Google Summer of Code 2024 [LLVM]

The 1001 thresholds in LLVM

By
Shourya Goel

Table of Contents

Basic Information

Name: Shourya Goel
Major: Computer Science and Engineering
Degree: Bachelor of Technology
Year: Sophomore
Institute: Indian Institute of Technology (IIT), Roorkee
Institute Email: shourya_g@cs.iitr.ac.in
Personal Email: shouryagoel10000@gmail.com
GitHub: Sh0g0-1758
LinkedIn: Shourya Goel
Telephone: +91-9971749585
Timezone: Indian Standard Time (UTC+05:30)

Prologue

Project Name	The 1001 thresholds in LLVM
Skills	LLVM, profiling
Mentor	Jhueckelheim jdoerfert wsmoses
Size of Project	Medium (175 hours)

Project Overview

Pilot:

LLVM has lots of thresholds and flags to avoid costly cases. But it is not really clear if these thresholds are useful, their value is reasonable, and what impact they really have. Since there are a lot of them, a simple exhaustive search would not suffice. Thus conducting an extensive study directed towards exploring the thresholds, understanding when they are hit, what it means if they are hit, how we should select their values, and if we need different “profiles” is needed to better understand the role of hard coded thresholds in llvm.

Objectives:

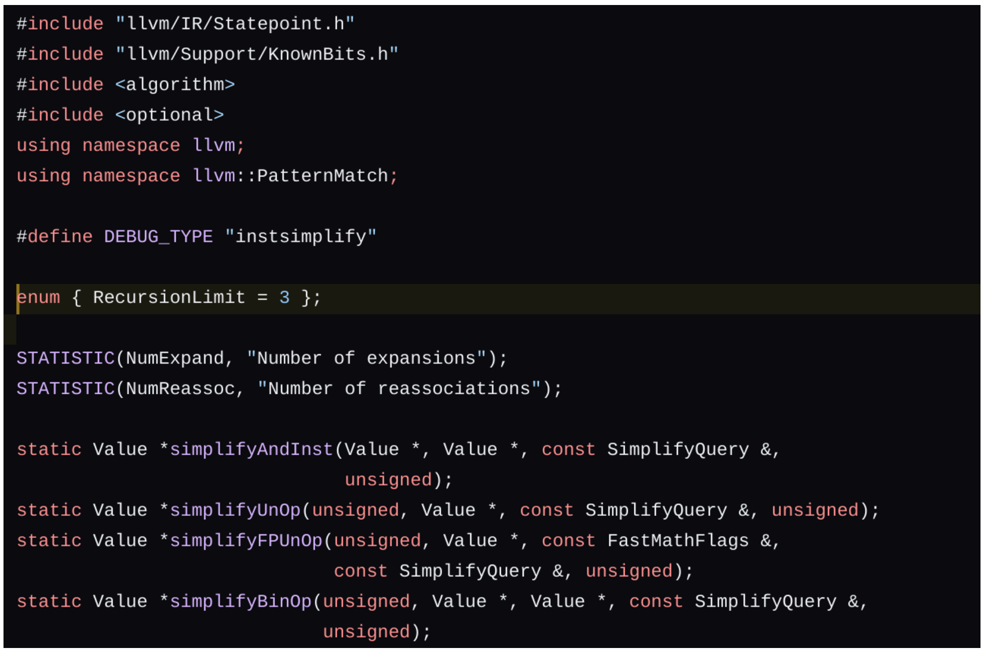
- Explore different thresholds in llvm.
- Understanding what it means for a threshold to be hit.
- Profiling different thresholds.
- Selecting optimal values for different thresholds.

Deliverables:

- Statistical evidence on the impact of various thresholds inside of LLVM’s code base, including compile time changes, impact on transformations, and performance measurements.

