Google Summer of Code 2020

Chapel
Project - Protocol Buffers Integration

Introduction

I am Aniket Mathur, an Electronics and Communication Engineering undergraduate student at the **Indian Institute of Technology Roorkee**. I am pursuing Bachelor of Technology and currently in my third year of study. I got introduced to the world of programming languages and software in my first year, and since then I have made efforts in and explored various fields of Computer Science. The fields that capture my interest are Operating Systems, Computer Networks, Database Management, and Information Security. I am also proficient and have experience in the field of Software development (Android and Web development), with a handful of experience in Machine Learning as well.

Relevant course work for the project

- Object-Oriented Programming
- Data Structures
- Computer Architecture and Microprocessors
- Operating Systems
- Computer Networks

Motivation

Why you wish to participate in the Google Summer of Code?

Being a software enthusiast from my first year I always loved to think and develop ideas of my own. My interest in web development introduced me to Git and Github which, for me, ultimately paved a path towards open-source. It is really satisfying and worth it to contribute to big open source organizations, as your work directly affects the open-source community and a large number of people using that software.

Last year I was selected as a **Google Summer of Code student for Sugar Labs**. Over the summers it was a great learning experience for me, so I decided to apply this year as well. I think that Google Summer of Code is the best way to learn how to work in an open-source community where you have a number of people from different parts of the world working and building code collaboratively. The three-month duration of GSoC will also allow me to hone my current skills as well as acquire a new set of them under the guidance of quality

mentoring organizations and best mentors. The remote working policy of GSoC makes it the best program a college student can get involved in during their summer holidays for a great learning experience.

Why you wish to work with the Chapel project in particular?

The Chapel project is a great opportunity for me to contribute to the field of programming languages and Distributed Computing. I have always been keen on how a programming language is built and its libraries are developed and always wanted to contribute to one of them.

While contributing to Chapel, I have seen that the developer team of Chapel discusses issues very well and an equal weightage is given to every contributor's opinion. The contributors are given an equal opportunity to propose their ideas and discuss them with the developer team.

Also Chapel has great mentors, who are always willing to provide the best possible help. The mentors are very responsive, friendly and knowledgeable. This kept me motivated and enthusiastic towards contributing.

What do you hope to learn over the summer?

Over the summers I will get to learn about Protocol Buffers and their implementation in detail. The valuable experience of working on this project will get my skill set furnished with the help of my knowledgeable and experienced mentors. Also my last year GSoC project involved the majority of the changes in the existing code-base, this project will provide me an experience of starting to code from scratch.

Language & Availability

How well can you comprehend and understand English? How strong is your written English?

My first language is Hindi but I am proficient in speaking, reading, writing, and understanding English as well. I have been taught English since my school time. I have also done a Communication Skills(English) course at my university in which I scored a grade point of 10/10.

Do you have any other commitments for the summer period? Do you have planned vacations?

My summer vacations are starting from 4th May to 19th July, and the official GSoC period is from 4th May to 31st August. In vacations, I can give about 45-50 hours per week and after college starts, I will be able to manage 40-45 hours a week. I am also free on weekends and will keep the community updated about my progress. **I have no other commitments for the summer vacations**, so I can devote most of my time to GSoC.

Contact

Name	Aniket Mathur
Github	Aniket21mathur
Email	amathur@ec.iitr.ac.in
Location	Roorkee, Uttarakhand
Timezone	IST(UTC+5:30)

The timezone will remain the same during the summers. The time I will be comfortable working-

- UTC 0430 - 0730 hrs (IST 1000 - 1300 hrs)
- UTC 0830 - 1330 hrs (IST 1400 - 1900 hrs)
- UTC 1430 - 2030 hrs (IST 2000 - 0200 hrs)

I can start/end my day a couple of hours earlier/late.

Coding experience

Languages- C, C++, Chapel, Python, JAVA, Ruby, Javascript, Php, SQL, HTML, CSS, Verilog, Matlab and Shell programming.

