FreeCompilerCamp.org: Training for OpenMP Compiler Development from Cloud

Anjia Wang
Lawrence Livermore National Laboratory
Livermore, California, USA
University of North Carolina at Charlotte
Charlotte, North Carolina, USA
awang15@uncc.edu

Alok Mishra
Lawrence Livermore National Laboratory
Livermore, California, USA
Stony Brook University
Stony Brook, New York, USA
alok.mishra@stonybrook.edu

Chunhua Liao
Lawrence Livermore National Laboratory
Livermore, California, USA
liao6@llnl.gov

Yonghong Yan
University of North Carolina at Charlotte
Charlotte, North Carolina, USA
yyan7@uncc.edu

Barbara Chapman
Stony Brook University
Stony Brook, New York, USA
Brookhaven National Laboratory
Upton, New York, USA
barbara.chapman@stonybrook.edu

ABSTRACT

OpenMP is one of the most popular programming models to exploit node-level parallelism of supercomputers. Many researchers are interested in developing OpenMP compilers or extending existing standard for new capabilities. However, there is a lack of training resources for researchers who are involved in the compiler and language development around OpenMP, making learning curve in this area steep.

In this paper, we introduce an ongoing effort, FreeCompilerCamp.org, a free and open online learning platform aimed to train researchers to quickly develop OpenMP compilers. The platform is built on top of Play-With-Docker, a docker playground for users to conduct experiments in an online terminal sandbox. It provides a live training website that is set up on cloud, so anyone with internet access and a web browser will be able to take the training. It also enables developers with relevant skills to contribute new tutorials. The entire training system is open-source and can be deployed on a private server, workstation or even laptop for personal use. We have created some initial tutorials to train users to learn how to extend the Clang/LLVM and ROSE compiler to support new OpenMP features. We welcome anyone to try out our system, give us feedback, contribute new training courses, or enhance the training platform to make it an effective learning resource for the HPC community.

1 INTRODUCTION

Due to the increasing complexity of supercomputer node architectures for high performance computing (HPC), high level programming models are used to improve the productivity of using supercomputers. OpenMP is considered by many as the de-facto portable programming model for exploiting node-level parallelism for supercomputers. Compiler support for OpenMP has been added in many open source compilers, such as GNU compiler collection, Clang/LLVM, and ROSE source-to-source compiler frameworks, as well as vendor compilers from Intel, Cray, NVIDIA and AMD. More and more researchers are interested in conducting research using OpenMP as a vehicle in the area of parallel programming models, compiler technologies and computer systems. However, one of the major challenges in developing an OpenMP compiler and to extend OpenMP language is the steep learning curve of compiler implementation and the development efforts of adding compiler support for language extensions.

Fundamentally, compiler development is a complex and time consuming task. Although many cloud-based, online learning platforms [3, 21, 25, 30, 31] have been created for computer science education, focusing on entry-level programming courses, there is a clear lack of such resources to teach compiler development. Even with the developer manuals of a compiler framework, it is difficult for beginners to teach themselves how to modify compilers which contains millions of lines of code. Training beginners by proficient compiler developers consumes lots of time, human efforts and cost, which is not scalable in the long term.

In this paper, we introduce an ongoing effort, FreeCompilerCamp.org, a free and open online learning platform aimed to train researchers to quickly develop OpenMP compilers and help them learn the skills of compiler development. FreeCompilerCamp.org has several distinct features: 1) It allows anyone who is interested in developing OpenMP compilers to learn the necessary skills for free; 2) A live training website is set up so a web browser and an Internet connection are the only requirements for anyone to take the training; 3) It enables those who have the relevant skills to contribute new tutorials; and 4) The entire training system is open-source so it can be be deployed on a private server, workstation or even personal laptop.

The remainder of the paper is divided as follows: Section 2 gives background information for our work. Section 3 explains the challenges faced in giving compilers training and our solutions. Section 4 presents the implementation of the framework. Section 5 gives an overview of the design of the tutorials with a few examples. Section

https://doi.org/10.22369/issn.2153-4136/11/1/9

6 covers work related to this paper. Finally Section 7 consists of the conclusion and our future plans.

2 BACKGROUND

The goal of this work is to improve the effectiveness and scalability of compiler development training for researchers, developers and graduate students. We choose two OpenMP compilers, Clang/LLVM and ROSE as examples. This section gives a brief introduction of background information.

2.1 Compilers

Compilers are essential for HPC. As opposed to interpreted languages, programs written in compiled languages gives a better performance and are more favorable towards high performance computing. A compiler takes high-level human readable programs written in programming languages, such as C/C++ or Fortran, and converts them into low-level binary machine codes for a specific architecture. The entire process of this transformation is complicated. A compiler need to parse the code, check for syntax correctness, gather necessary semantic information (like type checking or variable declaration before use and so forth), then convert the source from high level language to intermediate representation and before transforming them into machine codes [10].

Today a compiler can do much more than converting a program into machine instructions. As HPC hardware designs are evolving, machines are becoming more and more complex, and issues which need to be address by the programmers are also getting convoluted. This raises the question about what more can a compiler do for the programmers. Compilers have very complex designs so that the work of an application developer becomes simpler. Owing to the complexity of design, extending a compiler to add a new feature is a very time consuming job. The development cycle of a compiler is at least 3-5 years. Training programmers to do compiler development is challenging to both the trainer and the trainee.

