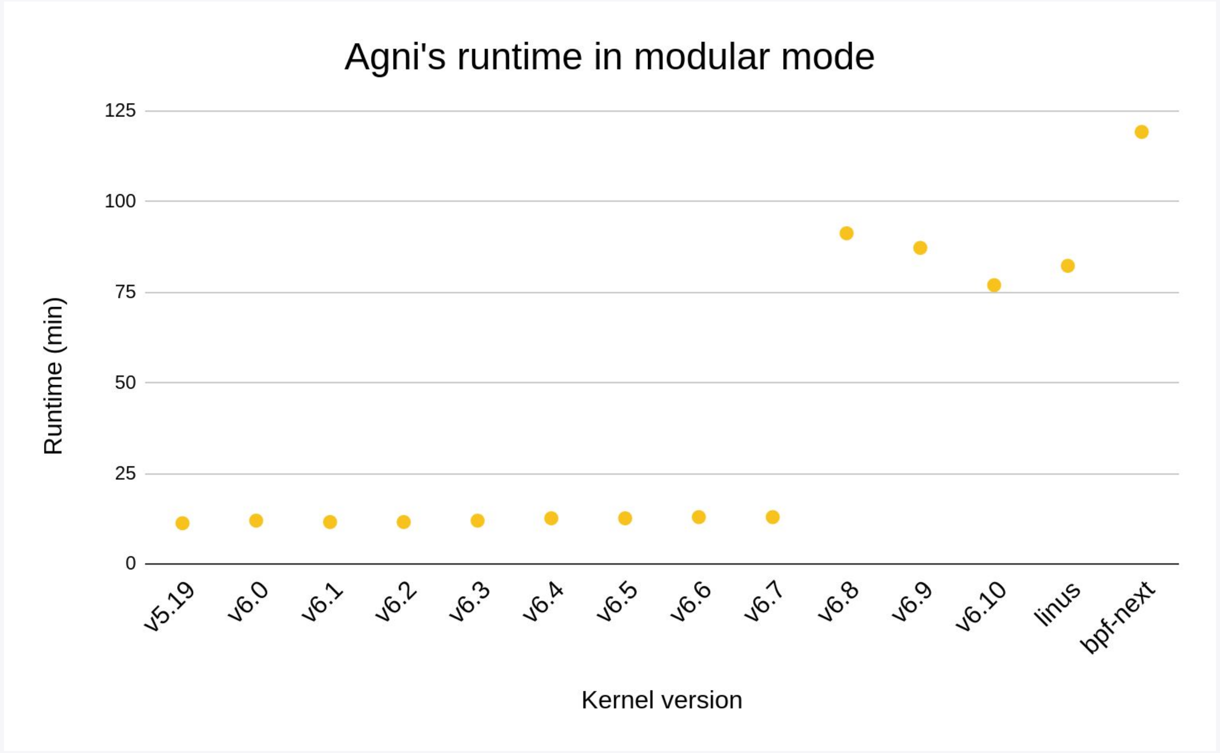
✓ End-to-End Tests End-to-End Tests #74: Scheduled	main	18 hours ago	2h 2m 31s
✓ End-to-End Tests End-to-End Tests #66: Scheduled	main	2 days ago	2h 3m 16s
✓ End-to-End Tests End-to-End Tests #59: Scheduled	main	3 days ago	2h 3m 3s
✓ End-to-End Tests End-to-End Tests #49: Scheduled	main	4 days ago	2h 14m 37s
✓ End-to-End Tests End-to-End Tests #48: Scheduled	main	5 days ago	2h 4m 49s
✓ End-to-End Tests End-to-End Tests #47: Scheduled	main	last week	2h 15m 4s
✓ End-to-End Tests End-to-End Tests #46: Scheduled	main	last week	2h 5m 49s
✓ End-to-End Tests End-to-End Tests #45: Scheduled	main	last week	2h 5m 4s
✓ End-to-End Tests End-to-End Tests #44: Scheduled	main	last week	2h 3m 4s