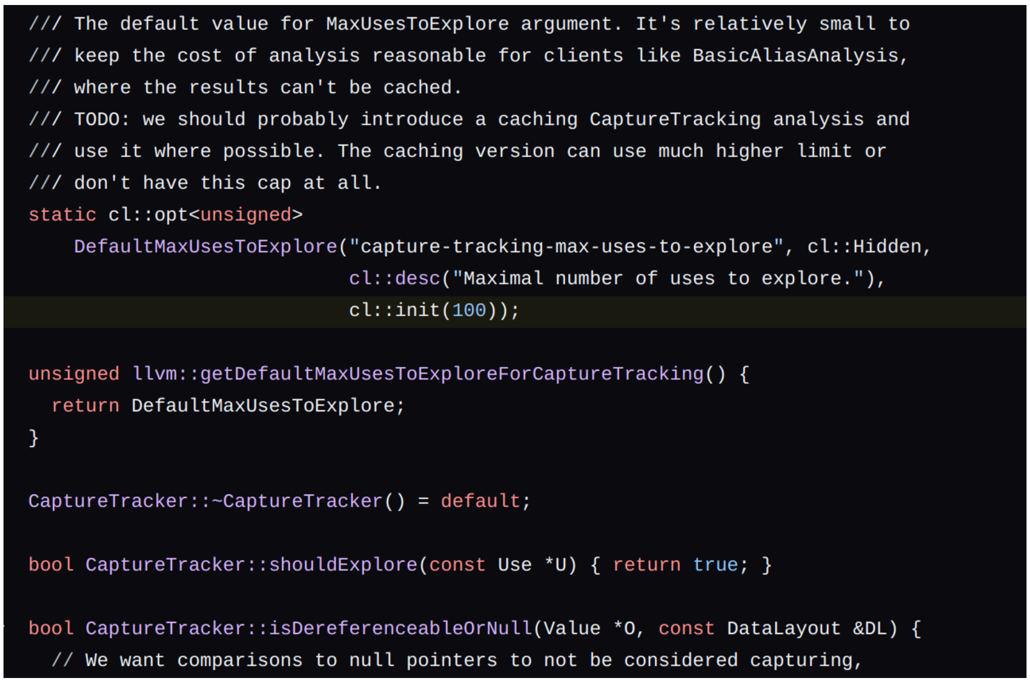
Implementation

My first step would be to figure out which part of the code needs to be tuned. For this task, I will first identify a subset of these thresholds that I can work with. I will add the others once I build the required methodology and tooling. A few examples of these thresholds are :



```
#include "llvm/IR/Statepoint.h"
#include "llvm/Support/KnownBits.h"
#include <algorithm>
#include <optional>
using namespace llvm;
using namespace llvm::PatternMatch;

#define DEBUG_TYPE "instsimplify"

enum { RecursionLimit = 3 };

STATISTIC(NumExpand, "Number of expansions");
STATISTIC(NumReassoc, "Number of reassociations");

static Value *simplifyAndInst(Value *, Value *, const SimplifyQuery &,
                              unsigned);
static Value *simplifyUnOp(unsigned, Value *, const SimplifyQuery &, unsigned);
static Value *simplifyFPUnOp(unsigned, Value *, const FastMathFlags &,
                             const SimplifyQuery &, unsigned);
static Value *simplifyBinOp(unsigned, Value *, Value *, const SimplifyQuery &,
                            unsigned);
```