Language	Experience(years)	Level (1-10)
Python	2.5	10
C/C++	2.5	10
JAVA	2.5	8
Javascript	2	7
Php	1.5	6
Ruby	1	5

Chapel

I started coding in Chapel a couple of months ago. To become familiar with Chapel, I did the following ;

- Went through the Learn X in Y minutes docs for Chapel, and played with the coding examples side by side.
- Went through the official online documentation.
- Followed some Youtube lectures on Chapel.
- I made some pull requests to the Chapel project(listed below).
- Worked with some modules of Chapel like **ZMQ**, **LinearAlgebra**, **Random**, **Map**, **Math**, **DistributedBag** and **DistributedDeque**.

As a result, I am quite comfortable in writing and understanding code in Chapel now. I will continue to learn Chapel for the next 1 month as well.

Projects

I am well versed in writing and understanding code in C/C++. Being a member of **InfoSecIITR**(Information Security Group), I am also familiar with **reversing** and **pwning** which includes understanding compiled C/C++ code. This provides me a better understanding of C/C++ structure and libraries. I also write code for competitive programming in C++, I have experience working with both advanced and basic data structures in C++.

Projects in **C/C++**

- A simple **Search Engine for Library**, implemented in C++.
- A wrapper taking data from a **Verilog** code and interacting with **Bash** to perform some operations.
- Implemented **Round Robin Scheduling** in C.
- Implemented **First Come First Serve(FCFS) Scheduling** in C.

Projects on **Parallel/Distributed** Computing

- A basic implementation of **Secure Data Transfer** using Image Steganography.
- A **Web System** with DNS server, Web server, and client program.
- A **Storage service** using GET, PUT and DELETE operations.

Projects on **Compiler Development**

- Implemented a **Lexical Analyser**, a **Parser** and a **Transpiler** based on Javascript following this tutorial.
- Implemented **Recursive Descent Top-Down** parsing.

Protocol Buffers

To get familiarize with protocol buffers, I did the following:

- Get to know what are protocol buffers from the official Google documentation.

- Followed some lectures and videos on Youtube.
- Went through the protocol buffer language guide for the **proto3** version.
- Went through the protocol buffer tutorial for some languages.
- Went through the protocol buffer API reference.

I have also **installed the protocol compiler** on my system for C++ and multiple other languages following the guide. I am playing around by compiling some examples with different languages and examining the output code to get a better understanding of protocol buffers. I have also implemented basic protobuf models for Python and C++ by creating a link with the protobuf files.

Familiarity

- **git/make-** I am using these tools for almost 2 years in my development projects. I am quite comfortable using them.
- **gdb/valgrind-** Being an information security enthusiast, I use these debuggers for reversing compiled files and finding out any memory leaks.
- **gcc-** I use the gcc compiler system for compiling the C++ code that I write, and is quite familiar with it.

Experience

I am exploring the field of software development for 2.5 years now. In that span of time, I have worked with a number of developers having different skill sets and expertise. My major involvements are-

Sugar Labs (Member)
I am an active member of the Sugar Labs organization. Sugar Labs is a community-run software project whose mission is to produce, distribute, and support the use of Sugar, an open-source desktop environment and learning platform. Sugar Labs is a member of the Software Freedom Conservancy, an umbrella organization for free software projects. I am contributing to Sugar Labs since December 2018. I was also a student developer under the **Google Summer of Code 2019** program for Sugar Labs. I was also a mentor in **Google Code-in 2019** from Sugar Labs. Being a member of this great organization, I have learned open-source ethics, opening and reviewing pull requests, discussing on threads and submitting issues.

SDS InfoSec (Member)
I am also a member of SDS(Software Development Section) InfoSec, a group of information security enthusiasts from IIT Roorkee. We actively participate in CTF(Capture the flag) competitions and organize talks related to information security. I have also designed various CTF challenges for https://backdoor.sdslabs.co. Being a part of this group I learned that quality software can only be built if you are well versed in exploiting and finding bugs in it.

Institute Alumni Relations Cell (Joint Secretary)
I am the Joint Secretary of the technical team of the cell working on maintaining the existing and developing new official portal for alumni of the Indian Institute of Technology, Roorkee. Being a part of this team, along

with **web development skills**, I have learned **how to lead** and make a group of people work together as a team.