2.2 Clang/LLVM

LLVM [18] is the prime environment for developing new compilers and language-processing tools. HPC programmers rely on compilers and analysis tools. LLVM is the environment of choice for the development of such tools, and thus should be of interest to many HPC programmers. LLVM makes it easier to not only create new languages, but to enhance the development of existing ones. Its primary C/C++ compiler frontend is Clang. Today most supercomputing clusters deploy LLVM as one of their compilers due to the following reasons:

(1) It provides a high-performance and up-to-date C/C++ compiler frontend Clang.
(2) Many researchers in HPC community enjoy Clang's diagnostic abilities and static-analysis framework.
(3) It allows for tapping other languages that have an LLVM back-end like Intel's ISPC [27] and different scripting languages.
(4) It makes for compelling compiler research, as evident by the plethora of projects built using LLVM [22].

Ever since its first release in 2003, LLVM has gone through a plethora of changes and updates. With every release new features are added and older features are deleted or updated. Owing to these diverse set of features and many more, using Clang/LLVM for developing a tool or a plugin is a very complex task. There are lots of tutorials which are available for Clang/LLVM, but they are all just text based tutorials and come with their own set of challenges.

2.3 ROSE

ROSE is an open source compiler infrastructure developed at Lawerence Livermore National Laboratory (LLNL). It is designed to build source-to-source program transformation and analysis tools for Fortran, C, C++, OpenMP, and UPC applications[28]. Internally, ROSE generates a uniform abstract syntax tree (AST) as its intermediate representation (IR) for input codes. Sophisticated compiler analyses, transformations and optimizations are developed on top of the AST and encapsulated as simple function calls, which can be readily leveraged by tool developers. The ROSE AST can be optionally unparsed to human readable and compilable source files, which in turn can be compiled into final executable by a traditional compiler such as GCC or Intel compiler.

However, for users who are not familiar with the ROSE compiler, it's not easy to customize the framework because of the complexity of ROSE. ROSE has more than two millions lines of code, including tests, built-in projects and tutorial examples. Creating a new transformation module could involve multiple functions, which are located in different files that far away from each other. Like any other compiler frameworks, ROSE compiler exposes its own API functions for developers to traverse, analyze, and modify its abstract syntax tree. Users not only need to learn the general knowledge of compilers but also have to understand how ROSE API functions work.

2.4 OpenMP

In HPC, OpenMP is the de-facto portable programming interface for exploiting node-level parallelism [13]. OpenMP uses C/C++ directives and Fortran comments to annotate base language programs written in C/C++ and Fortran, respectively. These annotations express additional semantics related to parallelism, worksharing, synchronization, tasking, and so on. A compiler supporting OpenMP can recognize OpenMP annotations and transform the annotated input code into multi-threaded code calling some OpenMP runtime functions.

There are multiple compilers implementing OpenMP, such as GCC[4], Intel[5], Cray[12], IBM XL[7], Clang/LLVM and ROSE[20]. Most of the parallel constructs in OpenMP are realized through compiler directives. This allows a serial program to be very easily converted into a parallel one by just adding the necessary preprocessor directives.

Figure. 1 is an OpenMP program to calculate PI in parallel. User inserted an OpenMP `parallel for` directive at line 14-15 right above the loop (Fig. 1a). An OpenMP compiler transforms (or lowers) the program into multi-threaded code with calls to runtime library functions (Fig. 1b). In the lowered code, at line 11-23 the loop block is outlined as a function containing the original statements in the loop. At line 15, a runtime function call is used to split loop iterations among several threads. At line 5 the `main` function passes

```c
#include <omp.h>
#include <stdio.h>

int num_steps = 10000;

int main() {
    double x = 0;
    double sum = 0.0;
    double pi;
    int i;
    double step = 1.0/(double) num_steps;

    // Run the code in parallel
    #pragma omp parallel for private(i,x) \
            reduction(+:sum) schedule(static)
    for (i=0; i<num_steps; i=i+1) {
        x = (i+0.5)*step;
        sum = sum + 4.0/(1.0+x*x);
    }

    pi=step*sum;
    printf("%f\n", pi);
}
```

(a) OpenMP program to calculate PI

```c
... // omitted headers and a data structure declaration storing variable addresses
static void OUT__1__2189__(void *__out_argv);
int main(int argc, char **argv) {
  ... // omitted variable declarations
  XOMP_parallel_start(OUT__1__2189__,&__out_argv1__2189__,1,0,"demo.c",10);
  XOMP_parallel_end("demo.c",15);
  pi = step * sum;
  printf("%f\n",pi);
  XOMP_terminate(status);
}
static void OUT__1__2189__(void *__out_argv) {
  ... // omitted variable declarations
  double *sum = (double *)(((struct OUT__1__2189___data *)__out_argv) -> sum_p);
  double *step = (double *)(((struct OUT__1__2189___data *)__out_argv) -> step_p);
  XOMP_loop_default(0,num_steps - 1,1,&p_lower_,&p_upper_);
  for (p_index_ = p_lower_; p_index_ <= p_upper_; p_index_ = p_index_ + 1) {
    _p_x = (p_index_ + 0.5) * *step;
    _p_sum = _p_sum + 4.0 / (1.0 + _p_x * _p_x);
  }
  XOMP_atomic_start();
   *sum = *sum + _p_sum;
  XOMP_atomic_end(); XOMP_barrier();
}
```

(b) Transformed (or Lowered) code

Figure 1: PI calculation using OpenMP and its corresponding multi-threaded code generated by ROSE

the outlined function's pointer to another runtime function which will spawn multiple threads to execute the outlined function.