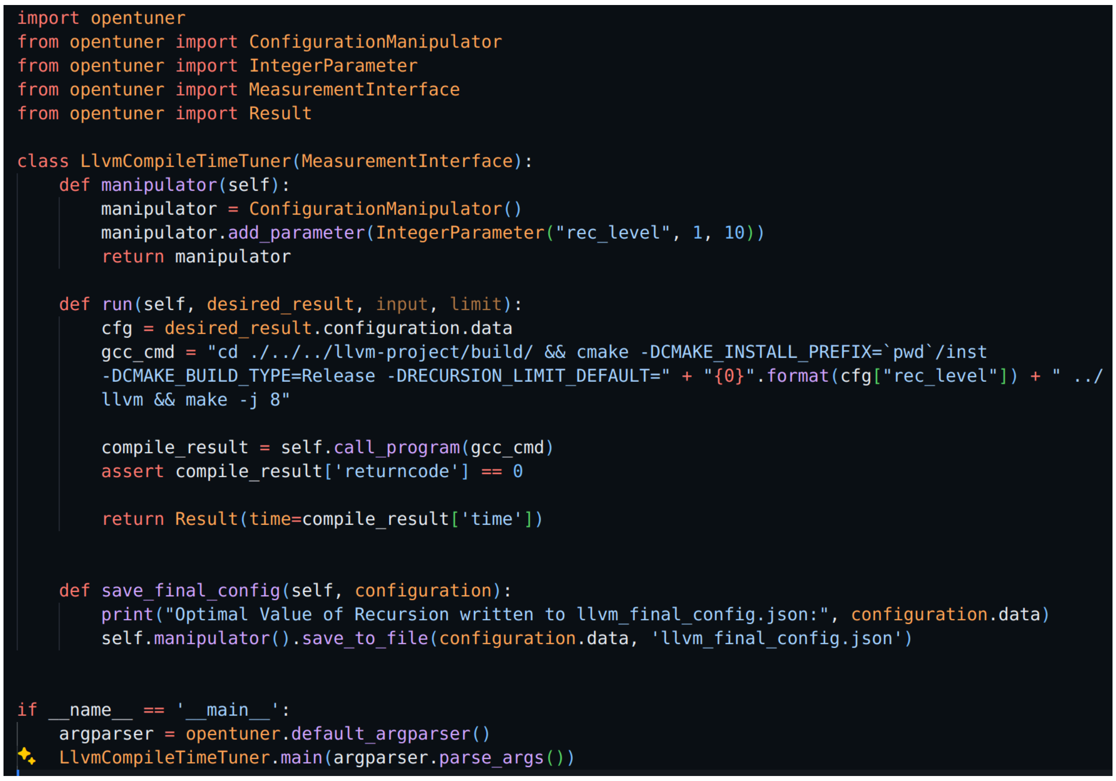
Here, RecursionLimit in llvm/lib/Analysis/InstructionSimplify.cpp can be tuned.



```cpp
/// The default value for MaxUsesToExplore argument. It's relatively small to
/// keep the cost of analysis reasonable for clients like BasicAliasAnalysis,
/// where the results can't be cached.
/// TODO: we should probably introduce a caching CaptureTracking analysis and
/// use it where possible. The caching version can use much higher limit or
/// don't have this cap at all.
static cl::opt<unsigned>
    DefaultMaxUsesToExplore("capture-tracking-max-uses-to-explore", cl::Hidden,
                            cl::desc("Maximal number of uses to explore."),
                            cl::init(100));

unsigned llvm::getDefaultMaxUsesToExploreForCaptureTracking() {
  return DefaultMaxUsesToExplore;
}

CaptureTracker::~CaptureTracker() = default;

bool CaptureTracker::shouldExplore(const Use *U) { return true; }

bool CaptureTracker::isDereferenceableOrNull(Value *O, const DataLayout &DL) {
  // We want comparisons to null pointers to not be considered capturing,
```

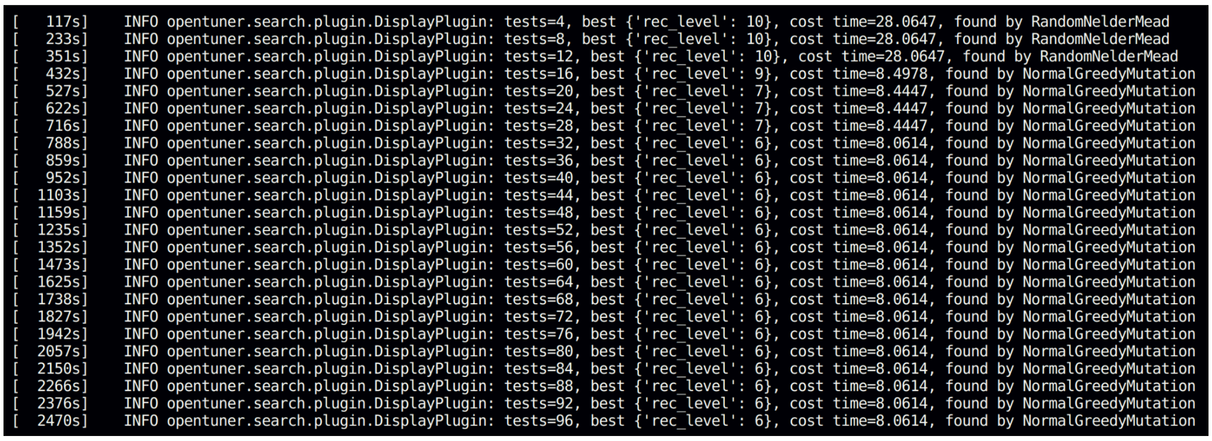
Here, cl::init(val) in llvm/lib/Analysis/CaptureTracking.cpp can be tuned.