The biggest project that I have worked on is **Port to Python 3** as a student developer under **Google Summer of Code 2019** for **Sugar Labs**. The aim of this project was to extend the compatibility of the sugar environment to Python 3. The major parts of the project were to port all static telepathy bindings to TelepathyGLib, port core sugar (sugar-toolkit-gtk3, sugar-desktop, sugar-datastore, and sugar-artwork), sugar activities and gwebsockets(A WebSocket server written in python. It uses GIO for network communication and hence it easily integrates with the GLib mainloop) to Python 3. The project involved working with Python and its libraries like telepathy, TelepathyGLib, dbus, six and tools like **gdb/pdb**, **dbus-monitor** for debugging and Wireshark for analyzing packets emitted from web sockets.

Along with the various technologies mentioned above, from this project I learned how to work in a disciplined manner. Submitting weekly reports, giving daily updates, group discussions and deadlines were a part of it. This project helped me to improve my code quality and developing standards. My major role in this project was to maintain smooth working and to have good progress over time by keeping a great interaction with my mentors. The entire report of my work can be found here.

Open-Source

I have been contributing to open-source since 2018. The projects I have worked upon are public on my Github page. My major open source contributions are to **Sugar Labs**, which can be found here. I have also contributed to **mushorg(snare/tanner)**, **DefectDojo, syslog-ng, IARC** and **InfoSecIITR** organizations.
I started contributing to **Chapel** from January 2020, below is the list of contribution I made to the organization-

Pull Requests(PRs)

Pull Request	Fixes	Description
#14828 (Merged)	**#14725**	Extend the LinearAlgebra.eig function to support complex-valued matrices as well.
#14857 & #15156 (Merged)		Grammar fixes in the chapel documentation.
#15154 (Open)	**#14865**	Implement a version of Random.choice function that returns indices instead of elements.
#15145 (Open)	**#5753**	Improve _expBySquaring Algorithm in LinearAlgebra module.
#15135 & #14872 (Merged)	**#11417**	Add throws tags to functions in Random, Path and Spawn modules.
#14881 (Merged)	**#13336**	Modifies the "Updating mason-registry" message to specify

		the registry name
#15140 (Open)	**#14388**	Implements parallel merge shuffle for Random.shuffle.
#14869 (Merged)	**#13257**	Add RVF pragma to DistributedDeque and DistributedBag.
#14863 (Closed)	**#11587**	Change language_level to default.
#15221 (Open)	**#15218**	Standardize string/bytes buffer allocations in ByteBufferHelper.
#15222 (Open)	**#15217**	Change implicit type to explicit type in ByteBufferHelper.
#14964 (Open)	**#14571**	Redesign getCurrentTime function.

I am also working on building a **gzip** library for Chapel.

Issues

Issue	**Description**
#15159	The array reindex function fails for a domain with negative stride.
#14963 (Duplicate)	Zero-based indexing for tuples and other cases.
#15227 (Duplicate)	Support type coercion within the array literals.

Survey

Had you heard about Chapel before the Summer of Code? If so, where? If not, where would you advise us to advertise?

Yes, I heard about Chapel before Summer of Code. One of my friends has an intern at Australian National University, his project includes the use of Chapel for memory management. He told me about Chapel and its benefits which motivated me to start contributing to Chapel and thus pursue to apply for GSoC in Chapel.

Though, the biggest way to advertise is to increase the use case of the language and telling the users what's unique about it. Conducting conferences, holding workshops and lectures is also a way. Sometimes making something creative using the language, like a game with a unique theme, also makes it popular among the users.

What was the first question concerning Chapel that you could not find an answer to quickly?

The first question that I could not find an answer to quickly was how to install and use LAPACK/BLAS packages for compiling LinearAlgebra. I found the answer to this question by discussing it with Ben on Github thread and self-exploring through Google search.

What will keep you actively engaged with the Chapel community after this summer is over?
I will keep myself engaged with Chapel after this summer is over. Great response and guidance from quality developers here will keep me motivated. I would keep working on and improving the protocol buffer library for Chapel, after the summer. I will also work on improving the user documentation for the same.

Are you applying to any other organizations for this year's Google Summer of Code? If so, what is the order of your preference, in case you are accepted to multiple organizations?
No, I am not applying to any other organization for this year's Google Summer of Code.

