h

# VLS Performance Tuning

## VLS

—

Lakshya Singh
Indian Institute of Technology BHU, Varanasi, India

## Name and Contact Information

**Name**: Lakshya Singh
**Primary Email**: lakshay.singh1108@gmail.com
**Discord username**: king-11#7210
**LinkedIn**: lakshyasingh11
**GitHub**: king-11
**Location**: India
**Timezone**: Kolkata, INDIA, UTC+5:30
**Typical working hours**: 6 – 8 hours between 10 AM to 10 PM (local time)

# About Me

## Education

- **University:** Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, India
- **Major:** Computer Science and Engineering (B.Tech.)
- **Degree Level:** 4th-year undergraduate
- **Graduation Year:** 2023

## Why is this project important to me?

The project deals with performance enhancement which I have been really enthusiastic about. I have been involved in our college's CTF community, which educates about cyber security; by working on this project, I can gather more knowledge on improving the security of lightning nodes. Moreover, the project is written in Rust, which I enjoy working with, given the surety of compiling time checks.

## Why me?

I am enthusiastic about open-source and learning about new technologies in core computer science. I have experience working remotely with open-source communities. I plan on learning and making an impact to my full potential in this project and improving my knowledge about Bitcoin and overall Web 3.0. Having previously worked on a rust project makes it the right choice of project for me due to its niche nature.

## Skills aligned with the project

I have experience with system programming and sophisticated projects like VLS. I have been doing a lot of rust programming in recent times:

- **Genetic Algorithm Library**: https://github.com/king-11/genx
- **Vehicle Routing Problem solution:** https://github.com/king-11/Vehicle-Routing-Problem
- **Advent of Code 2022:** https://github.com/king-11/AdventOfCode/tree/main/src

All this has provided me experience in writing excellent and standard rust code using standard library optimally for various tasks. All these projects ensure the

usage of good programming paradigms writing clean code, and using the most optimal and performant implementation method.

## Experience

I have worked previously with several open-source communities, which can be found on my github profile. I have also done several internships, two of which were open-source mentorship similar to the summer of Bitcoin, i.e. Google Summer of Code and Linux Foundation LFX Mentorship.

Linux Foundation LFX Mentorship - Vitess

Software Developer Intern (Full-time) Sep 2021 - Dec 2021

Blog

- Collations: Add in support for MySQL Collations and Character Sets.
- Query Planner: Make collation-aware decisions for Order By, Group By and Join planning.
- Automation: Automation scripts for transporting MySQL's C++ code to Go Lang.

Google Summer of Code - Chapel

Student Developer (Full-time) May 2021 - Sep 2021

Blog

- Sockets: Creation of a Socket Library in Chapel Lang using C System Calls
- Asynchronous: Async IO model for IO calls using event notification library and qthreads
- Concurrency: Ensuring the Parallel and Non-Blocking nature of constructs with robust unit testing.

## Synopsis

The project aims to benchmark and improve the performance of the Validating Lightning Signer. The signer holding the keys must be able to sign transactions when the user requests quickly. The goal is to evaluate the current latency of VLS and implement optimisation strategies to reduce latency while maintaining a high level of security.

The project can be divided into three main tasks:

1. Identifying main operations for benchmarking
2. Benchmarking the main operations of a node running against the VLS
3. Identifying bottlenecks causing latency for different configurations
4. Updating the codebase to handle those bottlenecks

## Mentors

- Dev Random (@devrandom01)
- Ken Sedgwick (@ksedgwic)

## Plan of Action

### 1. Operations Identification

For benchmarking, it's necessary to identify the main/important functions of the VLS process. We can consider integration tests as the basis for identifying operations like pay, receive, etc.

As mentors would better understand the whole architecture, I would request their help and work alongside them to identify operations of key interest.

### 2. Benchmarking Operations

We would have to write down new tests or use existing unit and integration tests for benchmarking operations identified in the previous step. There are currently two options that we can use for benchmarking.

- External Crate

  Crate-like Criterion can be used for the purpose of benchmarking. It is widely used and stable, unlike the second available option. The only drawback with criterion might be more involved boilerplate code apart from that the API is more or less similar in both of the cases

```
fn criterion_benchmark(c: &mut Criterion) {
    c.bench_function("fib 20", |b| b.iter(|| fibonacci(black_box(20))));
}
```

- Built-in Benchmarks

Rust's built-in benchmark tests are a simple starting point, but they use unstable features and only work on Nightly Rust. After discussing in the matrix channel, it's evident; we are already using nightly for a few of the packages considering that it won't be much trouble for us even if we start using nightly for benchmarking in all the crates. The ability to mark functions as the bench reduces the boilerplate for us.

```
#[bench]
fn bench_add_two(b: &mut Bencher) {
    b.iter(|| add_two(2));
}
```

## 3. Running on Different Configurations

Identifying latency on different configurations is also very important for this project. Given that I don't have access to servers and microcontrollers, we can use virtual machines and systems like QEMU for testing, which would allow identifying different bottlenecks on different systems.

