Winter of Code 2.0

GDSC IIIT Kalyani

Project and Organization:

Organization : LLVM

Project : LLVM FLang

Category : Compiler Design

General Information:

Name: Harshil Jani

Email ID: harshiljani2002@gmail.com

Github Username: https://github.com/Harshil-Jani

Country: India

Primary Language: English

Work:

Differential Patch : https://reviews.llvm.org/D119141

Commit on Github : https://github.com/llvm/llvm-project/commit/bea53eead1de84a28affc6a7cbf88f87a258fed4

Exploration Phase

Firstly, Let's talk about how I came to know about WOC 2.0. In our Linux and Open Source group, once there was a post regarding this event. And, I love participating in multiple events. So, I keep things starred. One day, Peacefully, I visited the official site and saw it. Also in that duration I was exploring some of the GSOC Organisations as well. Suddenly, I found out that LLVM Organisation was also part of GSOC. Now, There were other organizations as well. But either they were on Web-Development or Machine Learning. And, I don't want to do Web Development so much because it is very much common thing these days. LLVM was about compilers, but I had no ideas about compilers. But this was Mid October and contributions were about to start from December. So, I thought the time would be sufficient for me to work on it. So, I just registered for the event from Devfolio.

Proposal Writing Phase and Community Bonding

This proposal writing phase was in November. Since, I had no idea about how the workflow would be but I knew how a proposal for GSOC is supposed to be crafted. There has to be a week by week rough lookup of what we would be implementing. But, I was completely blank about what I will be doing. So, I approached my Mentor **Nimish Mishra.** He told me not to worry about the Weekly Timeline and just mention the project details. He would understand it and just once consult with the organizers to avoid any sort of miscommunications. Finally, I made up my mind to the fullest to work for Compilers. Within a few days with little bit of research, I crafted out a proposal for myself. Here is my proposal if you wish to glance through it.

Proposal

Now, After this comes the community bonding period. By mail, I connected with my mentor on Discord to get things started. I got to know about the codebase we would be looking after and other basic stuff. I told, I am very new to this field and currently learning about compilers from the basics. Finally, I was being told to move the project at my pace and keep learning and updating. Eventually Coding Period Started.

Coding Phase

Now, The real thing starts. Once I went back to my mentor and told him that I am learning compiler theory and along with it I also have an eye on the issues in the llvm repository. But, I am gaining no sense of how we would be working. He suggested that I should work for some Semantic Check in OpenMP support for the Flang and provided me a sheet by OpenMP and asked to choose something to work on.

At my first, I selected the Atomic Construct So, He himself had a patch for this Atomic Construct. Now, I had to choose something else and I now selected map clause task. But that was not so beginner friendly, So we had to switch to some basic task in the map clause itself. Now, Everything was decided and I was told to build the project on my system. And, This thing almost kept me in my vain. I did every possible try to build the project but in LLVM we have Ninja build system Generator. This was something that required RAM as it have to link the Bindings to the system hardware. I was working on 4 GB RAM Laptop. And Each time I ran ninja build I was facing severe hangs on my Laptop. So, At once, We decided to switch to the Clang project, But I was unable to build Clang as well. And I have to postpone my journey. But, I was quite adamant about the compilers now. The efforts I put in learning it, I didn't want them to go in vain. I talked to the organizer of the event and concluded that, I can work with LLVM whenever I will have suitable hardware. It was expected to be with me by March.

But, In January end, due to more workloads I asked my parents for a new laptop and in a week or so I got a new one with 16 gigs RAM. So, Now I was all set to work for LLVM thing. I built the system myself, went again through the sheet of OpenMP and chosen some issue that suited my learning and informed the mentor about everything I can do now. He then, asked me to inform again to the organizers regarding the same, And we had started our work.

Let's talk about my tasks. I had to work on target enter data and target exit data constructs and in it, there was a device clause that was supposed to take only the non-negative integers and not the negative numbers. The exact statement was :

> **"The device expression must evaluate to a non-negative integer value"**

Now, My tasks were to find out where this **target enter data** and **target exit data** failed. I was expected to know what device expressions were and after knowing them in detail from the OpenMP Specification sheet and the Example sheet, I had to craft a test case where on passing the negative number inside the device clause of both the constructs it should compile with no errors returned. Now, This was the thing. There has to be some error on passing the negative number and that was what I took up as work.

