Introduction to Compilers and Stages of Compilation

Ashutosh Pandey

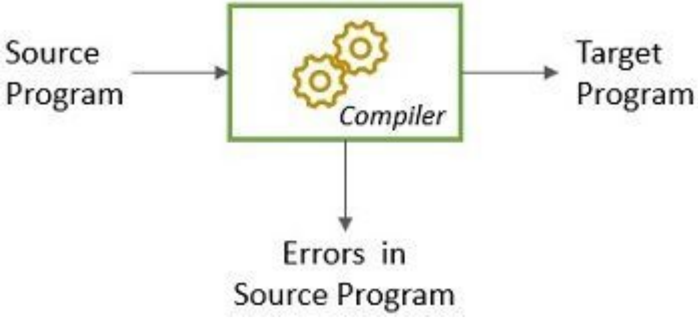
Shraiysh Vaishay

Agenda

- What are compilers?
- Usage and scope
- Compilers Today
- Stakeholders in the compiler industry
- What do compiler engineers work on?
- LLVM and its significance. What is LLVM IR and why we need it?
- Hands-on with LLVM
- Diving into the source code
- A few pointers
- Reach Us

What are Compilers?

- Translating computer code written in one language (source) to another (target).
- The target language may be low-level such as assembly, object code etc. Or it may be high-level such as Babel.
- Compilers also provide optimizations (see: Don't Help the Compiler) and error handling.