The initial OpenMP standard in 1997 only specified a handful of directives. Since then, substantial amount of new constructs have been introduced and most existing APIs have been enhanced in each revision [14]. The latest version of OpenMP 5.0, released in 2018, has more than 60 directives. Compiler support thus requires more efforts than before [19]. A full compiler implementation of the latest OpenMP standard for both C/C++ and Fortran would involve a large amount of development efforts spanning multiple years. Furthermore, more and more researchers and developers are interested in designing various extensions to OpenMP in order to tame the increasing complexity of heterogeneous node designs in high performance computing. Such extensions could be used to enhance the expressiveness, performance or productivity of OpenMP. Support for those extensions requires significant amount of compiler development.

3 CHALLENGES AND SOLUTIONS

Table 1 summarizes the main pain points for compiler training. For example, one of the first problems for developers is getting hands on a machine which is suitable for compiler development. Getting access to a supercomputing cluster could be a challenge and a potential deterrent for many. The second, and the most frustrating challenge for beginners is making sure that all the software packages necessary for developing a compiler are met on the said machine. Sometimes user might not have suitable access to install certain dependencies. Or sometimes the dependencies are just too complex to resolve on a particular machines. One solution to these

Pain Points	Description	Proposed Solution
Accessibility	Paperwork to get accounts on suitable machines	Online sandbox terminal open to anyone
Installation	Many software packages are needed	Docker images
Effectiveness	Traditional text tutorials are not effective	Learning by doing, testing, certification
Content	No single person/group knows all details of OpenMP compiler development	Self-made tutorials + crowd-sourcing to accept external contributions
Design trade-offs	One compiler cannot demonstrate all options	Hosting tutorials for multiple compilers
Costs	Hosting websites with containers costs money	Open-source, self-deployable framework
Security	Online websites have inherent risks	Containers + Cloud machines

Table 1: Pain points and solutions for training OpenMP compiler developers

two problems is to provide a free online sandbox terminal which will already have an environment setup for compiler development.

Based on our experiences, traditional text tutorials are not as effective for compilers development, as hands-on tutorials. If a framework is provided which gives its users an option to learn by hand-on practice, freedom to dig deep and perform self experiments, then such a framework will be most efficacious way of teaching compilers.

Another problem of creating the content of compiler development, especially for OpenMP, is that no one person or group knows all the details of OpenMP implementations since they involve many compilation and runtime stages including parsing, AST, transformation, as well as runtime support. No one implementation demonstrate all the options of OpenMP. This generally results in incomplete or unproductive tutorials. Having an open source environment where multiple users can submit tutorials for multiple compilers can resolve such complications.

Finally hosting tutorials on website costs money. Having containers can result in larger disk space which means more expenses. Having an open sourced, self-deployable framework can help users host their tutorials for free.

FreeCompilerCamp.org is aimed to build a free and open cloud-based training platform integrating the solutions mentioned above. This platform aims to facilitate the training of researchers to quickly develop compilers for OpenMP and help them learn the skills of compiler development. We will elaborate the design and implementation of this platform in the next sections.

4 FREECOMPILERCAMP.ORG PLATFORM

FreeCompilerCamp.org is a learning system with several distinct design principals:

- It aims to allow any developer, who is interested in understanding the internal working of OpenMP compilers, to learn the necessary skills for free.
- It provides a pre-configured compiler development environments in an online sandbox, which eliminates the burden of beginners' tedious and error-prone software installation processes.
- A live training website based on the system is set up, so a web browser and an Internet connection are the only requirements for anyone to get the training.
- The entire training system is open-source, so it can also be deployed by anyone on a private server, workstation or even personal laptop.
- It enables anyone who has the relevant skills to contribute new tutorials as well.

There are two components in the FreeCompilerCamp.org platform (or FreeCC as an abbreviation) as displayed in Figure 2 – a web-based framework with all tutorials and a Play-With-Compiler (PWC) engine for the sandbox environment. The website provides a browser-based interactive interface with two panels: the left panel contains the training instructions in text, and the the right panel connects with the PWC engine, which creates a live terminal sandbox for real-time practice.

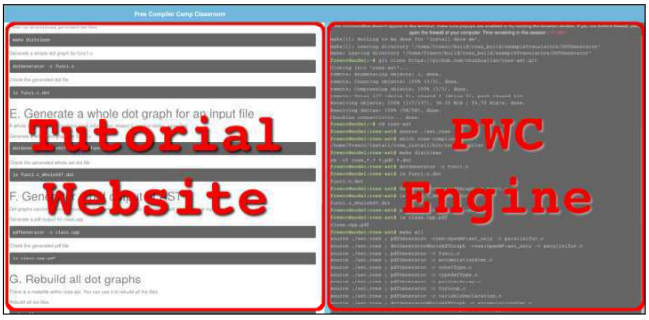

Figure 2: Two components of FreeComplierCamp.org

4.1 Tutorial Website

The tutorial website is created as the major interface of FreeCC. It provides easy-to-understand document in multiple tutorials organized by categories. Users can choose any entry on demands or learn in order.

