"LLVM Clang Utilities"

Borase Gayatri S., Gawale Mohini R., Mate Satyabhama S., Mujawar Nilofar S.

***Abstract*—**Engineering is not only a theoretical study but it is a food for thought to implement something new and making new things which grabs the bull by its horn and get the job done through practical study. It is an art which can be gained with systematic study, observation and practice. In the college curriculum we are only stinking with the theoretical knowledge of industries and a little bit of implementation knowledge that how it is work? But how can we prove our practical knowledge or efficiency of the industry.

Software engineers who are always running the gauntlet of questions and doubts need to understand programs in order to effectively maintain them. The *callgraph*, which presents the calling relationships between functions, is a useful representation of a program that can aid understanding. It also avoids the wastage of time in spinning the wheels in interpreting the code line by line manually. We are presenting a tool that will generate Callgraph from C/C++ code for *LLVM compiler*. This tool will be helpful for developers, testers to understand and maintain the code.

This software system is generating Callgraph and UML from C/C++ code. This system is using existing code from LLVM clang project's *C/C++ parser*. It is analyzing and interpreting the abstract syntax tree generated by clang parser and output callgraph data. Similarly it is using the Abstract Syntax Tree to derive class definition data. This system is helping the developers to easily understand the flow of program for giant codes. Callgraphs are very useful in understanding how program works.

***Index Words*—**Abstract Syntax Tree, Callgraph, CDD, Clang, Compiler, JIT, LLVM.

I. INTRODUCTION

In this paper, we are presenting a tool which can be helpful in software engineering process. It provides food for thought to the user to develop software effectively and reasonable description. We are introducing a callgraph generating software with LLVM-Clang compiler which will represent the calling relationships between the member function of the program. In software engineering, during SDLC process developer needs such tool to keep an eye out to easily get such calling relationship and also the details of each and every class of a program to efficiently modularize the program.

So, through our project we are addressing these all utilities in a single application. CDD (Class Definition Details) gives details of every class like member functions, attributes etc. LLVM as a backend compiler and Clang as a frontend compiler speed up this process. It takes very less time as compared to existing software. Today's GCC based callgraph generators restrict the usage of application because of GCC's drawbacks or limitations. LLVM-Clang overcomes these limitations through this project.

II. PROBLEM STATEMENT

Understand and use existing code from LLVM clang projects C/C++ parser. Write analyzer to interpret the abstract syntax tree generated by clang parser and to output call graph data. Similarly, use the AST to derive class definition data.

III. EXISTING SYSTEM

The existing callgraph generator software's such as Egypt, KcacheGrind, GraphViz etc. are based on GCC compiler which is very slow and time consuming. Currently there is no software for callgraph generation based on Clang LLVM compiler.

A) Disadvantages of Existing System

1) The existing callgraph generator softwares aretimeconsuming; it requires more time both in serial executionmode, memory disk operations and in parallel mode.

2)These systems do not differentiate between the functionswith the same name and argument list.

3) The existing callgraph generator software's requiresmore memory space during bootstrapping processwhere the most significant part of the compilationprocess is probably the compilation of compilers.

4) Existing systems do not specify any restrict level for callgraph generation.

IV.PROPOSED SYSTEM

Proposed system is an intuitive solution over all above mentioned drawbacks. We are developing a system based on Clang-LLVM compiler, which keeps an eye out to overcome the problems like time consumption, more memory utilization etc. Users who are using GCC based callgraph generators running the gauntlets

with so many questions and queries like why to use these systems with so many bottlenecks.

In this system, Clang frontend's preprocessor and AST building is significantly faster. The internal code representation of GCC goes through several conversions that are not necessary in LLVM backend, because code is always kept in the LLVM IR there.

A) Advantages of Proposed System

1) Clang frontend's preprocessor and LLVM IR form makes the system's execution very fast.

2) Callgraph can detect polymorphism calls. It also detects classes and methods which are never used so that we can keep them at arm's length from program.

3) We can give multiple C/C++ files as inputs which will describe the calling relationship between those separate programs.

4) Callgraph clarifies program control which increases human comprehension of the program.

V. SYSTEM ARCHITECTURE

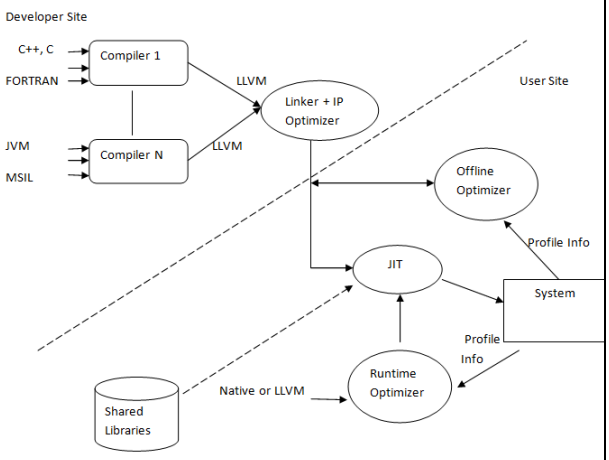

Fig.1 System Architecture

In this architecture, LLVM is used for compile-time, link-time, run-time and idle-time optimization of program. Linker is used for linking the files. Compiler generates the bytecode, that bytecode is executed by JIT (Just-In-Time).