My next step would be to use profiling software like OpenTuner to fine tune these values to get the optimal compile_time for the ComPile dataset. Here is the script I wrote while conducting an initial study :



```python
import opentuner
from opentuner import ConfigurationManipulator
from opentuner import IntegerParameter
from opentuner import MeasurementInterface
from opentuner import Result

class LlvmCompileTimeTuner(MeasurementInterface):
    def manipulator(self):
        manipulator = ConfigurationManipulator()
        manipulator.add_parameter(IntegerParameter("rec_level", 1, 10))
        return manipulator

    def run(self, desired_result, input, limit):
        cfg = desired_result.configuration.data
        gcc_cmd = "cd ./../../llvm-project/build/ && cmake -DCMAKE_INSTALL_PREFIX=`pwd`/inst
        -DCMAKE_BUILD_TYPE=Release -DRECURSION_LIMIT_DEFAULT=" + "{0}".format(cfg["rec_level"]) + " ../
        llvm && make -j 8"

        compile_result = self.call_program(gcc_cmd)
        assert compile_result['returncode'] == 0

        return Result(time=compile_result['time'])


    def save_final_config(self, configuration):
        print("Optimal Value of Recursion written to llvm_final_config.json:", configuration.data)
        self.manipulator().save_to_file(configuration.data, 'llvm_final_config.json')


if __name__ == '__main__':
    argparser = opentuner.default_argparser()
    LlvmCompileTimeTuner.main(argparser.parse_args())
```

When I ran the script, I observed that while changing the Recursion Limit should not affect the compile time of LLVM that much, but playing with its value gave 6 as the optimal value. This value was reached using Algorithms like RandomNelderMead and NormalGreedyMutation. The Following are the results I obtained after running a script using opentuner for studying compile time.



```
[   117s]    INFO opentuner.search.plugin.DisplayPlugin: tests=4, best {'rec_level': 10}, cost time=28.0647, found by RandomNelderMead
[   233s]    INFO opentuner.search.plugin.DisplayPlugin: tests=8, best {'rec_level': 10}, cost time=28.0647, found by RandomNelderMead
[   351s]    INFO opentuner.search.plugin.DisplayPlugin: tests=12, best {'rec_level': 10}, cost time=28.0647, found by RandomNelderMead
[   432s]    INFO opentuner.search.plugin.DisplayPlugin: tests=16, best {'rec_level': 9}, cost time=8.4978, found by NormalGreedyMutation
[   527s]    INFO opentuner.search.plugin.DisplayPlugin: tests=20, best {'rec_level': 7}, cost time=8.4447, found by NormalGreedyMutation
[   622s]    INFO opentuner.search.plugin.DisplayPlugin: tests=24, best {'rec_level': 7}, cost time=8.4447, found by NormalGreedyMutation
[   716s]    INFO opentuner.search.plugin.DisplayPlugin: tests=28, best {'rec_level': 7}, cost time=8.4447, found by NormalGreedyMutation
[   788s]    INFO opentuner.search.plugin.DisplayPlugin: tests=32, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[   859s]    INFO opentuner.search.plugin.DisplayPlugin: tests=36, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[   952s]    INFO opentuner.search.plugin.DisplayPlugin: tests=40, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[  1103s]    INFO opentuner.search.plugin.DisplayPlugin: tests=44, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[  1159s]    INFO opentuner.search.plugin.DisplayPlugin: tests=48, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[  1235s]    INFO opentuner.search.plugin.DisplayPlugin: tests=52, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[  1352s]    INFO opentuner.search.plugin.DisplayPlugin: tests=56, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[  1473s]    INFO opentuner.search.plugin.DisplayPlugin: tests=60, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[  1625s]    INFO opentuner.search.plugin.DisplayPlugin: tests=64, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[  1738s]    INFO opentuner.search.plugin.DisplayPlugin: tests=68, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[  1827s]    INFO opentuner.search.plugin.DisplayPlugin: tests=72, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[  1942s]    INFO opentuner.search.plugin.DisplayPlugin: tests=76, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[  2057s]    INFO opentuner.search.plugin.DisplayPlugin: tests=80, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[  2150s]    INFO opentuner.search.plugin.DisplayPlugin: tests=84, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[  2266s]    INFO opentuner.search.plugin.DisplayPlugin: tests=88, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[  2376s]    INFO opentuner.search.plugin.DisplayPlugin: tests=92, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
[  2470s]    INFO opentuner.search.plugin.DisplayPlugin: tests=96, best {'rec_level': 6}, cost time=8.0614, found by NormalGreedyMutation
```

