Contact Information

• Instructor: Aaron Smith
  • Email: aaron.l.smith@ed.ac.uk
  • Office: IF 1.29

• TA for LLVM: Andrej Ivanis
  • Email: andrej.ivanis@ed.ac.uk
  • Office: IF 1.07
Schedule

• Week 1
  • Nov 13: Introduction to LLVM
  • Nov 16: How to Write an LLVM Pass
    • LAB: Your First LLVM Pass

• Week 2
  • Nov 20: LLVM Internals
  • Nov 23: Dead Code Elimination
    • LAB: Dead Code Elimination

• Week 3
  • Nov 27: Dataflow Analysis
  • Nov 30: Compiler Trivia!!
    • LAB: Work on Final Project
Project Overview

• LLVM is written in C++
  • But no templates or tricky C++ code
  • If you know C or Java you will be OKAY

• LLVM sources are hosted in both SVN and Git
  • You can use either but we will only discuss Git in the course
  • You need to submit the final project to Gitlib

• Project will be graded on Linux
  • LLVM works on OS X and Windows but we will only grade on Linux
  • If you work on other platforms make sure it also works on Linux!

• Final project is due by Monday, January 14, 2019 at 10am
Getting Started

• Read the original LLVM paper (optional)
  • LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation, Chris Lattner and Vikram Adve, CGO 2004
  • http://dl.acm.org/citation.cfm?id=977673

• Read the Dr Dobbs article on LLVM (optional)
  • The Design of LLVM, Chris Lattner, 2012
  • http://www.drdobbs.com/architecture-and-design/the-design-of-llvm/240001128

• Look at LLVM.org
What is LLVM?

• An open source framework for building tools
  • Tools are created by linking together various libraries provided by the LLVM project and your own

• An extensible, strongly typed intermediate representation, i.e. LLVM IR
  • [https://llvm.org/docs/LangRef.html](https://llvm.org/docs/LangRef.html)

• An industrial strength C/C++ optimizing compiler
  • Which you might know as clang/clang++ but these are really just drivers that invoke different parts (libraries) of LLVM
History of LLVM

• Started by Chris Lattner at UIUC ~2000
  • First commercial use was as an OpenGL Jitter on OS X at Apple

• Evolved over many years into a complete C/C++ compiler which until recently required parts of GCC
  • llvm-gcc

• Many uses of LLVM in the world today
  • OS X (XCode) platform compiler
  • FreeBSD platform compiler
  • Google Android NDK compiler
  • ARM reference compiler
  • Microsoft DirectX shader compiler
  • NVIDIA CUDA compiler
Typical Optimizing Compiler
LLVM Optimizing Compiler

- **clang**: C/C++, FORTRAN, Python, Ruby, Javascript, Objective-C, Haskell, Lua, ...

- **opt**: Loop unrolling, Dead code elimination, Common subexpression elimination, ...

- **llc**: ARM, x86, PowerPC, MIPS, SystemZ, Hexagon, WebAssembly, ...

- **lld**: a.out

- **libraries/objects**:
What Tools Does LLVM Provide?

• Lots! clang, opt, llc, lld are just four of many
What Optimizations Does LLVM Support?

• Lots! Let’s see by running ‘opt --help'
How to Get the LLVM Sources

• LLVM is split into multiple Git repositories
  • For this class you will need the clang and llvm git repos

• Choose a directory to clone the repos into
  • The LLVM repo is always cloned first
  • Other repos are cloned inside the LLVM directory

```bash
cd directory-to-clone-into
git clone https://github.com/llvm-mirror/llvm

cd llvm/tools
git clone https://github.com/llvm-mirror/clang
```
How to Build LLVM

• LLVM requires Cmake version 3.4.2+ to generate the build files
  • The latest version of Cmake is already installed on DICE

• By default Cmake generates a debug version of the build files that compile LLVM at the lowest optimization level and with assertions enabled and debug symbols
  • Easiest to debug but slow to compile large programs and takes up the most disk space

• Cmake supports several build systems
  • make, XCode, Visual Studio, Ninja and more
  • If you are working on DICE you will generate Makefiles for make

• Create a new directory outside the LLVM source directory for your build

```bash
cd directory-for-build
cmake path-to-llvm-sources
cmake --build .
```
Let’s Try Compiling a Program with LLVM
How to Generate LLVM IR from Source

• To generate LLVM IR use clang with ‘-emit-llvm’ option
  • ‘–S’ generates a text file and ‘–c’ generates a binary
    • clang foo.c –emit-llvm –S
    • clang foo.c –emit-llvm –c

• To convert a binary file (.bc) to a text file (.ll) use the llvm disassembler
  • llvm-dis foo.bc

• To convert a text file (.ll) to a binary file (.bc) use the llvm assembler
  • llvm-as foo.ll
Let’s Look Closer at LLVM IR

• Some characteristics of LLVM IR
  • RISC-like instruction set
  • Strongly typed
  • Explicit control flow
  • Uses a virtual register set with infinite temporaries (%)
  • In Static Single Assignment form
  • Abstracts machine details such as calling conventions and stack references

• LLVM IR reference is online
  • https://llvm.org/docs/LangRef.html
Do you remember how to generate bitcode?

```cpp
int x = 7;
int main() {
    int n = 0;
    if (x != 0)
        n++;
    return n;
}
```

Where are the virtual registers?
What are the types?
Where is the control flow?
What does ‘@x’ mean?
How about ‘alloca’?
Optimizing LLVM IR

- Previous LLVM IR was not optimal
- We know the program returns 1 by looking at it
- Let’s optimize the bitcode with ‘opt’
  - By default ‘opt’ does nothing, you must specify an optimization such as ‘–O2’

```c
int x = 7;
int main() {
    int n = 0;
    if (x != 0)
        n++;
    return n;
}
```

```c
define i32 @main()
local_unnamed_addr #0 {
    %1 = load i32, i32* @x, align 4
    %2 = icmp ne i32 %1, 0
    % = zext i1 %2 to i32
    ret i32 %.
}
```
Generating Machine Code from LLVM IR

- Use ‘llc’
Next Time

• How to write your own LLVM pass