Prerequisites

What operating system(s) do you work with?
I work with Ubuntu 18.04 and Kali Linux.

Are you able to install software on the computer you plan to use?
I am planning to use Ubuntu 18.04 for this project. I am using Ubuntu for almost 2 years now. I am comfortable in installing any software on my system. I was also able to successfully set up Chapel in my system.

Will you have access to a computer with an internet connection for your development?
Yes, I will have access to a computer with a good internet connection for the development period.

Self-assessment

What does useful criticism look like from your point of view as a committing student?
As a committing student, I think one should accept useful criticism, and should use it as a guide for improvement and better implementation. There is no harm in accepting if you are wrong.

What techniques do you use to give constructive advice? How do you best like to receive constructive feedback?
I like to give any kind of advice by side by side reviewing the current work and pointing out the advantages and disadvantages of the current implementation, also highlighting what benefits we can achieve by doing it the other way. I like to receive constructive feedback in the same manner, it helps in having healthy and productive communication.

What is your development style? Do you prefer to figure out/discuss changes before you start coding? Or do you prefer to code a proof-of-concept to see how it turns out?
I like to prefer to code a proof-of-concept and then submit it for review, making further changes as suggested. I have also demonstrated this in my pull requests to the Chapel project. Though I have learned a valuable thing from the Chapel developers that a design issue should be discussed before start coding it.

The task

The task here is to create a Chapel library that works with basic Protocol Buffers functions and to implement a Chapel plugin for the protocol buffers compiler. Also, do a demonstration of this tool with protocol buffers.

The key elements that I will focus on during this project include-

- Creation of a well structured and designed Chapel library compatible with basic functions of the protocol buffers.
- Implement a Chapel plugin for the protocol buffers compiler.
- Example integration with protobufs for demonstration of the newly implemented tool.
- Create documentation for the above library with examples.

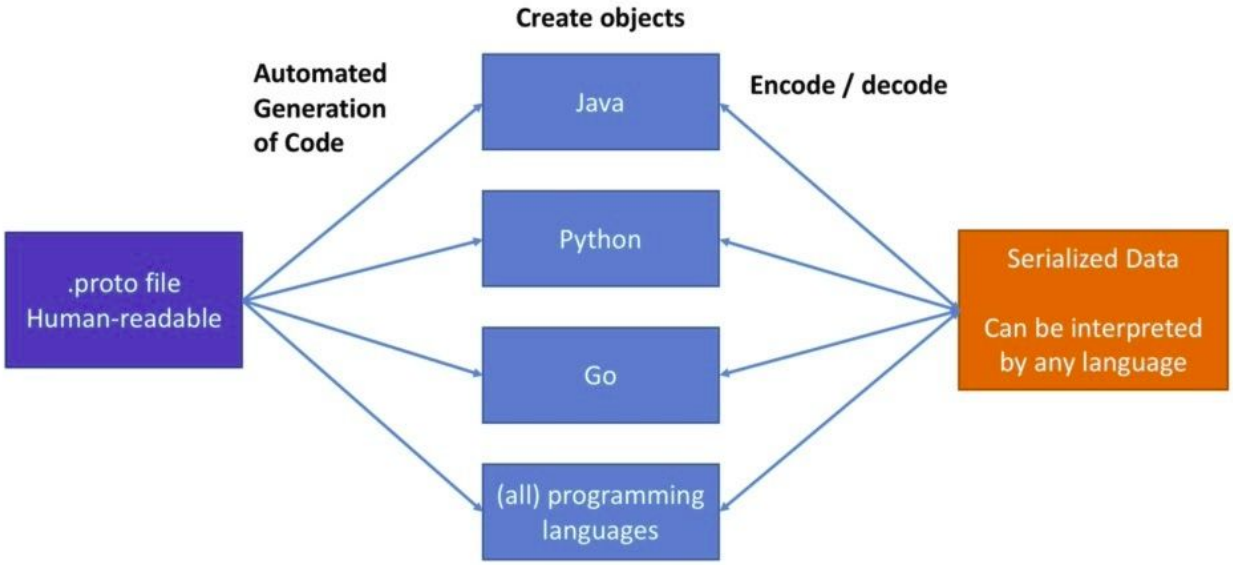
Description

Protocol Buffers (a.k.a., protobuf) are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data.