4.2 Play-With-Compiler Engine

The Play-With-Compiler engine is based on Play-With-Docker (PWD) [26], which is an online sandbox platform for visitors to learn basics about container techniques using Docker [23]. Docker uses OS-level virtualization to deliver software, libraries and configuration files in packages called containers, which are isolated from one another though there are defined channels to enable their communication. Containers on a same machine shares a single operating-system kernel and are thus more lightweight than virtual machines.

Play-With-Docker uses a so-called Docker-in-Docker technique. While the host service is running in an outer docker, the component of this service runs in an isolated inner docker so that multiple components won't affect each other[16, 33]. In the case of PWD, each user has their own sandbox and won't get interrupted by others' activities. PWD uses Apline Linux, which is widely used in docker images due to its lightweight and security.

4.3 Customization

We encountered several technical issues during the development of FreeCompilerCamp.org and subsequently resolved them. Most of these issues may not be new in web development, but our target audience is mostly people with a HPC background, who may not have a flair for web development. Also these issues are common and will be faced by anyone who would like to deploy our framework. Hence mentioning these issues here is vital.

4.3.1 Same-Origin Policy. The same-origin policy [29] restricts resources loaded from one origin to interact with resources from another origin. This prohibits training website and PWC to be deployed on different servers. We had to apply Cross-Origin Resource Sharing [32] mechanism that uses additional HTTP headers to enable resources on PWC server to be accessed by training website.

4.3.2 Port Conflict. Later to simplify management and lower the cost, we decided to deploy both the training website and PWC

on the same server. This caused port conflict since they both use port 80 by default. We set up an HTTP server using Apache and non-default ports redirection to resolve this conflict.

4.3.3 Alpine Linux. The PWD sandbox had dockers built from Alpine Linux, which was unfit for compiler training. Compilers are sensitive to the host system environment. Alpine Linux is not supported for the development of either ROSE or LLVM. Therefore, we created new docker images based on Ubuntu for better compatibility with both ROSE and LLVM. Ubuntu has a much wider application support, hence if future even more compilers can be added in the tutorial.

4.3.4 Security. The PWD sandbox by default gives users root access inside the terminal. This is a security risk since a malicious user may hack into web hosting directories where they are not supposed to access. As a solution we create a user/group (freecc/freecc) in our sandbox and let all process run in that user account instead of root. This way we have more control over what access we want to provide the users.

5 TUTORIAL DESIGN

We have created several initial tutorials to take advantage of FreeCompilerCamp. The goal is to have a good mix of text and commands for users to read and practice essential compiler skills.

5.1 Concepts

Tutorials of FreeCC are designed based on the principle of experimental learning or learning-by-doing. Learning-by-doing was introduced by John Dewey and it promotes the idea that students should learn by actively interacting with environments[15]. Kolb reviewed the major experimental learning models and created his own comprehensive structural model[17]. He also explored the application of experimental learning in higher education. Students not only read static texts but also apply the theoretical knowledge into practical cases. They learn the skills by solving problems, working on small projects, and so on.

Under the guidance of this theory, FreeCC hosts the tutorials to let users start from any point they like with a ready environment, with the following major features:

- We make users practice as much as possible with detailed instructions, by providing an easy-to-use sandbox for users to test given code or conduct their own experiments.
- FreeCC covers different topics in compiler development, including parsing, AST generation, OpenMP programming, compiler extension, and so on.
- We split larger learning tasks into smaller ones to fit each tutorial into a 10-15 minutes session. The goal is to ensure that we can grab sufficient attention from visitors.
- The tutorial not only lists the steps but also explain why each step should be conducted and how it works.
- FreeCC supports clickable code snippets, which can be tested in the sandbox right away by clicking.
- Video instructions are not included currently because more students prefer static tutorials to video tutorials[24]. Using static tutorial is easier to seek and pick different sections of tutorial and learn at a comfortable pace for themselves.

5.2 Example Tutorials

FreeCompilerCamp.org provides a flexible learning experience based on the concepts mentioned above. In particular, we split the training content into several tutorials with incremental complexity so visitors can jump into the right levels they are comfortable with. We start with simple ones to let visitors play with input and output of compilers and get familiar with compilers' internal representations for input programs. After that, we let them try out how to traverse the tree representations and finally how to change the tree for writing transformations.