A few other tools that I will use for benchmarking are :

- llvm-compile-time-tracker
- stats
- `Timer` and `TimeTraceScope class.`

My next step will be to introduce a wrapper around possible thresholds which can increment/decrement them. Internally, it will store what I did and if things like comparison yields true or false, in which case the threshold would be hit. This will reduce some of the guesswork. I will then further refine it to make it specific to a use case e.g., one for decreasing counters, one for upper bounds, etc.

I will also use the llvm-test-suite to benchmark the generated build on tests given in the suite and compare the results by storing them in a json file.

Then, I will move towards more large scale studies using ComPile for IR. This contains information regarding the raw bitcode that composes the module. This can be written to a .bc file and manipulated using the standard llvm utilities like opt or passed in directly through stdin if using something like Python's subprocess. I will run the programs and look at statistical outputs and other signals. Further, since the LLVM test suite allows us to get stats output, compile time and runtime numbers for a given compiler or compiler options, I will use it to profile different thresholds.

The Last part comes down to profiling and what/whether different profiles should be introduced. A good example of this is that maybe a counter is useful to reduce compile time but modifying it gives a much better vectorization percentage. That will be a good candidate for some sort of user-facing trade-offs, aka. Profiles. I can further pick “compile fast” or “executable fast”, in addition to O1/2/3 etc.

Finally, I will gather the results from all the above experiments and profile/fine-tune different thresholds.

TimeLine

Dates	**Task**
	Pre GSOC Period
April 3 - April 30	● I will contribute to LLVM by solving the remaining issues assigned to me. ● If time permits, I will try to solve issues targeting LLVM.
	Community Bonding Period Begins
May 1 - May 15	● With the help of my mentor, I’ll help to familiarise myself with the community and the codebase. ● I’ll discuss potential ideas for profiling and refine the project goals.
May 16 - May 26	● Explore various profiling tools and ensure that they are in sync with LLVM ie. They can be used for this project. I would then discuss with my mentor which of them can be potentially used further.
	Coding Period Begins
May 27 - June 15	● An In depth exploration of the llvm codebase ● Finalising the list of thresholds to be tuned / profiled.
June 16 - July 7	● Use tools like OpenTuner to set timing benchmarks. ● Explore other tools like stats to get more detailed information about the effect of changing a threshold. ● Wrap up current work and document it properly to discuss with mentors about current progress and possible approach changes.
July 8 - July 12	**Midterm Evaluations**