Using protobufs

In order to make use of Protocol Buffers:-

- We need to define message structures in **.proto** files. Each file act as a description of data that might be transferred from one node to another or stored in data sources.
- After the definition of message structure, a compiler is needed to convert the language-neutral content to a specific language code of the user's choice.
- Depending on the output language, the messages in a .proto file are compiled into descriptor classes and methods.
- These objects can then be encoded/decoded into serialized data which can be interpreted by any language or system(Sometimes we interface with systems that for some reason cannot use protobufs).



Protocol buffer library

Following the language guide for the proto3 version, the protocol buffer library for any language is expected to-

- Should have modules to generate **message classes** from the message types defined in a .proto file with support for **singular**, **repeated**, **reserved**, and **enum** field types.
- Should support import of definitions from other .proto files. Should support the **Any** message type and the **oneOf** feature.
- Have support for **packages**. A package specifier in a .proto file prevents name clashes between protocol message types.

```
package foo.bar;
message Open { ... }
```

- May have support for proto **maps**. The protocol buffer map is **backward compatible**, it can be implemented using message type and repeated fields as well.

```
map<key_type, value_type> map_field = N;
```

Is equivalent to

```
message MapFieldEntry {
  key_type key = 1;
  value_type value = 2;
  }

repeated MapFieldEntry map_field = N;
```

- Having modules to support **JSON** mapping would be great, it will make it easier to share data between systems.
- May have modules to support message types with **RPC**(Remote procedure calls) support.

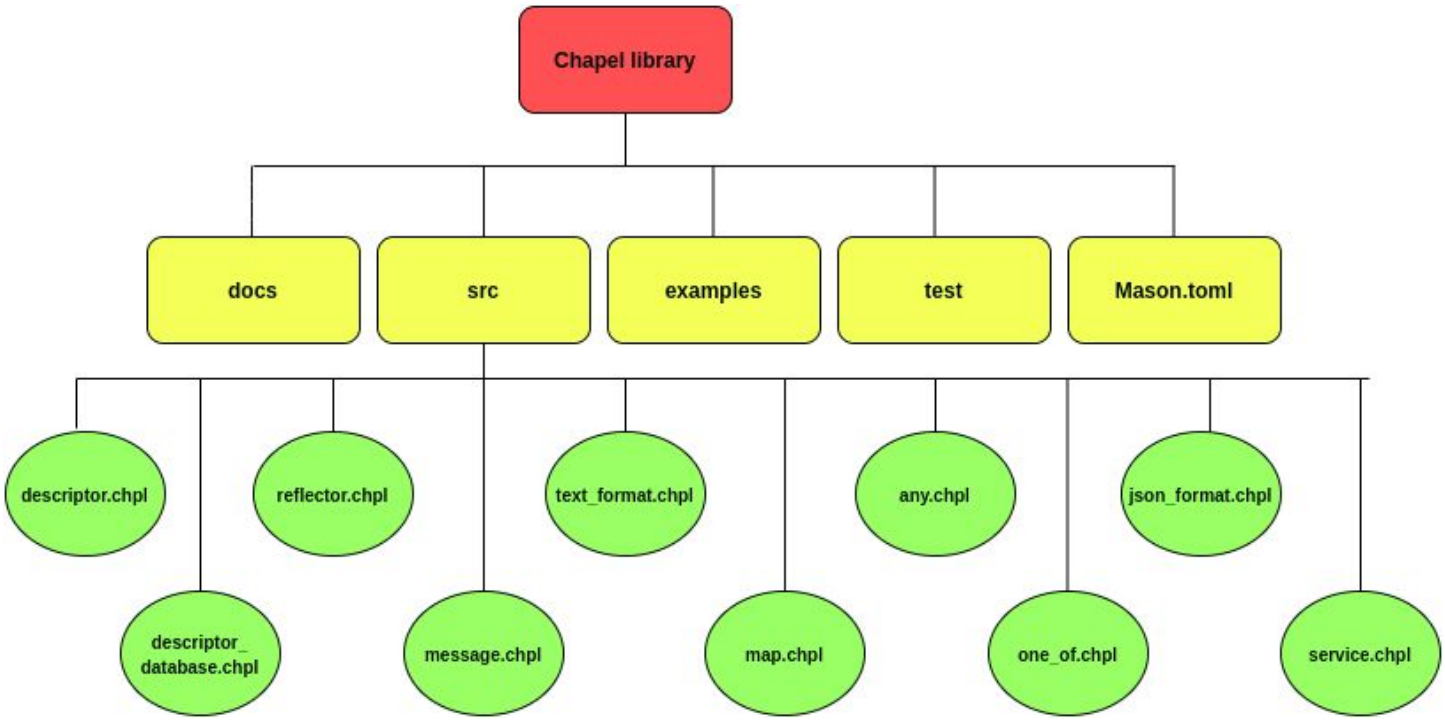
The implementation of protocol buffers is a bit different in Python as compared to C++, Java and other languages-

