Advanced Compiler Construction Labs 2012

Part 1 – LLVM IR

Erik Hansson
eriha@ida.liu.se

Overview

- Low Level Virtual Machine – LLVM
  - http://llvm.org
  - Modern module based compiler infrastructure
  - Open Source
  - Written in C++ (mainly)
  - Started 2000 at University of Illinois at Urbana–Champaign by Chris Lattner and Vikram Adve.
  - In 2005 Lattner got hired by Apple to work with LLVM.
  - Popular, check for example www.compilerjobs.com

LLVM get started

- In Linux
  - Download LLVM 2.6 source
    - http://llvm.org/releases/
  - Unpack it (into LLVM_DIR)
  - Download Clang source
    - http://llvm.org/releases/
  - Unpack Clang source into LLVM_DIR/tools/clang/
  - In LLVM_DIR run ./configure and ./make -j 2
  - Add LLVM_DIR/Release/bin/ to your PATH (edit .bachrc)
- LLVM also works with Mac OS X , Windows (Cygwin) and Solaris.
  - Feel free to try it.
- If you don't have a Linux machine, we can create an account.
- It is OK to try with the latest LLVM version (3.0) as well, but some thing may of course have changed

LLVM first try

- Write a small C program.
- Compile and run
  - `clang -o test1 test1.c`
  - `./test1`
  - Just like gcc.
- LLVM IR
  - `clang test1.c -S -emit-llvm`
  - This produces `test1.s`
    - Take a look on the output your self!
  - Convert to bitcode: `llvm-as test1.s`
    - Produces `test1.s.bc`

```
#include <s t d i o . h>
int main ( )
{
  p r i n t f ( " H e l l o LLVM +
CLANG! \n" ) ;
}
```

LLVM Passes

- A pass can do analysis or transformations on the IR.
  - Simple example from the LLVM documentation
    - http://llvm.org/releases/2.6/docs/WritingAnLLVMPass.html
    - http://llvm.org/docs/WritingAnLLVMPass.html
    - Create dir: LLVM_DIR/lib/Transforms/Hello
    - Copy the make file from the page
  - For each function in a program, print out its function name.
  - `opt -load ../../../Debug/lib/Hello.so -hello < hello.bc > /dev/null`

```
#include "llvm/Pass.h"
#include "llvm/Function.h"
#include "llvm/Support/raw_ostream.h"
using namespace llvm;
namespace {
  struct Hello : public FunctionPass {

    static char ID;
    Hello() : FunctionPass(&ID) {}
    virtual bool runOnFunction(Function &F) {
      errs() << "Hello: " << F.getName() << "\n";
      return false;
    }
  };
  char Hello::ID = 0;
  RegisterPass<Hello> X("hello", "Hello World Pass");
}
```

Part 1: Exercise 1

- Write a simple pass that calculates how many calls there are to `printf()` function. Output can for example be something like:

main:

printf(): 2 calls

foo:

printf(): 23 calls

```
for ( i =2;  i <4 ; i ++)
        {
          p r i n t f ( " H e l l o LLVM + CLANG! \n" ) ;
        }
```

- It should handle loops, at least if the number of times the loop body will execute can be determined statically, and the loops are perfectly nested.

Part 1: Exercise 2a

- Write a pass (or a set of passes) that recognizes the vector init (initialization of a vector with a constant)
- Example:

```
for ( i =0;   i < 5 ; i ++)
    v [ i ]= 42;
```

- The pass should report:
  - Matched computation, operand, size etc.
  - Alternatively replace matched loops by an equivalent function call.

Part 1 Exercise 2b

- **Optional** – will give bonus in the exam
- Write a pass that recognizes dot product calculation:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=0}^{n} a_i b_i$$

- More details in the lab instruction.

Part 2

Back-end

Some Target Code Generation

Exercise by Mattias Eriksson, 2010

Part 2: Adding an instruction to LLVM

- In this assignment you will add a theoretical instruction to the *Sparc* target.
- Create an add-instruction that takes three operands: **`addthree a,b,c,d`**
- This instruction should match the computation `d=a+b+c`, where:
  - `a,b,c` are all integers located in registers,
  - `a,b,c` are all floats located in registers,
  - `a,b` are integers in registers, and `c` is an immediate value. What is the largest value that `c` can have in this instruction? (This value depends on how you made your implementation.)

Part 2: Hints

- You can compile your example program to sparc assembler like this:

```
clang -O2 -S -emit-llvm foo.c
llvm-as foo.s
llc -march=sparc -mcpu=generic -stats
-asm-verbose foo.s.bc
```

- Look at `foo.s.s` to see the result.
- Declare your variables to be volatile:

```
volatile int x;
```

What should be in your report

- Strategy/Approach for solving the problems.
- Results of your tests, with comments.
- Well commented implementation source code.
- Test programs, in C and LLVM assembly. Do not forget to write which built in passes you used and if you used any optimizations etc. Their invocation order is interesting as well.
- Also write which version of LLVM you used and which platform you used. If you downloaded it via svn please write the revision number you used.
- Nice if you have a (small) discussion part where you can take up problems you had.
- Send it the report to me: erik.hansson@liu.se
- Check deadline on home-page!

Hints: Read, read and try

- Read a lot – LLVM is large
  - Documentation
    - http://llvm.org/docs/
    - http://llvm.org/releases/2.6/docs/index.html
  - API
    - http://llvm.org/doxygen/

      (svn version of LLVM, but still useful)
  - Source files
    - .h
    - .cpp
  - Mailing list
    - Brows and search it, for example:
      - http://dir.gmane.org/gmane.comp.compilers.llvm.devel