July 13 - Aug 1	● Introduce a wrapper class and use it to manipulate different threshold values. ● Explore the llvm-test-suite and get results by running opt on bitcode files. ● Start Large scale studies using Compile dataset.
Aug 2 - Aug 18	● Continue with benchmarking of thresholds using Compile dataset. ● Profile different thresholds and explore whether or not different profiles should be introduced.
Aug 19 - Aug 26	● Solve any pending or last minute issues and prepare for the final evaluation.
Aug 27 - Sep 2	**Final Evaluations**

Personal Background and Contributions

Overview:

I am a sophomore in Indian Institute of Technology (IIT) Roorkee, pursuing a major in Computer Science and Engineering (CGPA 9 / 10). I have loved tweaking with codebases, finding out what makes them tickle and understanding all the smart optimisations that have been done in it ever since I got to know about open source. At first I was blown away how this great piece of software is not proprietary but later I realised just how important a pillar it really is. My history with LLVM dates back 1 year when I first used it to make my own custom language. It was a pretty fun experience for me and since then I have tried my best to dive deep into system programming and development in general. I have participated in a lot of CTFs and hackathons and also won several of them. Apart from that I am a part of BlocSocIITR (A student run technical group in our college focusing on scalable decentralised applications and cryptography) and VLG(A student run technical group focussing on Deep Learning). I was also a part of MDG (A student run technical group focussing on Web Development) but later left it to pursue my interest in Compilers and Scalable systems.

Prior compiler and compiler-related experience:

- Implemented a custom language in my first year using the LLVM API and have a rough understanding of compiler architecture.
- Following the philosophy of “You only understand something when you implement it”, I decided to make a bootstrapped compiler myself. So I started following this :

A compiler writing journey.

Prior contributions:

PR	Status
LLVM	
#85940 [libc][math][c23] Implement canonicalize functions	Merged
#85628 [libc] Implement fileno on Linux	Merged
#86928 [libc][POSIX] implement fseeko, ftello	Merged
#78338 [Clang] Fix : More Detailed "No expected directives found"	Merged
#81183 FIX : bugprone-too-small-loop-variable - false-negative when const variable is used as loop bound	Merged
#84903 [DAG] Matched FixedWidth pattern for ISD::AVGFLOORU	Merged
#85031 [DAG] Matched Fixedwidth Pattern for ISD::AVGCEILU	Merged
#81015 [Clang][OpenMP] Fix `!isNull() && "Cannot retrieve a NULL type pointer"' fail.	Merged
#81002 [Github Automation] Update Execute command	Merged
#85682 [libc] Added tablegen definition for fileno	Merged
#86924 [libc][math][c23] Fix X86_Binary80 special cases for canonicalize functions.	Merged
#87103 [libc][math][c23] Fix impl and tests for X86_80 canonicalize function.	Merged

#84609 [ADT] Add implementations for mulhs and mulhu to APInt	Approved but later Closed
#84639 [DAG] TargetLowering::expandABD - investigate alternative expansions	Helped close the issue
#86340 [InstCombine]: Missed Optimization, treat the disjoint Or as an add	Open
#86435 [DAG] Added m_AnyBinOp and m_c_AnyBinOp in SDPatternMatch.h	Draft
Others	
#487 feat : Add positive and negative trait impl tests for SIMD types	Merged
#18 Fix - Logic Error in p_atan2 and atan1to1	Merged
#2562 Replaced Deprecated Function	Merged
#2563 Added more tests for PublicKey::from_str	Merged
#46 Ripemd160 implementation	Open
#89 Updated R1CS for sha256_maj and sha256_ch	Open

Note: I am actively working on more issues in LLVM and will update the above as and when the PRs are merged/approved.

Other Projects:

I have also worked on several **projects**, few of which I am proud of are :

1. **RISC V-Instruction-simulator :**

    - Repo Link
    - Made an assembler for RISC V instruction set

- Implemented a working 5-stage pipeline simulator for running RISC instructions.
- Implemented a Write-Back Allocate Cache with LRU Replacement Policy and integrated it with the simulator

2. **HorseRiders:**

    - Repo Link
    - Implemented a complex maths and Fast Fourier Transform Library for Huff and EVM Assembly
    - My major work was to get the Fourier Transform algorithm work within an acceptable range of error in a constrained environment with limited library usages.