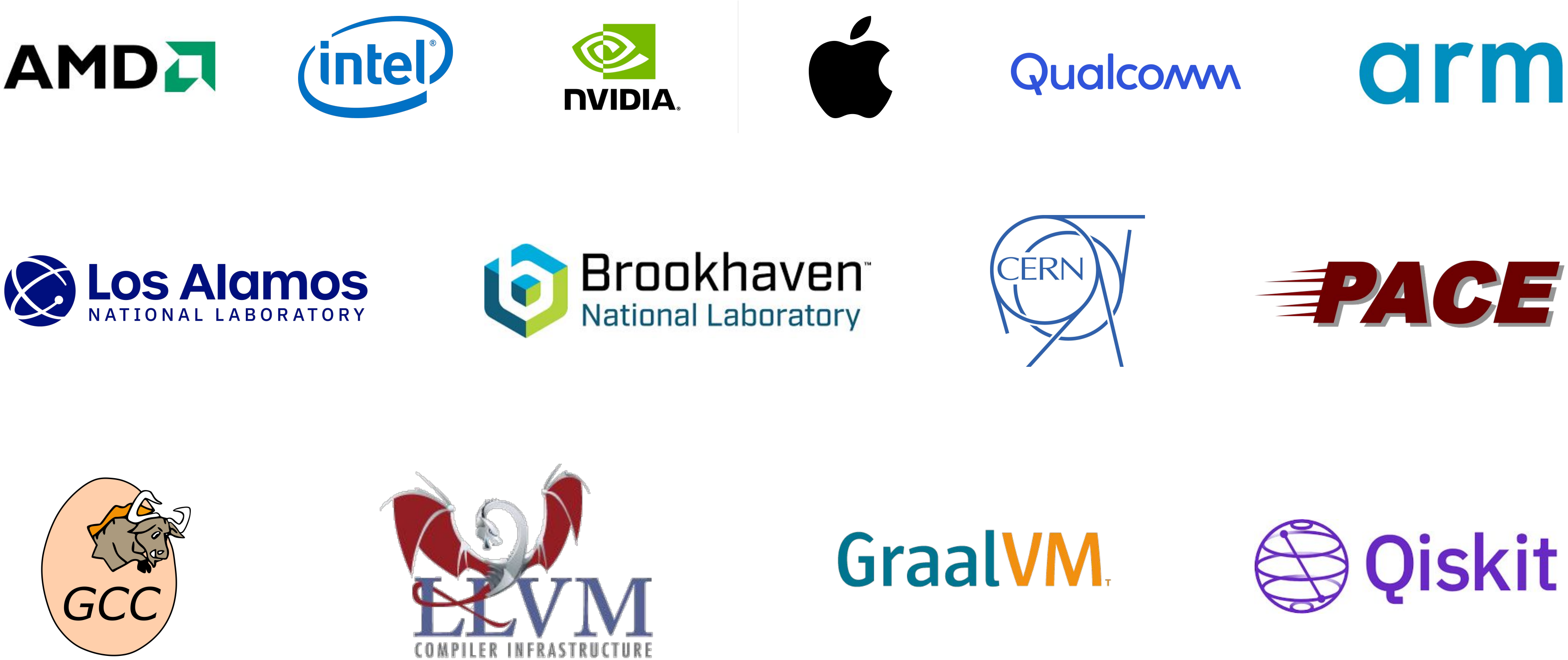
Usage and Scope

- Domain specific and device specific processing. Eg. Halide is a DSL for image processing and computational photography.
- Tools such as linters (clang-tidy), code formatters (ClangFormat), code completion (Intellisense), static analysis (Clang Static Analyzer), Debuggers etc.
- High Performance Computing (HPC) makes use of optimizing compilers for applications related to weather modelling, healthcare, physics simulations etc.
- Translating HDL’s (Hardware Description Languages) to the schematic of circuits and EDA tools. Eg: LLVM-CIRCT.

Compilers Today

- Many compilers exist for C/C++/Fortran as these are the main HPC languages.
- Open Source compilers as Clang/LLVM and GCC.
- Proprietary compilers such as ICC, AOCC, PGI, Cray Compiler, ARM etc.
- Many new languages - Rust and Julia have features such as memory safety, multiple dispatch and optional typing. These communities are very active.
- Upcoming compilers such as F18.
- Machine learning techniques such as RL are being explored for compiler optimizations.

Stakeholders In the Compiler Industry


AMD
intel
NVIDIA
Qualcomm
arm
Los Alamos NATIONAL LABORATORY
Brookhaven National Laboratory
CERN
PACE
GCC
LLVM COMPILER INFRASTRUCTURE
GraalVM
Qiskit

What do Compiler Engineers work on?

- **Optimizations**: figuring out ways to extract the maximum performance out of a system. Examples include vectorization, inlining, loop unrolling etc.
- **Implementing new features**: every few years new features are added to C++. The standards need to be implemented by compiler engineers.
- **Verification/Validation**: compilers need to be tested thoroughly for correctness as developers rely on them. This process includes lots of scripting, debugging etc.
- **Research**: research into new kinds of tools, techniques and processes is going on all the time as new hardware and software paradigms come to the fore. Example: llvm-bolt (from facebook).

What is LLVM and Why is it so important?

- Created by Vikram Adve and Chris Lattner in 2000 at UIUC. Originally LLVM stood for Low Level Virtual Machine.
- It is a **modular** set of compiler and toolchain technologies that can be used to create a frontend for any programming language and a Backend for any ISA.
- It currently has support for languages such as C, C++, Fortran, Rust, Objective C/C++, Julia, Haskell, Halide, Common LISP, Scala and many others.
- It has a permissive open source license that enables collaboration and commercial use.
- LLVM IR serves as a portable, high level assembly language that makes many of these optimizations and modularity possible.

