A Proposal for Google Summer of Code 2014

Clang Plugin providing automatic application instrumentation

Info.
Shitao Weng
Sun Yat-sen University
Major: Computer Science and Technology
Timezone: Beijing +0800 UTC

Contact
Tel. +86-13480201943
E-Mail: wengsht.sysu@gmail.com
Github: github.com/wengsht
Blog: wengsht.github.io
IRC Nick: wengsht

GSoc Information

It's my first year participating in GSoC. And this proposal is the only one I submit to GSoC this year.
I plan to treat GSoC as a full time job this summer, with 40 to 60 hours working for this project per week if this proposal accepted. Before the result confirming, 20+ hours per week will be paid to perpare for the coding.

Project Motivation / Introduction

The function tracer in LTTng base on -finstrument-functions parameter is useful but sometime overhead for simple purpose.
As the LTTng ideas page describes it "A more refined automatic instrumentation solution is necessary to uphold LTTngs objective of providing a low-intrusiveness tracing solution".
Clang plugin is such a solution that we can travel(in clang, RecursiveASTVisitor is useful for this project) the parsed AST tree for the filter work, which will cut down the load of the instrumention and uphold LTTng's objective.

Project Goals

During this GSoC summer, I will deliver:
1. A source-to-soucre transformation clang plugin for C/C++ automatic instrumentation in build time which is:
1) Easy use, easy configuration instrumentation rules.
2) Basically can be use to trace the enter and exit of the function call using LLTng.
3) Insert LTTng tracepoint follow the instrumentation rules.

After this gold achieved, It should be easy to insert LTTng tracepoint into base on the configuration:
1) Funtcion enter and exit.
2) Control flow matter Stmt(for, while, if, etc.).

This will be the **major part** of this project.
You may wonder why I chose source-to-source, but not source-to-LLVMIR. It is because of:
1) The support for this transformation within a clang plugin is scarce supported(Or I have not learn enough to know how to do that?). It seems normal for the small support. clang is just a FrontEnd which end after LLVM IR generated, code optimizing for LLVM IR can be done using LLVM pass, but not a clang plugin. The only thing I know how to change the LLVM IR(insert

tracepoint in particular position) within a clang plugin is changing the AST, that too danger and unnecessary for the gold of this project.
2) Source-to-source is enough for automatic instrumentation. It is easy to link with LTTng.
3) Inserting tracepoint in particular position is easy for me, and enough for automatic gold.

2. A clang plugin for C/C++ automatic function instrumentation in run time which is:
1) work with -finstrument-functions and liblttng-ust-cyg-profile-fast.so.
2) Easy use, easy configuration instrumentation rules.

This will be easy to be achieved.

3. Analysis scripts using the Babeltrace Python API to detect resource leaks, dead code, hot paths and so on will be developed if I finish the plugin early. This gold depends on time. It should be hard to judge the running stage accurately using the tracing report.

Implementation I divide the implementation of this project for 5 parts:
1. Design and coding of instrumentation rules, including whitelist and blacklist. This job will design a easy usage/configuration rules for tracing, which depends on:
1) Regular Expression matching function declarations name.
2) Location of the declaration or statement(call from particular include file, stmt locate in a particular file, for instance)
3) Type of the function(Cxx member function? a constructor? a c-style call?)
4) Control flow statement trace configuration.(trace for loop or not, etc.)
More tracing rules will be added in futurn work. These rules will be designed carefully and implemented easily with the support of regex library in STL and the AST API support in clang. For example, I can get the NameInfo from a FunctionDecl pointer while traveling the AST using RecursiveASTVisitor, and then use the filter rule to decide the tracing action.
The class SourceManage in clang will solve problem about file location. It is easy to know where a decl or stmt come from using this class.

I don't think it is possible to know which library a CallExpr call to before link time. Clang plugin is a Frontend plugin during compile time. Anyway, I will try my best for a rule about library.

2. A simple Clang plugin for C/C++ automatic function instrumentation in run time.
It seems that it is a easy work with support of -finstrument-functinos and liblttng-ust-cyg-profile.so.What I need to do is build a plugin to travel all the FunctionDecl on the AST, add Attribute (no_instrument_function) to the FunctionDecl and let the Codegen finish compile work with the '-add-plugin' instead of '-plugin' to load the plugin.

The most difficulty this part is that clang may stop compile flow when plugin work. But it can be solved because of the small change on the AST with little efforts.
Using this plugin, user will compile there project with plugin loading parameters and offer a configuration for tracing filter.
It is likely a simple plugin for me to check the clang AST work harmoniously with the LTTng.

3. A library base on clang API for tracepoint inserting.
Inserting tracepoint into particular position will use the API mainly about SourceLocation, clang/Rewriter. This library will resolve inserting tracepoint of different situations, tracepoint for function enter/exit and for loop must have different tracing type.
I treat this library link the clang AST with LLTng tracepoints.

4. A source-to-source Clang plugin for C/C++ automatic tracepoint insert in build time.
It is the major work of this project. Having the base I have built, it is simpler a lot. All I have to do is to use the configuration rules library to travel the clang AST with the help of Recursiveastvisitor. Handling every type of declarations and statements, then insert a tracepoint here using the library built up in 3.

This part is not hard to finish because I have built a source-to-source clang tool for c/c++(can be found on my Github).But It should have many code to be coded for different tracing need.

