Performance Analysis Based on Noise Injection

Výkonnostní analýza programů založená na vkládání šumu

Matúš Liščinský

Supervisor: **Ing. Tomáš Fiedor, Ph.D.**

Motivation

- During profiling performance issues **may not manifest**.
- It is **hard to find** “exhausting” workloads.
  - Although, automatic generation of workloads has been explored (e.g. *fuzz-testing*).
- However, many workloads do not impact performance much!

Motivation

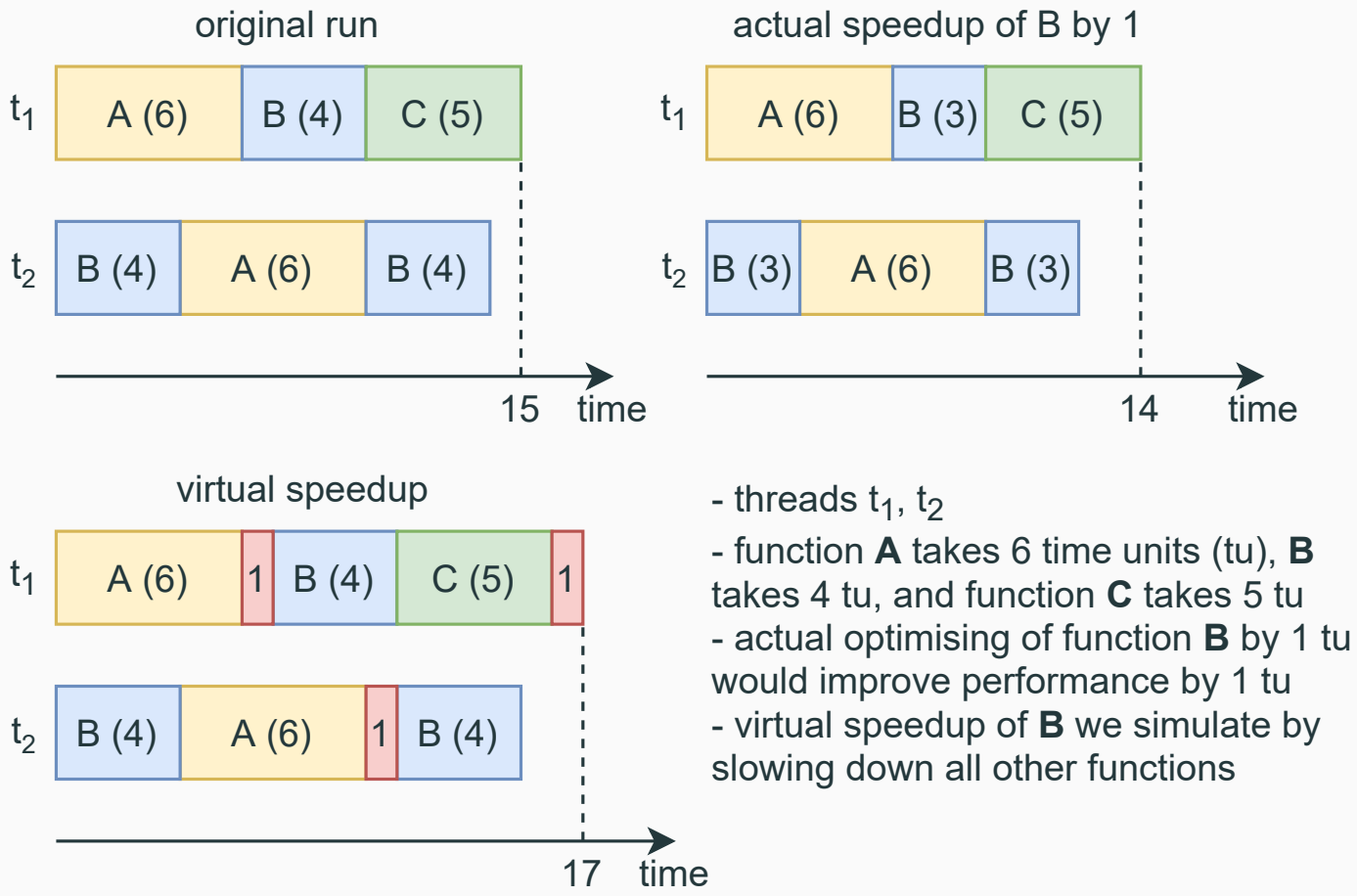
- During profiling performance issues **may not manifest**.
- It is **hard to find** “exhausting” workloads.
  - Although, automatic generation of workloads has been explored (e.g. *fuzz-testing*).
- However, many workloads do not impact performance much!
- Recently, several works (e.g. *PerfBlower*, *Coz*) proposed to adapt noise injection to **amplify the performance**.
- Our Goal: Blow up the performance! Make the issues manifest!