3. **Zenith-Zealots:**

    - Repo Link
    - Implemented a game similar to Candy Crush but the reduction function is implemented using quantum gates.
    - My major role was to deal with the python API for rendering the game graphics and reviewing the gate implementation.

4. **ShieldFi:**
    - Repo Link
    - Implemented a protocol that can give zero knowledge proofs about whether a contract in the EVM has a bug or not.
    - My major role was to implement the state where prover and verifier interact ie. generating and verifying the proof.

5. **FlockChain:**
    - Repo Link
    - A Federated Learning protocol built on Proof of Stake to establish Economic Security in the network.
    - The architecture is built on top of micro-rollups to provide verifiable off-chain computation for state management and providing slashing conditions.

6. **Netra:**
    - Repo Link
    - Hardware based object detection and Text to speech aimed to ease the daily life of blind people.
    - My major role was writing code on raspberry PI for object detection.

7. **FlivGen:**
    - Repo Link
    - A Federated Framework for Training Differentially Private Deep Convolutional Generative Adversarial Networks
    - I learned a lot in this project. Using cryptographic techniques for making AI secure, making the model learning scalable by implementing a multi party

computation architecture and the internals of pytorch.

8. **zKFL:**
   - Repo Link
   - A unique way to train DL models using decentralised computing & Zero-Knowledge proofs for enhanced security & faster computations based on trustlessness & ultra-privacy.
   - My major role was to write a ZK-Circuit for the aggregation algorithm used in Federated Learning Architecture.

Achievements:

1. **EthIndia 2023:** I participated in one of Asia's biggest Ethereum hackathons, ETHIndia in 2023 where we were the top 12 world finalists and the **winners** of the ETHIndia'23 Hackathon.
2. **Circuit Breaker 2024:** I participated in an EthGlobal hackathon focused on Zero-Knowledge Cryptography. I was runner up in the best use of ZK in an ML context track.
3. **Huffathon 2023:** I participated in the first Huff language hackathon and was one among the 7 winners worldwide. Huff is a low-level programming language designed for developing highly optimised smart contracts that run on the Ethereum Virtual Machine (EVM)
4. **Other Achievements:** Apart from this, I have participated and won in several college level hackathons and CTFs. Apart from this I enjoy participating in global CTFs with InfoSecIITR who generally manage to get in the top50.

knowledge of programming languages:

I am well versed in C , C++ and Python and can program freely in these languages, as should be evident from my projects and open source contributions. While my rust is a little rusty at the moment. Though I am working on a ripemd160 hash function implementation for the Bellpepper repo. I think that by the time GSOC starts, I should be comfortable with it also.

Other Commitments and post GSOC

My University's End Term Examination ends on 7th May after which the Summer Break starts. I have no prior commitment or planned holidays during the break and I can easily devote 30+ hours per week for this project. I might have to travel to my hometown which may cost a day or two in between. My Classes will restart on 16th July after which I can devote 20+ hours to the project. I have made the timeline for the project accordingly where I propose to do most of the heavy lifting before mid term evaluation. Given the size of the project, I firmly believe that my current allocation of time would definitely help finish the project on time and most likely will leave space for buffer weeks also.I generally work from 4PM IST (UTC + 5:30) to 12PM (UTC + 5:30) and I plan to give 5-6 hours every day for 6 days with sunday being my off day. Though I am quite flexible in my working hours and can change them according to the mentor’s convenience.

There are very few chances that some things might go unimplemented as 12 weeks is enough time for the project, but still, If it happens, I’ll complete them post-GSoC, during the extended timeline period .I’ll keep contributing to LLVM to the best of my ability and participate in the community's discussions.Regardless of GSoC, I would love to engage in discussions with the LLVM community to get exposure to new ideas. I would love to be of any help even after the GSoC period.

Thanks a lot for taking out the time to read my proposal. If you have any queries, please feel free to reach out to me on discourse or through mail.