Hands On with LLVM

Getting LLVM

- To use LLVM for compilation - https://releases.llvm.org/download.html
- To dive into the source of LLVM - https://llvm.org/docs/CMake.html

  -
    ```
    cmake -G "Ninja"  \
    -DCMAKE_BUILD_TYPE=Release \
    -DLLVM_TARGETS_TO_BUILD="X86" \
    -DLLVM_ENABLE_PROJECTS="clang" \
    -DLLVM_ENABLE_ASSERTIONS=ON \
    -DLLVM_ENABLE_LLD=On \
    -DCMAKE_EXPORT_COMPILE_COMMANDS=On ../llvm
    ```

Hands On with LLVM (Using LLVM for compilation)

Compiling Hello World

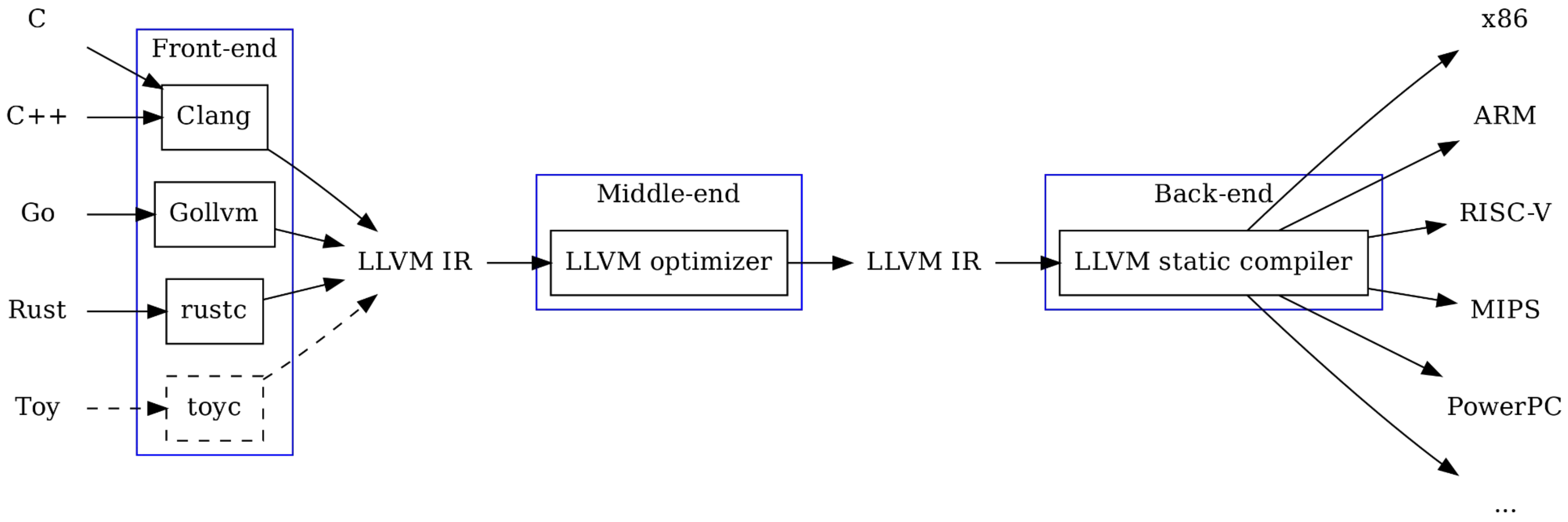


Image Credits: https://blog.gopheracademy.com/advent-2018/llvm-ir-and-go/

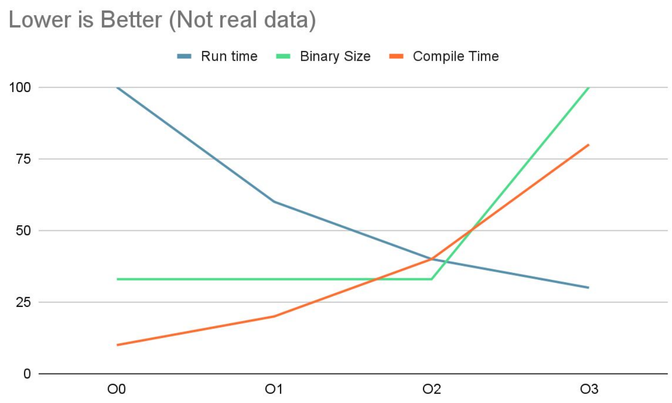
Frontend with C/C++ (Language Dependent)

Input: C file, Output: LLVM IR File

- Preprocessing
  - `clang -E example.c`
  - Replace all #defines, #includes etc. with code
  - example
- AST and semantics
  - `clang -Xclang -ast-dump example.c`
  - example
- IR Generation
  - `clang -Xclang -disable-O0-optnone -S -emit-llvm example.c`
  - example

Middle end (Language and Machine Independent)

Input: LLVM IR File, Output: Optimized LLVM IR File



More options for different compilation speeds, binary size, and execution speeds

example

Middle end (Language and Machine Independent)

Input: LLVM IR File, Output: Optimized LLVM IR File

- Run transformations
  - `opt -dce -S sample.ll`
  - `opt -passes='dce' -S sample.ll`
  - example
- Run analysis
  - `opt -passes='print<scalar-evolution>' -disable-output sample.ll`
  - example
- Pass pipeline
  - `opt -passes='pass1,pass2' /tmp/a.ll -S`
- Many options related to passes and debugging them in `opt-15 --help`

Backend (Machine Dependent)

- Assembler (llc -march=”x86”): example
- Linker: ld, lld, gold etc.
- Tooling such as LLVM MCA (machine code analyzer). An example is here.
- Tools such as Godbolt have an analysis mode to give you machine code level information on code execution. example.
- Timeline mode (-timeline -mcpu=name) can give Decode, execute and wait information for each instruction. example
- We’re still beginners when it comes to backend stuff, so feel free to research this at your own pace!

Diving into the source code