5.2.1 Tutorial for Learning AST. Taking ROSE as an example, we designed the following tutorials:

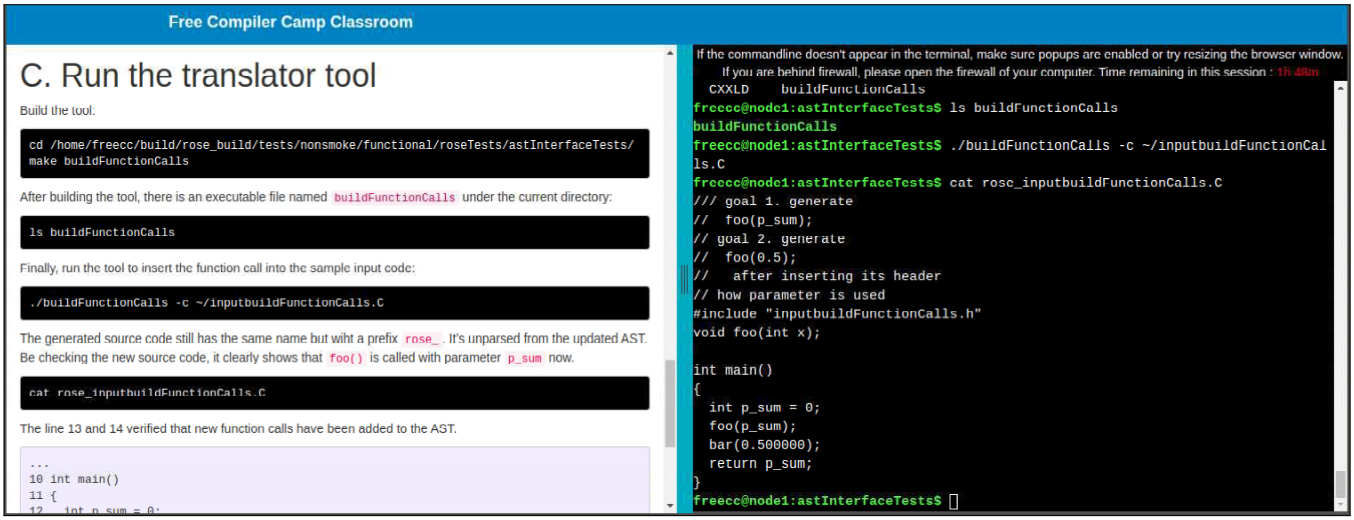

Figure 3: The tutorial for teaching AST modification

- AST/IR Generation. For a given input source file, an AST will be generated and represented visually in a graph. This tutorial shows how information is retrieved from source code and organized internally inside ROSE for future use.
- AST/IR Traversal. After AST generation, this tutorial shows how to traverse the tree to search for certain information of interests, such as loops or functions.
- AST/IR Modification. This tutorial demonstrates the method to add function call nodes into AST. Unparsing the AST will result in an output source file with the inserted function calls.

For example, the AST modification tutorial teaches users how to insert a functional call node into AST and check the updated AST by looking into the corresponding unparsed source code (Fig. 3). User can click the corresponding code snippets to download those files without leaving the page. All necessary source files can be downloaded in the sandbox on demand. In the sample input, there's no function calls in the `main` function. The tutorial explains how a function call subtree is constructed in the compiler and showed all steps to create the subtree and attach it to the AST to complete the task. The input and expected output are both provided in the tutorial so that users can compare their results with the correct solution.

5.2.2 Tutorial of Fixing a Compiler Bug. Developers often learn many things by fixing real bugs. Figure 4 is an example tutorial to fix a user-reported bug in ROSE. A PI calculation program in OpenMP compiled by ROSE generated some wrong value. Upon debugging it was found that during ROSE's transformation of the loop body of `'omp parallel for'`, the loop stride was miscalculated due to incorrect operand nodes were retrieved in the AST. The tutorial first highlights the bug and describes the steps to reproduce it. It then explains how compiler transformation and a runtime library function collaborate to schedule loop iterations among multiple threads. After that, it gives specific instructions on which source files should be modified to fix the bug. At last, with a few simple clicks, the modified ROSE is re-built to compile the test program and correct execution output is generated. Thus in a wholesome way this tutorial gives an example of a real OpenMP implementation bug and explains how to reproduce, debug and resolve it.

5.2.3 Tutorial for Writing a Clang Plugin. We take Clang as another example to show our tutorials. This is a self-contained tutorial about how to write a short plugin in Clang which modify the source code as required.

Let's say that we want to analyze a simple C file as shown in Listing 1. Suppose we want to do some simple fixes on this C file. We would like to change the name of `func1` to `add` and `func2` to `multiply`. Then we would also like to change the function calls of `func1` and `func2` to `add` and `multiply` respectively. This will result in a code as shown in Listing 2. We can write a plugin which will parse through the AST and make the above changes to the file.