There is a **target construct** that deals with the device data environments for handling variables created in data environment. This construct has some directives, target data which is **structured directive** or target enter data and target exit data which are **unstructured directives** and also **standalone**. Now use of unstructured directives is creation and deletion of data on the device at any point within host code. So, in the specifications sheet as mentioned we have **device** clause for unstructured data constructs. As arguments we can provide the device id as a non negative integer.

First I had to write a test case file and run it. Since it was not implemented beforehand, you will observe No error. Then your task is to *write code* to flang's codebase that detects the number is -ve and throws an error. Finally you add your test case to the test folder and submit the patch for review. That's the basic workflow.

I somehow, With the help of OpenMP example sheet managed to write a test file. At very first I was unable to compile the file as I have not exported the flang compiler to my system path. So, with the help of some guidance I did that and then got file compiled successfully. Now, Came the real implementation of how to stop things from going negative. Now, For this we need to have a look at the parse tree of the test file written by me. With the help of command I dumped the whole parse tree of the FORTRAN code. Now inside the llvm flang files there is one file named as "parse-tree.h" header file and we can look into this header file for understanding the different nodes of the parse tree. Now parse-tree.h will allow you to make a plan, like you can get **OpenMPStandaloneConstruct** within in you need to extract **OmpClause**, which is a **std::variant** so you may need to extract Device, then look at definition of Device to extract Scalar, and then Integer, and then the Expr and then check the sign

We had some brainstorming around **std::variant** and **OpenMPStandaloneConstruct** and figured out which function was working in our code for execution of the test file code that I wrote. Inserted some print statements within some functions and with some trial and error, found out the function responsible for working. It was the following function :

void OmpStructureChecker::Enter(const parser::OpenMPConstruct &x)

Now, Inside this function there was a need to write another function dealing with the sign of the Device expression. So, I created the function named

void CheckDeviceExpr(const parser::OpenMPConstruct &x)

Initially, I thought the function required taking every node element of the parse tree. But on further research it turned out that from just OpenMPStandaloneConstruct we can have access to OmpClause and from there extract the definition of Device clause . I somehow,

On the search from various files I found something known as **llvm::omp::clause::OMPC_device** This brought me closer to my implementation.

For the next few days what I was trying was to extract the **OMPC_device** clause somehow from the parser path into my function. But, I was unable to get any idea. I again tried a hit of trial and error and got myself tangled into 3 scenarios.

1. **if (llvm::omp::Clause::OMPC_device)** This won't work as this is clause and won't return any boolean values, So it failed to compile.

2. **if (FindClause(llvm::omp::Clause::OMPC_device))** Now here the main thing is that we are not using **void function_name::some_constructor(const our_arg)** but instead we use **void function_name (const our_arg)** So, FindClause won't be defined in it. It is pre-defined for some used constructors such as **::Enter ::ChecksOnOrderedAsStandalone** etc. So, this also failed.

3. **std::cout<<llvm::omp::Clause::OMPC_device** this however was the worst thing to do. It is a clause so it won't work this way. It needs some iterative prints.

Then I had again omitted the whole function and started staring codes again. Suddenly, I found some functions with valid comments. The first set of functions says // Use when clause falls under 'struct OmpClause' in 'parse-tree.h'. When we compare this with the parse Tree of Test File then we have Device Expression under Omp Clause. **#define CHECK_REQ_SCALAR_INT_CLAUSE(X, Y)** This Macro is what the Spec sheet says. Our argument is scalar integer which will have positive values checked already as seen in it's function definition. What I did was simply changed

CHECK_SIMPLE_CLAUSE(Device, OMPC_device) with **CHECK_REQ_SCALER_INT(Device,OMPC_device)** and the test file throwed me error as expected. So, Now I felt I made it. I thought this is it. My first thing in LLVM got completed. Here was the new twist tho. It just accepted the positive numbers. For me it was to allow non-negatives. So I had problems with passing 0 into this. Then, What I did is created a new function inside directive calls where passing 0 was allowed as well. And finally made it.

Then I created a patch for this and with the guidance from my mentor, got it uploaded on Phabricator where It was reviewed for couple of days with some minor changes which taught me more clean code writing. And then with the final revision it was accepted.

Above and Under, It was an extremely new experience for me to get into compilers. And Special thanks to Mentor who made things clear whenever I made some blunders or was gone blank. It is one of my best open source contributions. It added value to my learning. I loved it.