Related work

- *Perun-fuzz*
  - automatic workload generator with **performance tuned** fuzzing
  - but, it is **not guaranteed** to find performance-intensive workloads
- *PerfBlower*
  - uses virtual amplification to **blow up** the effect of small performance problems
  - focuses on **memory-related** performance problems in Java
- *Coz*
  - focuses on **causal profiling**: calculating virtual speedup by **stopping concurrent threads**
  - has overhead in **communication between threads**
  - has to deal with special cases of **thread blocking**
  - selects in **random** the **source line** which will be virtually sped up

Noise Injection: Motivation

Mode 1: inject to **each** function except one to *simulate speedup*.

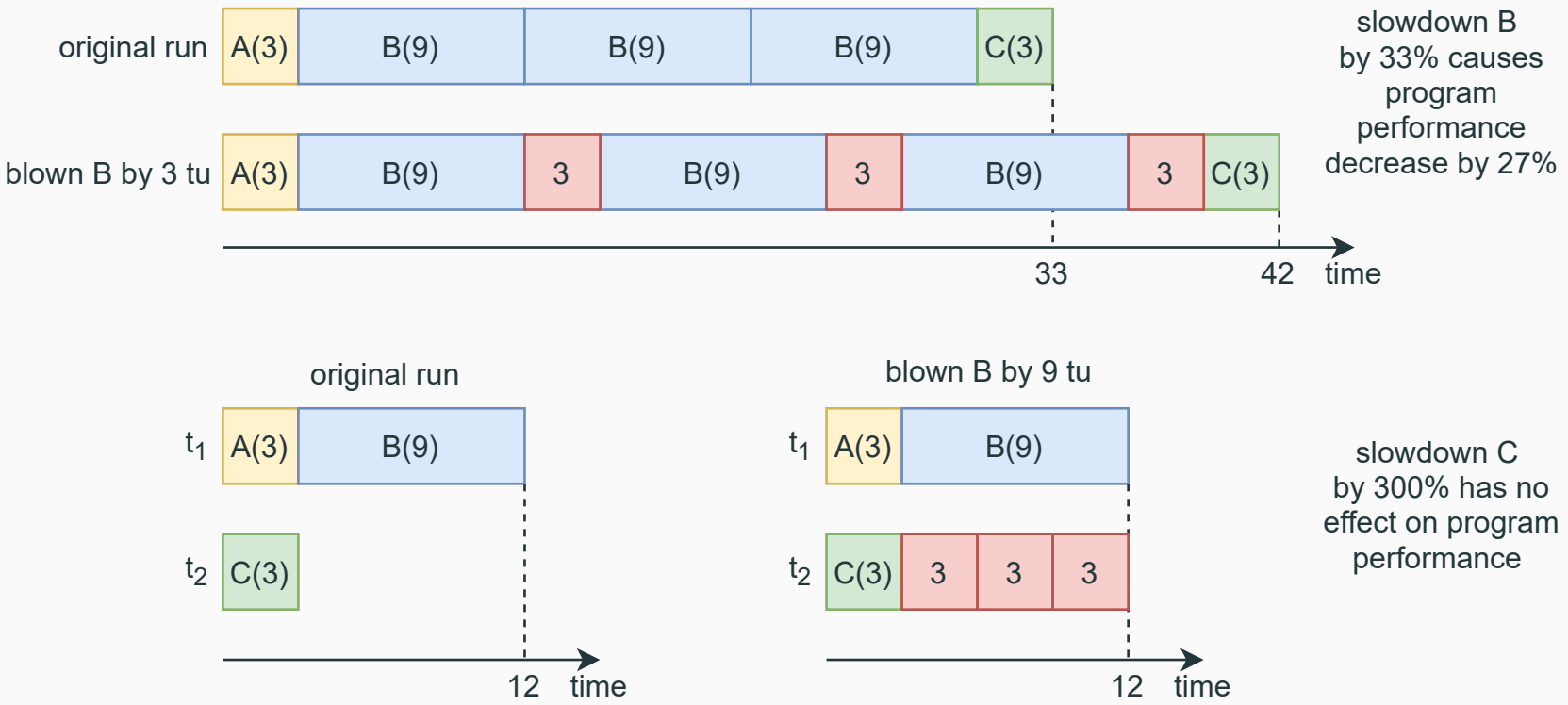


- threads t_1, t_2
- function **A** takes 6 time units (tu), **B** takes 4 tu, and function **C** takes 5 tu
- actual optimising of function **B** by 1 tu would improve performance by 1 tu
- virtual speedup of **B** we simulate by slowing down all other functions

estimated speedup (t) = (original time (t) + functions' calls (t) * delay) - virtual speedup time (t)

Noise Injection: Motivation

Mode 2: inject to **one** selected function to *find* *bottlenecks* *or* *places with no impact on performance*



Noise Injection Implementation

How? We could utilise frameworks already **incorporated** in *Perun*:

- *SystemTap*
  - supports **function level** probing (probe points can be at the function beginning, return, ...)
  - can probe a specific statement defined by either a line in the source code or the address in the module
  - supports **C-embedded** functions that can insert a delay
  - noise implementation by **busy waiting** (checks can be suppressed)
- *eBPF*
  - **state-of-the-art** dynamic instrumentation tool
  - high **safety** requirements, so noise injection is not possible