Listing 1: Example input code

```
int func1(int x, int y) { return x+y; }
int func2(int x, int y) { return x*y; }
int saxpy(int a, int x, int y) {
    return func1(func2(a,x),y);
}
```

Listing 2: Expected output code

```
int add(int x, int y) { return x+y; }
int multiply(int x, int y) { return x*y; }
int saxpy(int a, int x, int y) {
    return add(multiply(a,x),y);
}
```

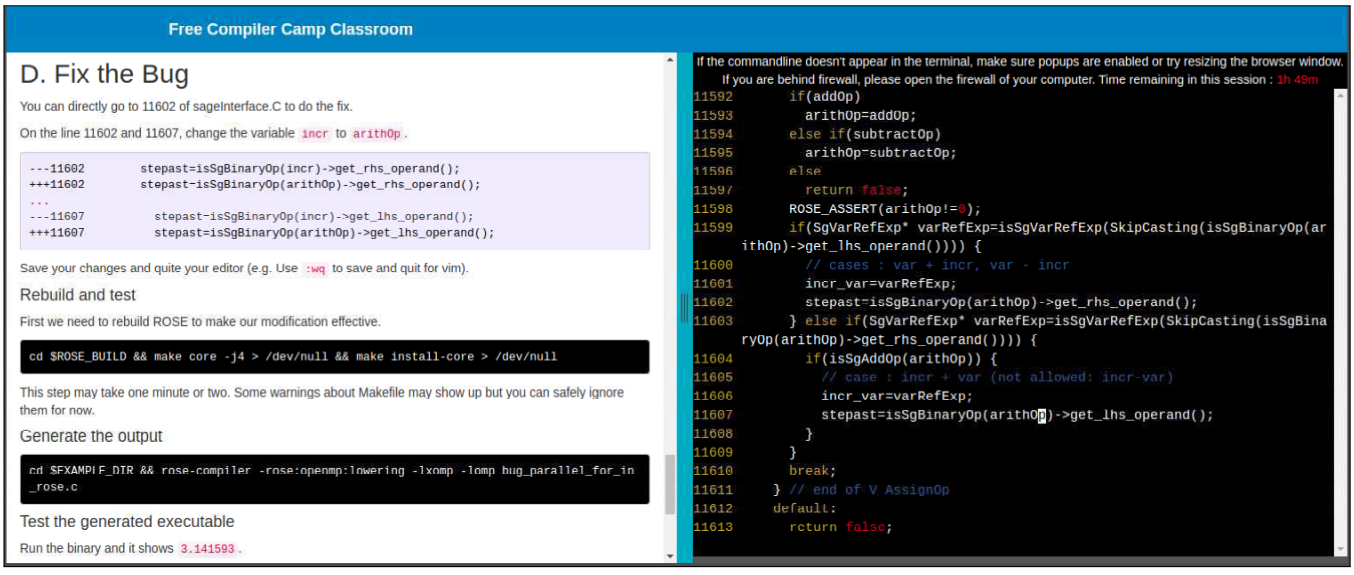

Figure 4: The tutorial for fixing an OpenMP translation bug in ROSE

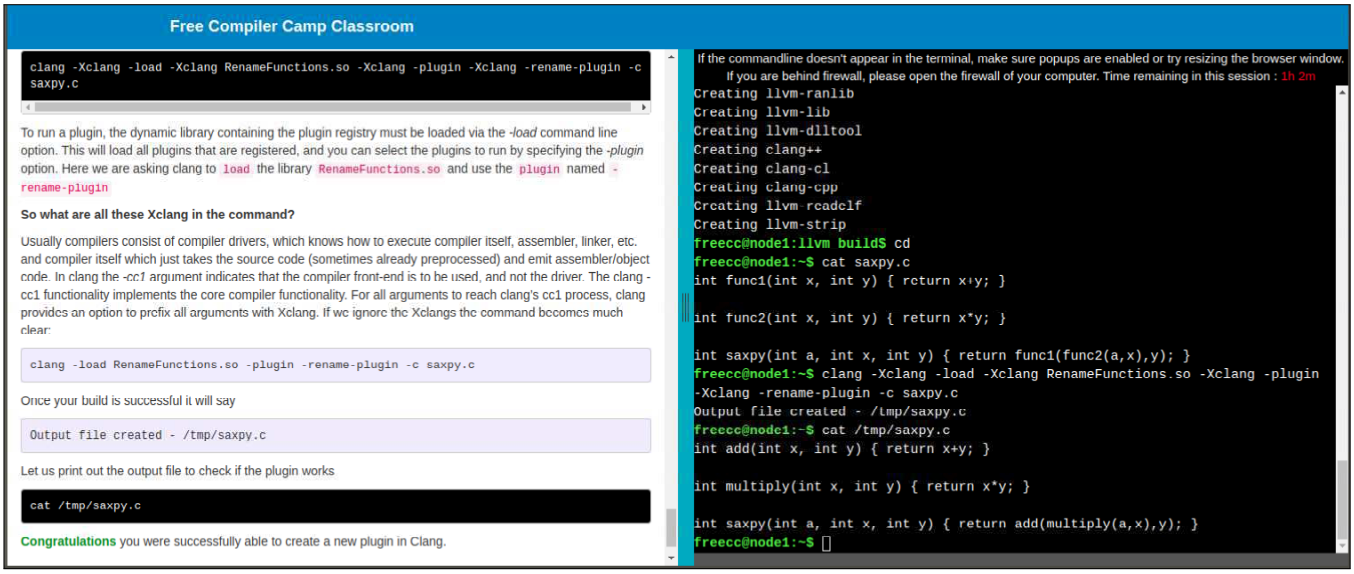

Figure 5: The tutorial for writing a Clang Plugin

This tutorial explains in details the steps that need to be taken to write this plugin. It starts with giving an overview of what is a clang plugin. Then it goes to explain what this plugin intends to do. Then it explains how to setup the source code structure of the plugin and which files need to be written or modified in order to write this plugin. The tutorial also provides an option to the user to download a reference plugin or to write the plugin by themselves. In the end it helps the user to build and test out the plugin. Figure 5 is a screenshot of this tutorial where the user tests the plugin.