## 4. Handling Bottlenecks

After finding the functions responsible for the latency of the system, we can work on optimising them using several techniques, which are listed below not an exhaustive list by any chance but just some suggestions:

- **Algorithmic Optimization**

Identifying methods for algorithmic optimisation like using better cache mechanisms, faster data structures, and reducing the time using logarithmic algorithms.

It is sometimes worth avoiding certain standard library types in favour of faster alternatives. This can be enabled using linting rules

```
disallowed-types = ["std::collections::HashMap", "std::collections::HashSet"]
```

- **Build Config Optimization**

Link-time optimisation (LTO) can be enabled, which is a whole-program optimisation technique that can improve runtime performance by 10-20% or more at the cost of increased build times.

The simplest way to try LTO is to add the following lines to the Cargo.toml file and do a release build.

```
[profile.release]
lto = true
```

The Rust compiler splits your crate into multiple codegen units to parallelize (and thus speed up) compilation. However, this might cause it to miss some potential optimisations. If we want to improve runtime performance at the cost of a larger compile time, we can set the number of units to one.

```
[profile.release]
codegen-units = 1
```

- **Inlining Functions**

Entry to and exit from hot, uninlined functions often accounts for a non-trivial fraction of execution time. Inlining these functions can provide small but easy speed wins.

- #[inline]. This suggests that the function should be inlined, including across crate boundaries.
- #[inline(always)]. This strongly suggests that the function should be inlined, including across crate boundaries.

These are some generic methods, but for different configurations, we might need a different plan of action based on results.

# Timeline

## Proposal Review Period - April 15 to May 1, 2022

1. Familiarise me completely with the project's functionality and architecture.
2. Help other contributors if they are stuck when mentors are unavailable.
3. Make contributions to the project and examine existing contributor patterns.

## Coding Period - May 1 to August 15, 2023

### May 1 - May 8:

1. Start planning the project and milestones with the mentor.
2. Attend talks and seminars on various Bitcoin-related topics.
3. Familiarise me with the code base by focusing on specific parts critical for our benchmarking project.

### May 8 - May 22:

1. Identify critical operations with consideration for mentor and community.
2. Set up the configuration for performing benchmarking across all packages on the different configurations, including microcontrollers.
3. Breakdown operations into finer components for performing benchmarking.

### May 22 - June 5:

1. Write down tests for performing benchmarking operations
2. Work on unit tests to perform benchmarks.
3. Progress on integration tests for studying the workflows of users.

### June 5 - June 12

1. Running unit test benchmarks on different configurations and saving the results.
2. Adding more integration benchmark tests for different necessary operations.

### June 12 - June 26:

1. Running integration benchmark tests
2. Having identified latency operations, list all of them down and separate them out

3. Find solutions for handling bottlenecks in operations.

### June 26 - July 3:

1. Working on algorithmic and trivial optimisation for bottlenecks
2. Re-running benchmark and comparing with baseline.

### July 3 - July 17:

1. Working on memory allocation-related optimisations.
2. Using inline to handle small function optimisation.
3. Re-running benchmark and comparing with updated baseline.

### Mid-Term Evaluations - July 3 to July 7, 2023

1. Mentors submit mid-term evaluations of their mentees.

### July 17 - July 31:

1. Adding in lint rules for restricting to performant code.
2. Updating the configuration for handling bottlenecks by choosing between compiler time vs runtime speedup.
3. Re-running benchmark and comparing with updated baseline.

### July 31 - August 7:

1. Identify LDK and CLN  bottlenecks causing latency in VLS
2. Read about both of these projects.
3. Make a plan for changes necessary in LDK and CLN to enable VLS speedup.
4. Propose the changes and start working on them.

### August 7 - August 15:

1. Gather a review of the work done till now from the community.
2. Writing down a blog on what I did this summer with VLS and sharing it.
3. Completing pending tasks and moving towards final evaluation.
4. Outline plans for further projects that can be done related to benchmarking and bottleneck handling.

### Post SoB:

1. I will continue contributing to **VLS** by adding more features and helping maintain the repository after SoB ends.

### Final Evaluations - August 21 to August 25, 2022

1. Mentors do a final review code and determine if the students have completed their Summer of Bitcoin project.