Python implementation	C++ implementation
A module with a static descriptor of each message type is generated which is used with a metaclass to create necessary Python data access at runtime.	A module is generated with classes for each message type along with accessor methods for each field defined.
The modules are organized according to their location in the file system, so the package directive is ignored.	The generated classes are wrapped inside a C++ namespace with the same name as that of the package.

Chapel library

I went through the sources of the third-party add-ons for protocol buffers of various languages. I also went through the official implementation for C++ and Python. Taking these as references, I have designed a proposed structure for the Chapel library. This structure contains most of the basic functions that should be present in the protobuf library.

(**This is a proposed structure, since it is a design issue I would like to discuss it with my mentor before finalizing things.**)



Components

- ***docs*** - documentation of the functions implemented in the library.
- ***examples*** - code examples for protocol buffers.
- ***test*** - tests for the implemented functions.
- **Mason.toml** - A manifest file, all the additional metadata about the packages as well as the dependencies are listed in this file.
- ***src*** - key source files for the protocol buffer library.
    - *descriptor.chpl* - This file will contain classes to describe the type of protocol message fields. A descriptor can be used at runtime to know what fields a message contains and what are the types of those fields. It will have descriptive classes for single, repeated and reserved fields.

- descriptor_databse.chpl - It will provide an interface to manipulate the descriptor database. It will help in improving the loading efficiency by building only particular descriptors that are needed.

- *reflector.chpl* - This file will contain a metaclass and helper functions to create protocol message classes from the descriptor objects during runtime.

- *message.chpl* - It contains an abstract base class for protocol messages with descriptors and reflection. It will have a messageLite class which will support a subset of features namely nothing that uses descriptors or reflection. The protocol compiler could be instructed to generate classes that only implement messageLite.

- *text_format.chpl* - Contains routines for printing protocol messages in human-readable text formats. It will have class useful in debugging and human editing of messages, printer class for scalar values into strings, class to find extensions, class to parse location, etc.

- *map.chpl* - This file will have helpers to support protobuf maps and will define the map container.

- *any.chpl* - The any message type let us use messages as embedded types without having their .proto definition. This file will have helpers to support the any message type.

- *one_of.chpl* - The oneOf feature can be used to save memory if we have a message in which at most one field will be set at the same time. They are like regular fields except that all the field share memory, and one will be cleared automatically as the other is set. This file will implement the oneOf feature.

- *json_format.chpl* - It will contain functions to print protocol messages into JSON format. Will have functions to parse a JSON format string as well as serialize a proto object to it.

