LLVM – Infrastructure

Jan Moritz Witt

12th November 2019

Structure

- What is a compiler?
- Structure of a compiler
- LLVM project
- LLVM core
- Primary sub-projects of LLVM

What is a compiler?

- Translator of a source code
- Typically: From high-level language to binary code
- Checks for errors
- Optimizes
- Creates an object file

- Problem: from source language into target language only

Compiler structures

- Different ways to build a compiler
- Classifying by the number of passes needed
- Splitting into small programs

- Structure of discussed compilers:
  - Front end
  - Middle end
  - Back end

Front end

- Takes source code as input
- Three phases:
  - Lexical analysis
  - Syntax analysis (Parsing)
  - Semantic analysis
- Generates error/warning messages
- Transforms source code into intermediate representation (IR)

```
int mul_add(int x, int y, int z) {
  return x * y + z;
}
```

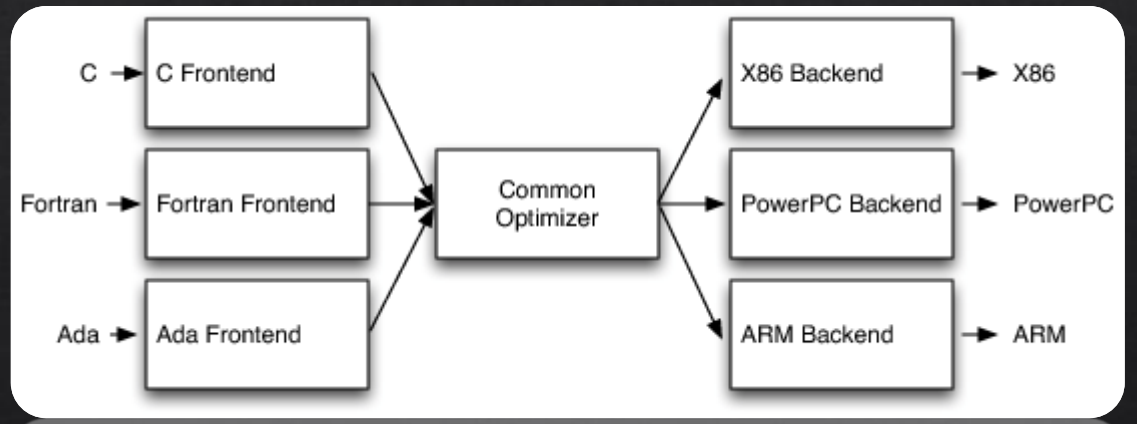
[1]

Middle end

- Takes IR as input
- Performs machine independent optimizations
  - Dead code elimination
  - Unreachable code removal
  - Discovery of constants and relocation
- Outputs an optimized IR
- Benefit of this structure: sharing of optimization of the middle end

Back end

- Takes optimized IR as input
- Performs more analysis, transformation and optimization
  - Peephole optimization
- Generates target dependent code



[2]

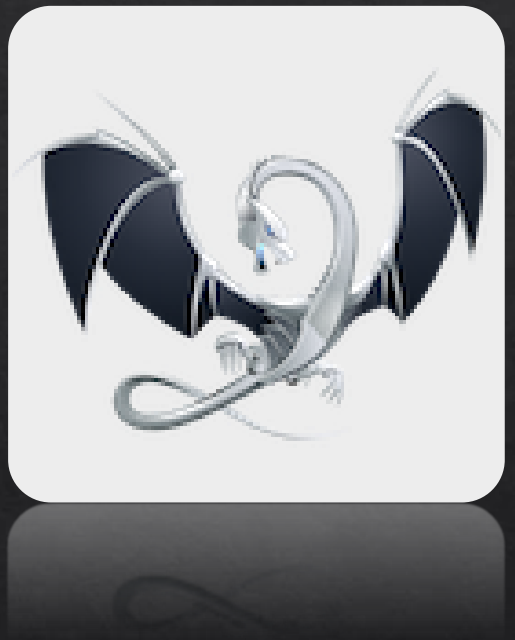
Intermediate Representation (IR)

- Representation of code, which is mostly independent of source and target language
- Break difficult translation from source to target
- To share target independent optimization part

- Different types
  - Structured
  - Flat, tuple-based
  - Flat, stack-based

LLVM

- Released 2003, written in C++
- Library structure
- Collection of modular and reusable compiler and tool technologies
- Open Source
- Capable of a wide spectrum of tasks
  - JIT compiling
  - Compiling code of supercomputers



[1]

LLVM core: Optimizer

- Target independent
- Implemented as passes:
  - Analysis passes (e.g. –da)
  - Transform passes (e.g. –constmerge)
  - Utility passes (e.g. –verify)

LLVM core: Code generator

- Translation framework
- Designed for many popular but also some uncommon CPUs
- Assembly or binary machine code form
- Possible to add further CPUs

LLVM core: IR

- Used throughout all phases of the LLVM compilation
- Provides type safety, low-level operations and flexibility
- Capable to represent ‚all' high-level languages
- Designed in three different but equal forms:
  - In-memory compiler representation
  - On-disk bitcode representation
  - Human readable assembly representation
- SSA-based

```
int mul_add(int x, int y, int z) {
  return x * y + z;
}
```

[1]

```
define i32 @mul_add(i32 %x, i32 %y, i32 %z) {
entry:
  %tmp = mul i32 %x, %y
  %tmp2 = add i32 %tmp, %z
  ret i32 %tmp2
}
```

[1]

LLVM core: Support

- Extensive documentations
  - Introduction
  - User guides
  - Search Page
- Tutorials
  - How to invent your own language
- Large community

Clang

- Front end for C family
  - Single unified Parser
- Goal: fast compiling with low memory use
  - ‘about 3x faster than GCC when compiling Objective-C code in a debug configuration’ [1]
- Compatible to GCC by ignoring unwanted extensions
- Tight integration with IDE
- Library-based structure
- Clang static analyzer

LLDB Debugger

- Invented for modern multi-thread programs
- No GPL required
- Converts debug information into clang type
- Up-to-date language support
- ‘Fast and much more memory efficient than GDB at loading symbols’ [1]

Projects for C++

- Libc++
  - C++ standard library
- Libc++ ABI
  - low level support for the C++ standard library
- Compiler-rt
  - Runtime libraries

Projects for parallel programming

- OpenMP
  - Build an executable openMP program
- Libclc
  - Uses the structure of the software

Further projects

- Polly
  - Data-locality optimizer
- Klee
  - Symbolic virtual machine
- LLD
  - Linker with drop-in replacement
  - ‘When you link a large program on a multicore machine, you can expect that LLD runs more than twice as fast as the GNU gold linker.’ [1]

Summary

- LLVM is a open source compiler project
- Library – based
- Own IR language
- Works with ‚all' high-level languages
- Well documented
- Main projects: Optimization and code generation
- Faster than most alternative compilers
- Useable for a large variety of task due to its flexibility

Literature

1. http://llvm.org/
2. https://www.aosabook.org/en/llvm.html#fig.llvm.rtc
3. https://en.wikipedia.org/wiki/Compiler
4. https://www.geeksforgeeks.org/compiler-lexical-analysis/
5. https://www.guru99.com/syntax-analysis-parsing-types.html
6. https://www.tutorialspoint.com/compiler_design/compiler_design_semantic_analysis.htm
7. https://cs.lmu.edu/~ray/notes/ir/
8. http://ssabook.gforge.inria.fr/latest/book.pdf