

888: LLVM

Week 1 - Introduction

Tobias Grosser



Organization

- ▶ DL 698 on Tuesdays at 2pm
- ▶ Need any help?
 - ▶ My office: Dreese 572
 - ▶ E-Mail: grosser.13@osu.edu
 - ▶ Website: <http://www.fim.uni-passau.de/~grosser>
- ▶ Weekly exercises: Given on Tuesday - Due to Sunday 6pm

What is LLVM?

- ▶ Project to build compiler infrastructure
- ▶ Subprojects
 - ▶ **LLVM - Infrastructure**
 - ▶ Clang/Dragonegg - C/C++/Objective C compiler
 - ▶ libc++ - C++ standard library
 - ▶ lldb - Debugger
- ▶ Written in C++
- ▶ OpenSource - BSD like license
- ▶ www.llvm.org

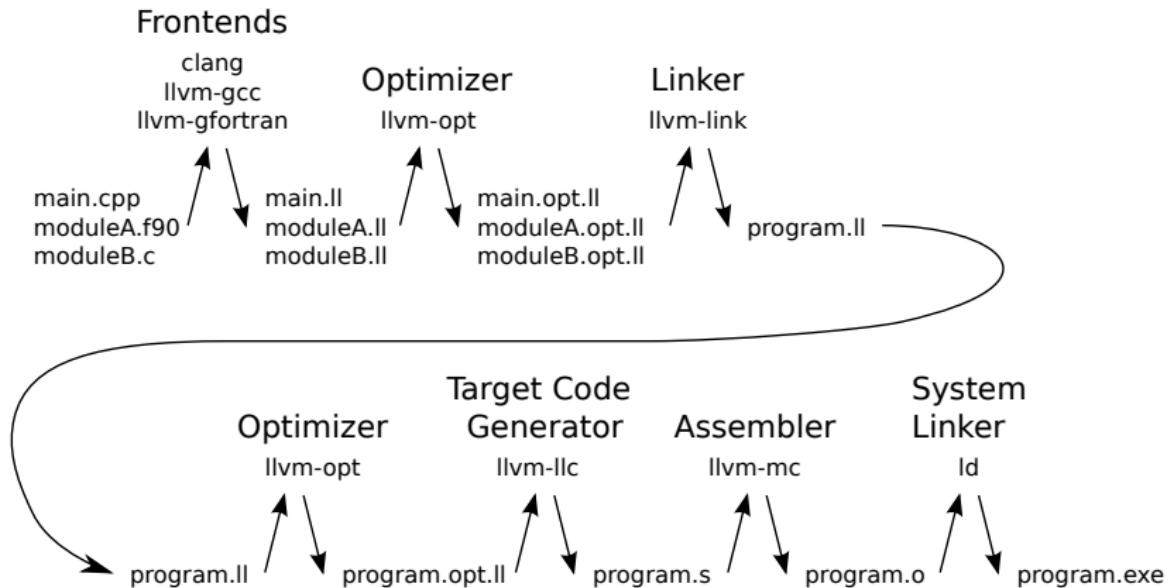
LLVM everywhere?

A word cloud visualization centered around LLVM, showing its widespread use across various projects and companies. The words are colored by project, with larger words indicating more prominent use.

Key visible words and their associated projects:

- OpenJDK (Java/Scala)**: Cray, FreeBSD, ARM, Qualcomm, Intel, LunarGlass (OpenGL Shader Optimizer)
- clang (C/C++/Objective C)**: unladen-swallow (Python), Cray, dragonegg (C/C++/Ada/FORTRAN/Go), Rubinius (Ruby), Google, Mono (C#/F#/...), emscripten (LLVM to JavaScript), Hydra (Adobe Image Processing Language), Trident (C to VHDL), Click-To-Net, FPGA (Package Processing), Apple, Sony, D
- NativeClient**: AMD & Nvidia OpenCL, Haskell
- C-To-Verilog (IBM)**, Gallium3D (OpenGL Software Rendering), LightSpark (Flash/ActionScript)

A static compile flow



LLVM-IR

- ▶ Base of all LLVM optimization passes
- ▶ Low level assembly like language
- ▶ Target independent (frontends might introduce target information)
- ▶ Register machine, infinite number of named registers
- ▶ Each register is assigned exactly once (SSA)
- ▶ Load/Store Architecture
- ▶ Defined at <http://llvm.org/docs/LangRef.html>

LLVM-IR

Representations:

- ▶ In memory representation - C++ data structures
- ▶ Bitcode - Binary file (.bc)
- ▶ Human readable form - Text file (.ll)

All representation are equivalent. To translate from one to another:

```
|> opt -S program.bc -o program.ll  
|> opt      program.ll -o program.bc
```

Creating LLVM-IR

main.c

```
int main() {  
    return 42;  
}
```

```
clang -S -emit-llvm main.c -o main.ll
```

```
; ModuleID = 'main.ll'  
target datalayout = "e-p:64:64:64-i1:8:8-i8:8:8-i16:16:16-i32:32:32-f32:32:32-f64:64:64-nan:  
target triple = "x86_64-unknown-linux-gnu"  
  
define i32 @main() nounwind {  
    %1 = alloca i32, align 4  
    store i32 0, i32* %1  
    ret i32 42  
}
```

Compiling LLVM-IR

```
> llc main.ll -o main.s && gcc main.s -o main.exe
> ./main.exe
> echo $?
42

> lli main.ll
> echo $?
42
```

- ▶ llc - LLVM to Assembly compiler
- ▶ lli - Just in time compiler

Our first function

Create a function that calculates n^2

Our first function

Create a function that calculates n^2

```
define i32 @pow(i32 %number) {  
%pow = mul i32 %number, %number  
ret i32 %sqrt  
}  
  
define i32 @main() nounwind {  
%result = call i32 @pow(i32 7)  
ret i32 %result  
}
```

Optimize our first function

```
| > opt -O3 pow.ll -o pow.opt.ll -S
```

Optimized function

Optimize our first function

```
| > opt -O3 pow.ll -o pow.opt.ll -S
```

Optimized function

```
define i32 @pow(i32 %number) nounwind readnone {
    %sqrt = mul i32 %number, %number
    ret i32 %sqrt
}

define i32 @main() nounwind readnone {
    ret i32 49
}
```

Multiple modules

```
| > llvm-link module_main.ll module_pow.ll \
|   -o module_combined.ll -S
```

Exercise

- ▶ Install LLVM from the source
- ▶ Git repository at: <http://llvm.org/git/llvm.git>
- ▶ Install Clang
- ▶ Git repository at: <http://llvm.org/git/clang.git>
- ▶ Take a small C project and create some