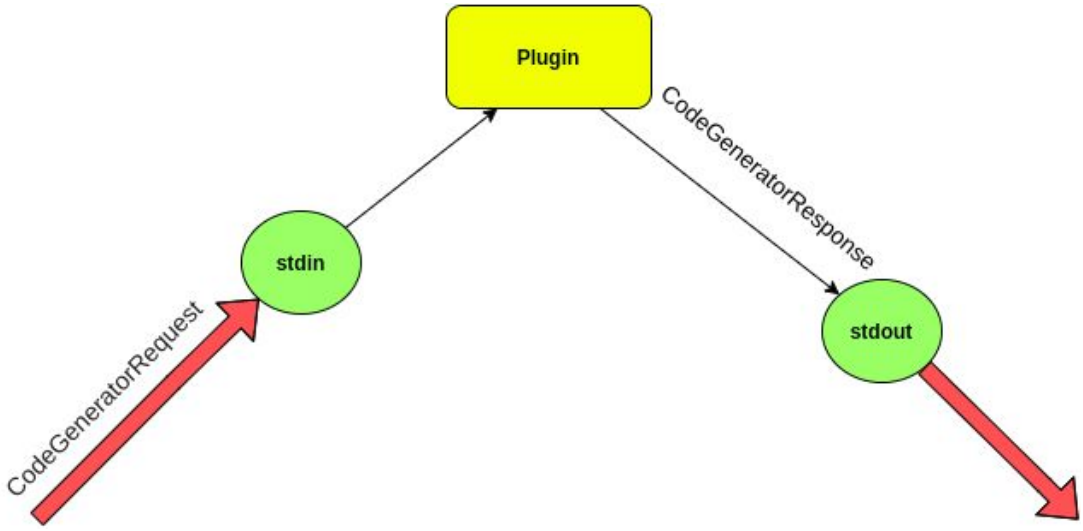
- *service.chpl* - This module will have the abstract interface for RPC(Remote Procedure Call) services.

Chapel plugin

The Protocol Buffers Compiler(**protoc**), can be extended to support new languages via plugins. A plugin is just a program that reads a CodeGeneratorRequest protocol buffer from standard input and then writes a CodeGeneratorResponse protocol buffer to standard output.

Plugins written using C++ can directly use plugin.h. The core part of PluginMain is to invoke the given CodeGenerator on a CodeGeneratorRequest to generate a CodeGeneratorResponse. But plugins in other languages will need to deal with the raw protocols defined in plugin.pb.h.

A simple representation of how a protoc plugin works.



A Chapel plugin can be implemented following a similar schema. The implementation of plugins for other languages can also be viewed for a general idea.

After the integration of the Chapel plugin with the protoc, Protocol Buffers can be used with Chapel. This will come up with scope for many features like making interlanguage calls in Chapel, or creating a Chapel distributed in-memory key-value storage service

Demonstration

A pseudo demonstration of how compiling a **.proto** file with **--chpl_out** option with the protobuf compiler will produce an output chapel file after completion of this project. The library structure described above is used for the generated out.

file.proto

```
syntax = "proto3";

message FriendRequest {
  string name = 1;
  int64 age = 2;
}
```

compiling

```
protoc -I=$SRC_DIR --chapel_out=$DST_DIR $SRC_DIR/file.proto
```

file_out.chpl

A mock output file with pseudo-code for basic functions for the field types. There might be more functions and classes that will be implemented in this file.

```
use descriptor;
use message;
use reflection;

// Generated class for friend request message format
// It is derived from the Message class in the message module
class FriendRequest: Message {

  //Accessor methods for `name` field

  /*
  Returns the current value of the field
  Returns empty string if the field is not set
  */
  proc name(): string {};

  //Sets the value of the field
  proc setName(val: string) {};

  //Sets the value of the field with explicit string size
  proc setName(val: string, size: int) {};

  //Clear the value of the field
  proc clearName() {};

  //Accessor methods for `age` field

  /*
  Returns the current value of the field
  Returns 0 if the field is not set
  */
  proc age(): int {};

  //Sets the value of the field
  proc setAge(val: int(64)) {};

  //Clear the value of the field
  proc clearAge() {};

}
```

Write/Read Operations

We can create and populate instances of protocol buffer classes and then write them to an output stream. We can also read data from the created file.

write.chpl

```
use file_out;
use IO;

var friendRequest = new FriendRequest();

//set name
friendRequest.setName("Aniket");

//set age
friendRequest.setAge(20);

//write the object to a file
var myFile = open("out", iomode.cw);
var writingChannel = myFile.writer();

//the SerializeToString method will be implemented in message.chpl
writingChannel.write(friendRequest.SerializeToString())
```

It will generate a file **out** with the object written to it.

read.chpl

```
use file_out;
use IO;

var friendRequest = new FriendRequest();


//open the file to read the serialized string
var myFile = open("out", iomode.r);
var readingChannel = myFile.reader();
var s: string;
var flag: bool = readingChannel.read(s);

//the ParseFromString method will be implemented in message.chpl
friendRequest.ParseFromString(s);

//print the values.
writeln(friendRequest.name());
writeln(friendRequest.age());
```

The compiled file will produce the following output.

```
$ ./read
Aniket
20
```

JSON format
Sometimes we want to interface with a system that for some reason cannot use the Protobuf library, for this reason, the protobuf will also have an encoding/decoding option to different formats, for example, JSON. We can use the json_format.chpl module described in the above structure.

