Clang+LLVM: The amazing compiler infrastructure.

Aravind Machiry

Machiry Aravind Kumar

30 March 2017 ·

you know life gave up on you when gcc segfaults.

internal compiler error: Segmentation fault

Please submit a full bug report,
with preprocessed source if appropriate.
See <http://source.android.com/source/report-bugs.html> for instructions.

Why is it amazing!?

- Very active and helpful community with extensive documentation.

- Try to do instrument something with `gcc` and let me know how it goes.

- <= 12 yrs (only!!) vs 32 yrs (gcc)

WEBASSEMBLY

How is it organized?

All components are extensible by design.

Clang: The FrontEnd

- Lexer, Parser and Semantic Analysis.
- Sensible errors.
- Extensible (libclang): Source-to-Source transformations, light-weight static analyzers.

opt (LLVM): The optimizations.

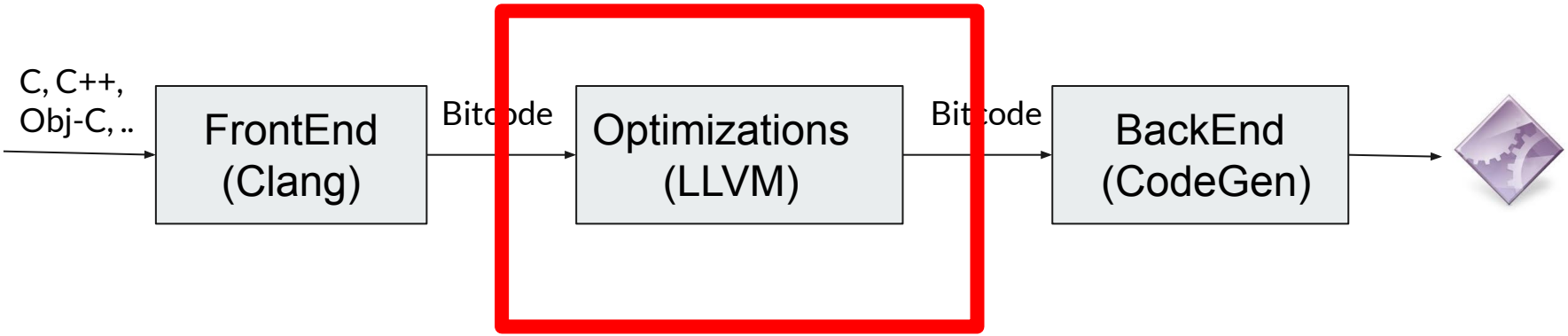
- Typed, SSA based IR: Source independent optimizations.
- Extremely extensible.
- Slow :(

CodeGen: The Backend

- Architecture dependent code generation.

- Extensible.

In this talk...


C, C++,
Obj-C, ..
FrontEnd
(Clang)
Bitcode
Optimizations
(LLVM)
Bitcode
BackEnd
(CodeGen)

Extending LLVM Optimizations

- Commonly called as **LLVM Passes.**
- Work on LLVM IR with *a lot* of support from LLVM.
- Common tasks: **Instrumentation and Static analysis**.

Instrumentation

- Add code into the program being compiled.

- The code will be executed as part of the program.
  - Example: Dynamic data flow analysis.

- Sanitizers[1]: Address Sanitizer, Undefined Behavior Sanitizer, API Sanitizer, etc.

[1][S&P 19] SoK: Sanitizers for Security

Let’s write a simple instrumentation pass

Logs all reads and writes to memory addresses.

How to log all the memory accesses?

1. Identify all the memory access instructions.

2. Add instructions to log the information.

How to log all the memory accesses?

1. Identify all the memory access instructions.

   `load` and `store`

2. Add instructions to log the information.

   Lets just `call` a function which will take care of logging.

Instrumentation is used *a lot* in Fuzzing.

[USENIX 13] Dowser

[S&P 18] Agora

[S&P 19] Razzer

...

https://scholar.google.com/scholar?hl=en&as_sdt=0%2C21&q=fuzzing%2Bllvm&btnG=&oq=llvm

We can also do static analysis!!

- Loop exit points.

- Functions which has at least one pointer argument.

Idea: Quine Fuzzing!!

- A program that fuzzes itself.

- Identify tainted data **right after input parsing but before input processing**.

- For every run, fill the tainted data with random values.

If interested..

- Whole-program-analysis.
- Dynamic Taint analysis.
- Using other frameworks: SVF, Phaser

Thank You!!!

"if you find yourself drinking a martini and writing programs in garbage-collected, object-oriented Esperanto, be aware that the only reason that the Esperanto runtime works is because there are systems people who have exchanged any hope of losing their virginity for the exciting opportunity to think about hex numbers and their relationships."

- James Mickens