Subprojects

- >15 subprojects in the monorepo llvm/llvm-project
- 2 incubated projects

To get involved, two main questions -

- What to work on?
- How to work on it?

Diving into the source code (answering the WHAT)

What can we do? (Not exhaustive, there are many more things to do)

- Implement new and missing language constructs
  - https://clang.llvm.org/cxx_status.html
- Fix bugs in existing language constructs
  - https://github.com/llvm/llvm-project/issues
- Implement new static analysis tools on clang for C/C++ programs
  - https://clang.llvm.org/docs/RAVFrontendAction.html
  - https://clang.llvm.org/docs/LibASTMatchersReference.html
- Find new optimizations from Research and bring them to the LLVM Project.
  - https://scholar.google.com/scholar?q=llvm&hl=en&as_sdt=0,5
- Implement/Improve static analysis tools (sanitizers etc)
  - https://clang.llvm.org/docs/ThreadSanitizer.html
  - https://clang.llvm.org/docs/MemorySanitizer.html
- Work on MLIR, LLDB, Flang, Bolt, Libc and all the other subprojects.
  - Very similar skill set is required for all the projects, and all of them are very exciting.
- Open Projects at LLVM
  - https://llvm.org/OpenProjects.html
  - https://mlir.llvm.org/getting_started/openprojects/

Diving into the source code (answering the HOW)

After deciding what to do, discuss how to do it.

- Discussion forum - https://discourse.llvm.org/
- Discord - https://discord.gg/xS7Z362

For example -

- https://discourse.llvm.org/t/rfc-improving-clang-s-diagnostics/62584
- https://discourse.llvm.org/t/rfc-introduce-ml-program-dialect-and-top-level-ops-proposal-v2/60907
- https://discourse.llvm.org/t/rfc-a-unified-lto-bitcode-frontend/61774/32

Feel free to reach out to us.

Some pointers for working on LLVM

- Use assertions, a lot of them.
- The dump() function.
- Don't be scared of arc/phabricator. It has a short and steep learning curve.
- Use intellisense/ctags - the codebase is massive.
- Add tests for everything. If you don’t know where, ask someone.
- Read the documentation/comments in code and also comment your code.
- Use cmake options efficiently to reduce build times.

Reach Us

@shraiysh	@ashupdsce
calendar	calendar