Agni’s CI


Agni's runtime in modular mode
Runtime (min)
125
100
75
50
25
0
v5.19
v6.0
v6.1
v6.2
v6.3
v6.4
v6.5
v6.6
v6.7
v6.8
v6.9
v6.10
linus
bpf-next
Kernel version

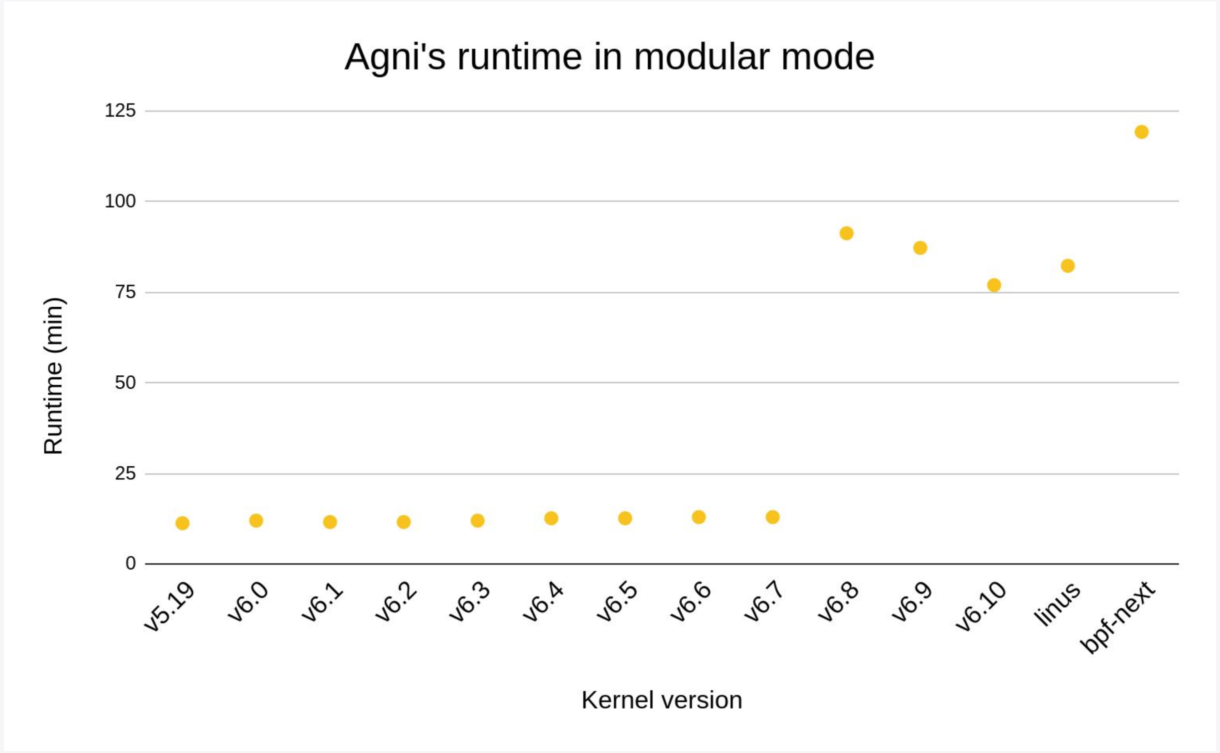
Agni’s CI

- It got worse again



Agni’s CI

- It got worse again
- More divide-and-conquer?
- Could verify `__update_reg_bounds`, `__reg_deduce_bounds_`, `__reg_bound_offset` separately
- Caveat: Need to keep them independently sound



Agni

- Agni recap
- Solvers are slow
- Divide-and-conquer
- Agni’s CI
- **Conclusion**

Conclusion

- Hardening Agni:
  - Reduce amount of glue code
  - More tests (ex. SMT equivalence check for PRs)
  - Keep reviewing CVEs for potential false negatives

- A small change in the verifier enabled a significant speed up of the formal verification

Thanks !

Appendix: Weakened Soundness Specification

- ~1y ago, a bug was found in verifier, missed by Agni because it never happened at runtime (always-false branch)

- But it could happen under speculative execution (hence verifier check)

- Now supported by Agni behind a flag:
  - Weakened specification to also essentially follow both branches

  - See agni#15 for details