VI.IMPLEMENTATION DETAILS

We are implementing our project in C++ language and also using the Clang-LLVM classes.

- First of all, system will compile the C/C++ program for which we are generating the callgraph.
- Then Clang will generate the AST for given C/C++ program.
- System will access the generated AST using LLVM classes which differentiate keywords, identifiers, functions etc. in the program.
- Finally it will generate the callgraph and CDD for given program from that AST.

VII. CASE STUDIES

LLVM/Clang Compilation: The border between the Clang frontend and the LLVM backend runs right through the code generation stage. The first step of that stage is to transform the AST into the LLVM IR. Everything that follows is performed by LLVM components. However the entire compile pass is controlled by Clang

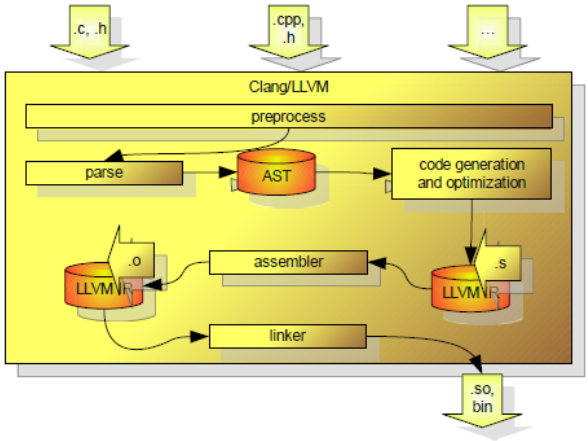

Fig.2 LLVM/Clang Compilation process

The border between the Clang frontend and the LLVM backend runs right through the code generation stage. The first step of that stage is to transform the AST into the LLVM IR. Everything that follows is performed by LLVM components. However the entire compile pass is controlled by Clang.

The most notable difference to the GCC backend is that the LLVM IR encompasses the entire compile run time. Because the LLVM IR is available at link time, the linker can perform additional optimizations across object files. This is most likely due to Clang's development goal to function as a drop in replacement for the GCC.

LLVM would be capable to attach the LLVM IR to the final binaries, allowing on the fly relinking when depending libraries are updated or to apply new optimizations passes, e.g. when the compiler is updated or when new profiling data is available, that suggests that specialized versions of certain functions would benefit runtime. All components to realize this are already in place, all it would need was another Clang driver without the GCC legacy architecture and an operating environment that tightly integrates LLVM.

*CallGraph:*Callgraph is a directed graph that represents the calling relationships between the program's procedures.

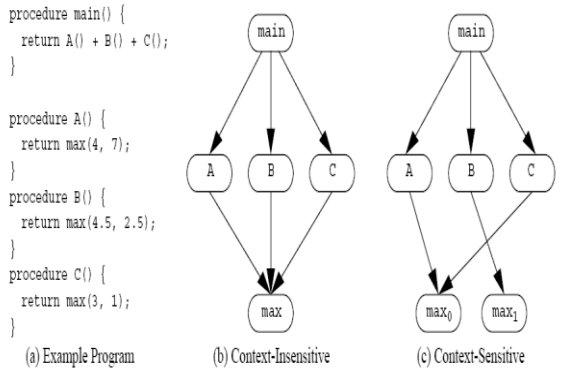

Fig.3 Callgraph

Call graphs are a basic program analysis result that can be used for human understanding of programs, or as a basis for further analyses, such as an analysis that tracks the flow of values between procedures.One simpleapplication of call graphs is finding procedures that are never called.

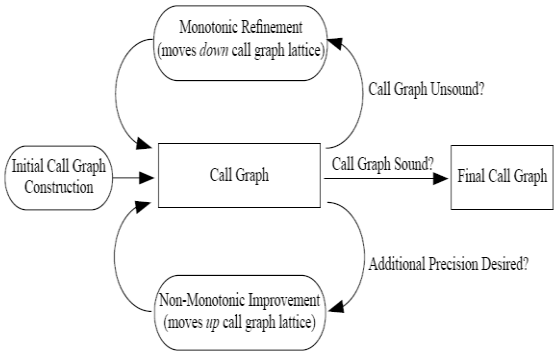

Fig.4 : Generalized Call Graph Construction Algorithm

Fig.4 shows the generalized callgraph generation algorithm.