5. Analysis scripts using the Babeltrace Python API.
It depends time. Still have not idea for this gold. But I will try.

More detail about the model design and implementation will be updated in the project page on my Github. It is still blank when this proposal being submited.

Timeline

I will treat GSoC as full time once acceptance confirm, I'm highly interested on working efficiently on this project. I had try my best to make the timeline plan base on my understanding about clang and LTTng. However, the only constant is change. I will constantly rejust the timeline according to the actual conditions to achieve the project goal faster and better. The course in my school is low stress for me, 40 to 60 hours per week will be ensured for this project after acceptance confirm.
I have some experience developing based on clang Front End API.It will take a few time for me to know how code insert happen in clang AST.That why I plan to do some analysis job.However, the analysis scripts will not be completed fully if the work before it get into trouble and consume more time.
Report will be wrote per week in my Blog.

Before Acceptance Confirm

21 - 30 March
Read document to learn how to work for a open source project.

31 March - 6 April
Continuous learn how to work for a open source project. Bug and patch report try.

7 - 13 April
Read documents and codes of LTTng to learn how tracepoint works and what type of tracepoint should be insert for all kind of instrumentations.

14 - 20 April
Continuous documentation on LTTng. Get a prototype/design for specific tracepoints for all the positions to insert.

after Accepted **21 - 27 April**
Review the Rewriter in clang API on the base of LTTng research. Check the feasibility for all the tracepoint designed in the prototype.

28 April - 4 May
Design of the trace rule of function calls. Build a function filter based on the trace rule design. Which will use regex library in STL and many other API in clang AST.

5 - 11 May
Work on a simple plugin using the function filter to add attribute(no_instrument_function). Which will trace on the way what -finstrument-functions and LTTngs cyg-profile library do. But with the filter support, the load will be cut down.

12 - 18 May
Test the plugin if it is working well.Adjust the function instrumentation rules for better/easier usage. Enlarge the instrumentation rules, not just for function call, but also for other control flow statements(if, while, for, etc.).Design a easily configure rules for the control flow trace.

19 - 25 May
Build the basis of the tracepoint inserting library.Which will bind the clang AST and LTTng together for trace gold.

26 May - 1 June
Build the basis of the source-to-source clang plugin. Basis AST transformation will be done for the inserting. Build a basis of a library for common AST API(It will be better to be a library, sometimes it is not good to call the API directly).

2 - 8 June
Work on the basis library.

9 - 15 June
Work on the function tracepoint. Automaticly insert the tracepoint in the

source file base on the instrumentation rules.

16 - 22 June
Test for the work finished. Perpare for the mid-term evaluation.

Mid-term

23 - 29 June
Work on other control flow tracepoints(loop, cond, and so on).Automaticly insert the tracepoint in the source file base on the instrumentation rules.

30 June - 6 July
Continuous coding and tesing for all the library and plugin main.

7 - 13 July
Continuous coding and testing. Read Babeltrace Python API to learn how to develop a analysis scripts for LTTng report.Learn how to find resource leaks, dead code, hot paths and so on base on the control flow trace data.

14 - 20 July
Continuous testing, Documentation. Continuous data analysis.

21 - 27 July
Continuous testing, Documentation. Continuous data analysis.

28 July - 3 August
Continuous testing, Documentation. Continuous data analysis.

4 - 10 August
Continuous testing, Documentation. Continuous data analysis.
Perpare for final evalution.

11 - 17 August
Perpare for final evalution.

More info. about project

1. Have you been in contact with us previously about your project ideas on the mailing list or on IRC?
Yes, I have some talk via email with Christian Babeux.

2. Would you like to be involved in that maintenance after the project finishes?
Of course.

And do I want to continue work for LTTng?
It depends, I'm looking for a open source project to work for long time, LTTng seems to be the one :).

3. What help/guidance will you need from the LTTng team?
I will do the design and find solutions for difficulty toward the project goal by myself. I hope the my metor(or LTTng team) could warn me if he has better designs or a simpler solutions when I ask for help. That must be helpful!

4. Which portions of the project idea do you see as the most challenging?
1). Design of Easy use tracing rule.
2). Trace a function callexpr from a particular libary. In fine, it is impossible(or just for me?) for clang plugin which work during compile time! Clang plugin can not insert something when linking. So this goal should have some other design but not simple base on the name of the library.
3). Data analysis.

5. Why are you the right person for this project?
1). I have develop a clang tool to for c/c++ source-to-source transformation using clang front end API, it's easy for me to work with clang AST. I'm enjoying it.
2). See more info. about me below :) .

More info. about me

I am a graduate student at Sun Yat-sen University in China. I have a good knowledge of C/C++, and a coding experience of about 4 years. Even though I have not experience in a big open source project, I have worked on a variety personal projects from low level to higher. I have just finish a clang tool based on the frontEnd API of clang, which is a source-to-source transformation. You can see for more detail in my Github.
During 4 years coding life, I have participated the ACM-ICPC, and get a sixth place(Without counting Staring team, Gold Medal) in one of 2013 Asia Regional Contests(2013 ACM-ICPC China Nanjing Invitational Programming Contest), You can click the Board for more detail about this contest, name of our team is SYSU_ChenStormstout. Yeap, it's the name of the panda brewmaster in War3, too:). Although I have not got a ticket for the Final Contest this year, I have a good knowledge in Algorithm and Data Strucure.
I am looking for a open source project to work for long time, not just for GSoC 2014, so far, LTTng seems to be a good choice.