5.3 Trial and Feedback

We have asked a group of students who major in Computer Science but with only basic compiler knowledge to take a trial of FreeCC. To assure the most accurate feedback, no pre-training ahead of the trial was provided. Students picked one tutorial based on their interests and completed it all by themselves without any other guidance. Then they filled a survey form about their experiences of using FreeCC. The feedback from the survey is summarized as follows:

- They feel comfortable with the length of each tutorial with 10-15 minutes.
- All steps of tutorial are completed without any issue.
- Students prefer to use clickable code snippets rather than type they manually.
- Providing a choice from multiple code editors will be helpful.
- Additional video instructions are not needed.
- The sandbox and clickable code snippets attracted the most attention. They make FreeCC unique comparing to conventional tutorials.
- Some students tried to conduct their own experiments in PWC as we expected.
- The overall appearance of FreeCC could be improved.
- They want to retrieve files from the sandbox (ssh or git might help).
- Support for X11 forwarding might be needed to display graphics.
- The tutorials can use some links to external courses for fundamentals about OpenMP and compilers.
- GPU support is needed for extending tutorials running on GPU.

Based on the feedback, we conclude that the current design of FreeCC tutorial is a very good start point. All testers are satisfied with the features of FreeCC. The sandbox, PWC, is highly rated since students don't need to configure any complicated environment but a modern browser on any system. Criticism mostly came from the website appearance, customization and cloud-machine resources for GPUs, which can be addressed in the future.

6 RELATED WORK

Existing Compiler Tutorials. Both ROSE [9] and Clang [1] already have abundant documentation on their official websites, including user guides, tutorials, and Doxygen generated API webpages, etc. There is also a ROSE wikibook which is open for anyone to contribute. Clang's official page provides documentation ranging from how to obtain and build clang, to how to write plugins and create tools, etc. Along with that there are several free and open source tutorial blogs which are available for Clang. OpenMP's official page provides links [8] to several open tutorials available on the internet. However, all the existing documentation is written in the traditional text format. It is still up to the readers to find a machine to install and configuration the development environment. The entire preparation phase may take hours to finish. Many learners simply give up due to the tedious steps or the lack of access to a suitable machine.

Online Education systems. There is a large amount of online learning systems [21], including Khan Academy [31], Coursera [2], edX [3] and so on. These learning systems mostly are aimed for

general education and training purposes. They are not specially targeting compiler development. A closely related website is freeCodeCamp [6], which is an online training platform for training web developers. Play-with-Docker is an online sandbox for people to learn docker. Our work builds on top of this framework with customization for compiler training.

Although several cloud-based tools have been leveraged for computer science education, there is a clear lack of such tools to teach compiler development. Ngo et. al [25] use CloudLab, a national experimentation platform for advanced computing research, to teach cluster computing to students. Bisbal [11] provides an outline of what topics need to be taught to computational scientists in a logical order to train them in open-source software. Shin et. al. [30] developed a web-based MOOC system related to computational science education which could hold various resources and efficient programming practices. Many such tools and resources are available across several domains of computation, but compiler development is still devoid of such online tools.

7 CONCLUSION AND FUTURE WORK

In this paper, we have introduced an ongoing effort, FreeCompilerCamp.org, a free and open online learning platform aimed to train researchers to quickly develop OpenMP compilers. FreeCompilerCamp.org is built on the Play-with-Docker platform to relieve learners' burden of finding suitable machines and installing software. The tutorials of FreeCompilerCamp are entirely web-based with both text content and a live embedded sandbox terminal in which learners can immediately practice compiler development skills. Instructors or students can customize this platform easily and deploy it on any local server, workstation or even personal laptop.

In the future, we will include more tutorials about how to develop OpenMP compilers for HPC. We will also design online examinations to help learners evaluate the effectiveness of their learning process. We welcome anyone to try out our system, give us feedback, contribute new training courses, or enhance the training platform to make it an effective learning resource for the HPC community.

ACKNOWLEDGEMENT

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, and partially supported by the U.S. Dept. of Energy, Office of Science, ASCR SC-21), under contract DE-AC02-06CH11357. IM Release Number: LLNL-CONF-791339. This material is also based upon work supported by the National Science Foundation under Grant No. 1833332 and 1652732. This research was also supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