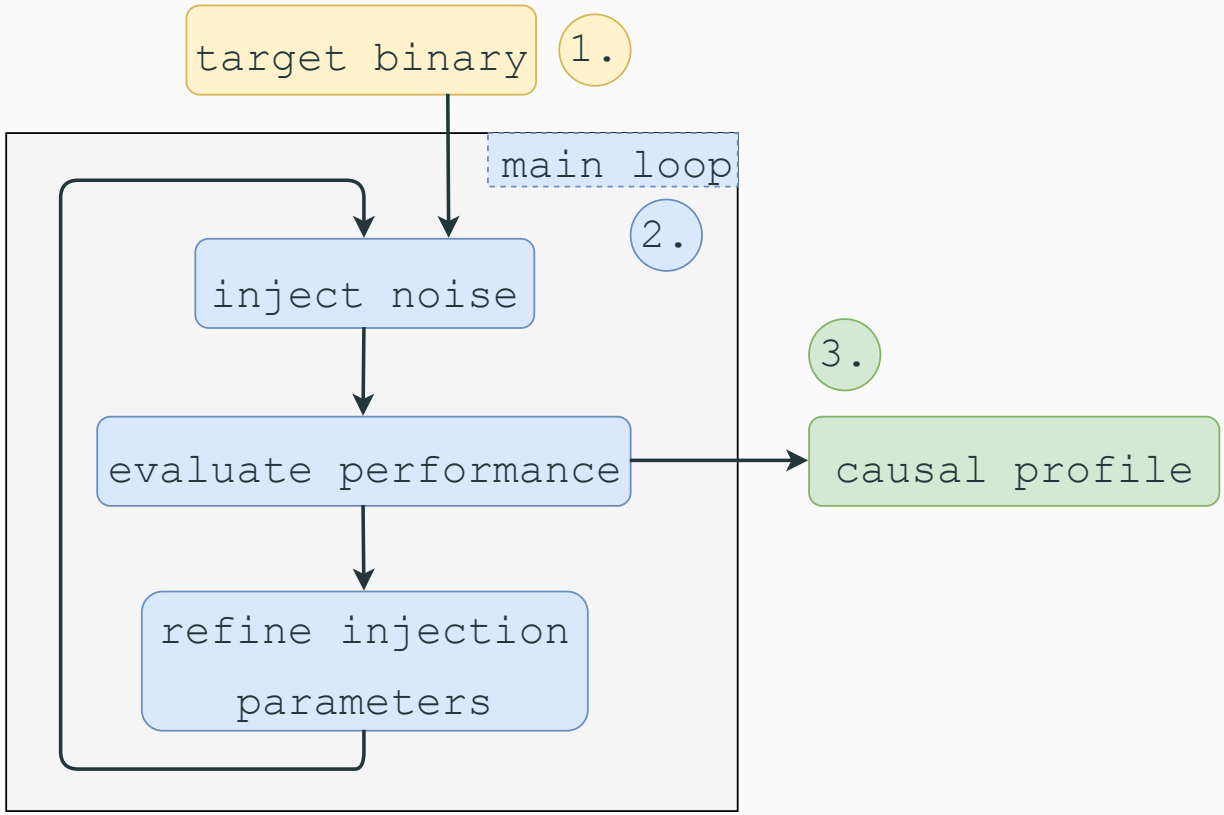
Noise Injection Implementation

How? Instead, we will use **new** dynamic instrumentation framework:

- *Pin*
  - supports **thread-level** instrumentation (can instrument multi-threaded programs)
  - supports different levels of **granularity** (*instruction*, *routine*, *trace*)
  - offers **built-in noise injecting** functions: *PIN_Sleep()*, *PIN_Yield()*
  - noise is implemented as **busy waiting** (looping until a condition is true) is also feasible
  - has other uses for **collecting runtime information**

Our proposal: Perun-blow

- We propose to adapt fuzz-testing loop with noise injection



- noise injection parameters:
  - *location* (e.g. to function *f()*)
  - *noise type* (e.g. *Pin_Sleep()*)
  - *strength* (e.g. *1ms*)

Summary of Perun-blow

How?

- We will use *Pin* for dynamic binary instrumentation.

Where?

- We will inject to **one** chosen function,
- or to **each** function except one.

How much?

- We will adjust noise depending on performance results.

What is the result?

- **Causal profile** - dependence of the program's performance on the injected noise and its parameters.

Summary

- *Already done*
  - ☑ Studied recent techniques for performance testing.
  - ☑ Got acquainted with frameworks for dynamic instrumentation of programs (*eBPF*, *SystemTap*, *Pin*) and their capabilities of noise injection.
  - ☑ Proposed an evolution algorithm, that automatically injects noise into analysed programs.
  - ☑ Partially implemented the algorithm.
- *Next steps*
  - ☐ Finish the implementation of the algorithm.
  - ☐ Propose a visualisation or interpretation, that will illustrate the results.
  - ☐ Evaluate the solution, and incorporate it into *Perun*.

Acknowledgements

This work is **supported** by:

- **Red Hat**, Inc.
- **VeriFIT** group (BUT FIT)

Red Hat