888: LLVM

Week 2 - LLVM-IR

Tobias Grosser


Google™
SUMMER OF
CODE
2011
CODE.GOOGLE.COM/SOC

- Work three months on OpenSource
- 5000$
- 1000 students every year
- More than 100 projects - including LLVM and GCC
- `http://code.google.com/soc`

Exercise - 1

- Install llvm/clang
- Modify the pow function to be called using a function pointer. Can opt -O3 still optimize the code?

Exercise - 2

Write a C main function that calls the pow function (written in LLVM-IR), as seen in class, for the integers from 0 to 100 and prints the result to stdout.

- Use clang to compile your main function to LLVM-IR
- Have a look at the LLVM-IR of the main function right after export from clang and after running opt -O3. Can you see any difference?
- Use llvm-link to link the LLVM-IR of the main and the pow module (optimized and unoptimized) and execute it on your machine. Is there any performance difference.

Exercise - 3

- Compile bzip2 with link time optimizations using the clang -S -emit-llvm, opt -O3, llvm-link, opt -O3, llc, gcc sequence.
- Does the binary size change if you skip the optimizations after linking?
- Try to compress some files and see if the runtime changes?

Exercise - 4

- Compile bzip2 with link time optimizations using the clang -S -emit-llvm, opt -O3, llvm-link, opt -O3, llc, gcc sequence.
- Does the binary size change if you skip the optimizations after linking?
- Try to compress some files and see if the runtime changes?

Exercise - 5

Experts? Can you replace the pow function with a function that calculates the nth fibonacci number? Written in LLVM-IR and not using any loops, but functional programming techniques.

LLVM-IR - Overview

- Low level assembly like language
- Register machine, infinite number of (named) registers
- Each instruction defines a new (named) register
- Load/Store Architecture
- Defined at `http://llvm.org/docs/LangRef.html`

LLVM-IR - Types

- LLVM-IR is strongly typed
- Each register/pointer/function has an associated type
- No implicit type casts
- A program without casts is typesafe in the absence of memory access errors (e.g. array overflow)

LLVM-IR - Type classes

- **Primitive**
  integer, floating point, label, metadata, void, x86mmx,
- **Derived**
  array, function, pointer, structure, packed structure, vector, opaque

First class types - Non first class types

LLVM-IR - Integer types

- Fixed bitwidth
- Any bit width from 1 bit to $2^{23} - 1$
- Larger types as function parameters $\rightarrow$ backend dependent
- Signedness not defined

```
i1       ; Boolean type
i8       ; char
i32      ; 32 bit integer
i64      ; 64 bit integer
i121212  ; Very large integer
```

LLVM-IR - Floating point types

```
float          ;  32 bit
double         ;  64 bit
fp128          ; 128 bit (112-bit mantissa)
x86_fp80       ;  80 bit (X87)
ppc_fp128      ; 128 bit (two 64-bits)
```

LLVM-IR - Label type

- A (named) reference for a basic block

```
define i1 @foo() {
start:
  br label %next

next:
  br label %return

return:
  ret i1 0
}
```

LLVM-IR - Values

- Created through an instruction
- Global values
- Constants
- Undefined

```
%result = add i32 5, 10

i1 0
i32 15

float undef
```

LLVM-IR - Constants

```
i1 true                          ; Boolean constants
i1 false
i32 -1                           ; equal to 2^32 - 1

float 123.421                    ; Exact decimal notation
                                 ; ! 1.3 has infinite
                                 ; binary representation

float 1.23421e+2                 ; Exponential notation
double 0x432ff973cafa8000        ; Hexadecimal notation

type zeroinitializer             ; 'type' can be any type
```

LLVM-IR - Instructions

- Calculations
- Vector/structure management
- Type conversion
- Memory management
- Control flow instructions

LLVM-IR - Computational instructions

- Side effect free
- Take values as input
- Create a new register value

```
%sum = add i32 %a, %b
%product = fmul float %a, %b
%unsigned_div = udiv i32 %a, %b
%signed_div = sdiv i32 %a, %b
%division = fdiv float %a, %b
```

LLVM-IR - Computational instructions - Comparisons

```
%equal = icmp eq i32 %a, %b
%not_equal = float ne i5 %c, %d
%signed_less_than = icmp slt i3 %a, %b
%unsigned_less_than = icmp ult i5 %a, %b
```

LLVM-IR - Control flow instructions

- (Un)Conditional branch
- Switch
- Return
- Indirect branch, Invoke, Unwind

```
start:
  br i1 true, label %left, label %right
left:
  br label %join
right:
  br label %join
merge:
  ret i32 %joinedValue
```

LLVM-IR - PHI instruction

- Implements the Φ SSA instruction

```
start:
  br i1 true, label %left, label %right
left:
  %plusOne = add i32 0, i32 1
  br label %join
right:
  br label %join
merge:
  %joinedValue = phi i32 [ %plusOne, %left],
                         [ -1, %right]
  ret i32 %joinedValue
```

LLVM-IR - Call instruction

- Calls a function
- Saves the return value in a new register

```
%result = call i32 @pow(i32 7)
```

Exercise

Will be sent out tonight.