VIII. SYSTEM FEATURES

1) FEATURE:
Compilation of C/C++ code. The most important thing the compiler must be able to do is compile C/C++ code that interpreter could do natively, without the help of any outside libraries or any complicated language constructs.

a) Stimulus/Response:
This is the normal path of code generation of the compiler.

b) Functional Requirement:
The compiler must be able to understand native C/C++ code completely. The compiler must be capable of detecting syntax errors.

2) FEATURE:
Generation of Bytecode. The LLVM bytecode representation is used to store the intermediate representation on disk in compacted form.

a) Stimulus/Response:
LLVM bytecode file format is actually two things: a bitstream container format and an encoding of LLVM IR into the container format. The bitstream format is an abstract encoding of structured data, very similar to XML in some ways. Like XML, bitstream files contain tags, and nested structures, and you can parse the file without having to understand the tags.

b) Functional Requirement:
To store the intermediate representation.

3) FEATURE:
Generation of Callgraph and CDD.

a) Stimulus/Response:
System will extract data from AST to generate the Call Graph. Generated call graph will help the user to understand the flow of program. Class Definition Data (CDD) helps the user to understand the program modules. Input to system is C or C++ file.

b) Functional Requirement:
To generate the callgraph and CDD. It will generate the callgraph by extracting the AST (Abstract Syntax Tree) which is an intermediate representation of program.

IX. FUTURE ENHANCEMENT

1) Generate callgraph in tree structure format to understand the program more efficiently and effectively.

2) Generate callgraph for languages other than C and C++.

3) Adding more utilities in same application like different symbols in callgraph to so various notifications.

Our Contributions:

It takes lot of time in testing or debugging any large program manually. So, in our proposed system we are generating callgraph which is calling relationship between the member functions of the programs. And through this callgraph the program can be easily handled by users. In software industries, this system can be used by software developers, analyst, testers or anyone who is directly dealing with software. The combination of Clang and LLVM provides a toolchain with library-based design like rest of the

LLVM, Clang is easy to embed into other application and Callgraph using Clang-LLVM gives effective flow of C/C++ program and also speedup the process of callgraph generation. Our project is beneficial in any stage of the SDLC.

X. CONCLUSION

The combination of Clang and LLVM provides a tool chain with library-based design like rest of the LLVM, Clang is easy to embed into other application and Callgraph using Clang-LLVM gives an effective flow of C/C++ program.

LLVM as a backend compiler and Clang as a frontend compiler speed up the process of callgraph generation. It takes very less time as compared to existing software.

XI. REFERENCES

[1] "Evaluating Value-Graph Translation Validation for LLVM" Jean-Baptiste Tristan, Paul Govereau, Greg Morrisett *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, June 2011.

[2] "LLVM and Clang: Advancing Compiler Technology" Chris Lattner *Keynote Talk, FOSDEM 2011: Free and Open Source Developers European Meeting*, Brussels, Belgium, Feb. 2011.

[3] "IntPatch: Automatically Fix Integer-Overflow-to-Buffer-Overflow Vulnerability at Compile -Time" Chao Zhang, Tielei Wang, Tao Wei, Yu Chen, and WeiZou *Proc. of the 15th European Symposium on Research in Computer Security (ESORICS 2010)*, Athen, Greece, Sep. 2010.

[4] "ClangBSD" Roman Divacky *BSDcan 2010*, May 2010.

[5] "Introduction to the LLVM Compiler System" ChrisLattner*Plenary Talk, ACAT 2008: Advanced Computing and Analysis Techniques in Physics Research*, Erice, Sicily, Italy, Nov. 2008.

[6] "Clang/LLVMMaturity Evaluation Report" Dominic Fandrey, June 2010.

[7] "Application-only Call Graph Construction" Karim Ali Ondrej Lhotak David R. Cheriton School of Computer Science, University of Waterloo.

Authors

Borase Gayatri S.
B.E.Computer
University of Pune
Department of Computer Engg.
Govt. College of Engg, & Research,
Awasari (kd), Tal- Ambegaon, Dis-Pune.

Gawale Mohini R.
B.E.Computer
University of Pune
Department of Computer Engg.
Govt. College of Engg, & Research,
Awasari (kd), Tal- Ambegaon, Dis-Pune.

Mate Satyabhama S.
B.E.Computer
University of Pune
Department of Computer Engg.
Govt. College of Engg, & Research,
Awasari (kd), Tal- Ambegaon, Dis-Pune.

Mujawar Nilofar S.
B.E.Computer
University of Pune
Department of Computer Engg.
Govt. College of Engg, & Research,
Awasari (kd), Tal- Ambegaon, Dis-Pune.