REFERENCES

[1] 2019. Clang Documentation. Retrieved Sep 26, 2019 from https://clang.llvm.org/docs/
[2] 2019. Coursera. Retrieved Sep 26, 2019 from https://www.coursera.org/
[3] 2019. edX. Retrieved Sep 26, 2019 from https://www.edx.org/
[4] 2019. GCC Support for the OpenMP language. Retrieved Jul 22, 2019 from https://gcc.gnu.org/wiki/openmp
[5] 2019. Intel C++ Compiler Code Samples. Retrieved Jul 22, 2019 from https://software.intel.com/en-us/code-samples/intel-c-compiler
[6] 2019. Learn to code with free online courses, programming projects, and interview preparation for developer jobs. Retrieved Sep 26, 2019 from https://www.freecodecamp.org/
[7] 2019. OpenMP support in IBM XL compilers. Retrieved Jul 22, 2019 from https://www.ibm.com/developerworks/library/l-openmp-support/index.html
[8] 2019. OpenMP Tutorials & Articles. Retrieved Sep 26, 2019 from https://www.openmp.org/resources/tutorials-articles/
[9] 2019. Rose Documentation. Retrieved Sep 26, 2019 from http://rosecompiler.org/ROSE_HTML_Reference/index.html
[10] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. 1986. Compilers, principles, techniques. *Addison wesley* 7, 8 (1986), 9.
[11] Prentice Bisbal. 2019. Training Computational Scientists to Build and Package Open-Source Software. *Journal of Computational Science Education* 10, 1 (Jan. 2019), 74–80. https://doi.org/10.22369/issn.2153-4136/10/1/12
[12] C Cray. 2019. C++ Reference Manual, S-2179 (8.7). Cray Research. Retrieved Jul 22, 2019 from https://pubs.cray.com/content/S-2179/8.7/cray-c-and-c++-reference-manual/openmp-overview
[13] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: An industry-standard API for shared-memory programming. *Computing in Science & Engineering* 1 (1998), 46–55.
[14] Bronis R. de Supinski, Thomas R. W. Scogland, Alejandro Duran, Michael Klemm, Sergi Mateo Bellido, Stephen L. Olivier, Christian Terboven, and Timothy G. Mattson. 2018. The Ongoing Evolution of OpenMP. *Proc. IEEE* 106, 11 (2018), 2004–2019.
[15] John Dewey. 1938. *Experience and Education.* Kappa Delta Pi.
[16] Tom Goethals, Dwight Kerkhove, Laurens Van Hoye, Merlijn Sebrechts, Filip De Turck, and Bruno Volckaert. 2019. FUSE : a microservice approach to cross-domain federation using docker containers. In *Proceedings of the 9th International Conference on Cloud Computing and Services Science*. Scitepress, 90–99. http://dx.doi.org/10.5220/0007706000900099
[17] David A. Kolb. 2014. *Experiential Learning: Experience as the source of learning and development.* Pearson FT Press.
[18] Chris Lattner and Vikram Adve. 2004. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*. IEEE Computer Society, 75.
[19] Ilias Leontiadis and George Tzoumas. 2001. OpenMP C Parser.
[20] Chunhua Liao, Daniel J Quinlan, Thomas Panas, and Bronis R De Supinski. 2010. A ROSE-based OpenMP 3.0 research compiler supporting multiple runtime libraries. In *International Workshop on OpenMP*. Springer, 15–28.
[21] Tharindu Rekha Liyanagunawardena, Andrew Alexandar Adams, and Shirley Ann Williams. 2013. MOOCs: A systematic study of the published literature 2008-2012. *The International Review of Research in Open and Distributed Learning* 14, 3 (2013), 202–227.
[22] LLVM. 2019. Projects Built with LLVM. Retrieved Aug 31, 2019 from https://llvm.org/ProjectsWithLLVM/
[23] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* 2014, 239 (2014), 2.
[24] Lori S. Mestre. 2012. Student preference for tutorial design: a usability study. *Reference Services Review* 40, 2 (2012), 258–276.
[25] Linh B. Ngo and Jeff Denton. 2019. Using CloudLab as a Scalable Platform for Teaching Cluster Computing. *Journal of Computational Science Education* 10, 1 (Jan. 2019), 100–106. https://doi.org/10.22369/issn.2153-4136/10/1/17
[26] Marcos Nils and Jonathan Leibiusky. 2019. Play with Docker. Retrieved Jun 18, 2019 from https://training.play-with-docker.com
[27] Matt Pharr and William R Mark. 2012. ispc: A SPMD compiler for high-performance CPU programming. In *2012 Innovative Parallel Computing (InPar)*. IEEE, 1–13.
[28] Dan Quinlan and Chunhua Liao. 2011. The ROSE source-to-source compiler infrastructure. In *Cetus users and compiler infrastructure workshop, in conjunction with PACT*, Vol. 2011. Citeseer, 1.
[29] Jörg Schwenk, Marcus Niemietz, and Christian Mainka. 2017. Same-origin policy: Evaluation in modern browsers. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 713–727.
[30] Junghun Shin, Jason Cholhoon Jang, Huiseung Chae, Gimyeong Rvu, Jaejun Yu, and Jongsuk Ruth Lee. 2018. A Web-Based MOOC Authoring and Learning System for Computational Science Education. In *2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. IEEE, 1028–1032.
[31] Clive Thompson. 2011. How Khan Academy is changing the rules of education. *Wired Magazine* 126 (2011), 1–5.
[32] Anne Van Kesteren and et al. 2014. Cross-origin resource sharing. *W3C REC-cors-20140116, latest version available at< https://www.w3.org/TR/cors/* (2014).
[33] Chanho Yong, Ga-Won Lee, and Huh Eui-Nam. 2018. Proposal of container-based HPC structures and performance analysis. *Journal of Information Processing Systems* 14, 6 (2018), 1398–1404.