We will generate an object using *file_out.chpl* and will encode it into the JSON format.

encode.chpl

```
use file_out;
use json_format;

var friendRequest = new FriendRequest();

//set name
friendRequest.setName("Aniket");

//set age
friendRequest.setAge(20);

//create a json object
var jsonObj = json_format.encode(firendRequest);
```

The json of the above data will look like-

```
{
      "name" : "Aniket",
      "age" : 20
}
```

Motivation for the task

This task excites me because it comes under Language interoperability. It involves the exchange of data through serialized objects. Being an information security enthusiast I have always worked to find loopholes in such systems, so it would be a great experience to build a functional and secure library.

I have chosen this particular task because it involves working with compilers and creating plugins. This task will also provide me the experience of building a library from scratch.

This task will entitle me with the knowledge of how exactly plugins for big existing projects like protobufs are built. By the end of the summers, I will have a good understanding of how compilers work. I will also get the opportunity to work with experienced and knowledgeable Chapel developers and other fellow students and I will, for sure, get to learn a lot of things from them. This project will be a major advancement for me in the open-source community.

Timeline

Community bonding period	
4 May, 2020 to 31 May, 2020	● Continue contributing to the Chapel project. ● Continue to learn about protobufs. Discuss project design problems with mentors and get their feedback. ● Get involved with the Community.
Coding Period	
1 June, 2020 to 13 June, 2020	● Implement the **descriptor** modules(descriptor and descriptor_databse). ● Add tests for the modules.
14 June, 2020 to 17 June, 2020	● Implement the **reflector** module. ● Add tests for the module.
18 June, 2020 to 26 June, 2020	● Implement the **message** module. ● Add tests for the module.
27 June, 2020 to 30 June, 2020	● Add examples for the implemented **base** modules. ● Make bug fixes and code refactoring.
Phase One Evaluation	
1 July, 2020 to 7 July, 2020	● Implement the module for supporting the **OneOf** feature ● Implement the module for supporting **Any** message type. ● Add tests for the modules.
8 July, 2020 to 19 July, 2020	● Implement the **text format** module. ● Implement the **json format** module. ● Add tests for the modules.

20 July, 2020 to 26 July, 2020	• Implement the module to support protobuf **map**. • Add tests for the module.
27 July, 2020 to 30 July, 2020	• Add examples for the newly implemented modules. • Make bug fixes and code refactoring.
Phase Two Evaluation	
31 July, 2020 to 15 Aug, 2020	• Implement the Chapel plugin for the protocol buffer compiler. • Implement the module for supporting **RPC** services. • Add tests for the module.
16 Aug, 2020 to 23 Aug, 2020	• Demonstrate the newly implemented tool with an example integration using protobuf. • Document the complete project. • Make bug fixes and code refactoring.
24 Aug, 2020 to 31 Aug, 2020	• Buffer period to complete any pending tasks. • Keep fixing issues and do code refactoring. • Discuss with mentors the future aspects of the project.
Post GSoC period	
Continue to work on the project, complete any remaining work from the summers, open issues, suggest features and maintain the project.	

Contributor Agreement

I have already submitted a CLA. Here is a link for the same.

References

- https://developers.google.com/protocol-buffers/docs/overview
- https://developers.google.com/protocol-buffers/docs/proto3
- https://developers.google.com/protocol-buffers/docs/reference/other
- https://github.com/protocolbuffers/protobuf/tree/master/python
- http://github.com/eigenein/protobuf/
- https://www.baeldung.com/spring-rest-api-with-protocol-buffers
- https://codeburst.io/protocol-buffers-part-3-json-format-e1ca0af27774