llvm-py: Writing Compilers Using Python

PyCon India 2010

Mahadevan R
mdevan@mdevan.org

September 8, 2010

Outline

- Compilers
- LLVM
- llvm-py

What is a Compiler?

- What is a compiler?

What is a Compiler?

- What is a compiler?
  - Something that transforms "source code" into "something else"

What is a Compiler?

- What is a compiler?
  - Something that transforms "source code" into "something else"
- What is "source code"?

What is a Compiler?

- What is a compiler?
  - Something that transforms "source code" into "something else"
- What is "source code"?
  - A *well-structured*, textual representation of a program

What is a Compiler?

- What is a compiler?
  - Something that transforms "source code" into "something else"
- What is "source code"?
  - A *well-structured*, textual representation of a program
  - Not: preprocessors, assemblers

What is a Compiler?

- What is a compiler?
  - Something that transforms "source code" into "something else"
- What is "source code"?
  - A *well-structured*, textual representation of a program
  - Not: preprocessors, assemblers
- What is "something else?"

What is a Compiler?

- What is a compiler?
  - Something that transforms "source code" into "something else"
- What is "source code"?
  - A *well-structured*, textual representation of a program
  - Not: preprocessors, assemblers
- What is "something else?"
  - Another well-structured textual representation

What is a Compiler?

- What is a compiler?
  - Something that transforms "source code" into "something else"
- What is "source code"?
  - A *well-structured*, textual representation of a program
  - Not: preprocessors, assemblers
- What is "something else?"
  - Another well-structured textual representation
  - Intermediate, binary representation (AOT compilers)

What is a Compiler?

- What is a compiler?
  - Something that transforms "source code" into "something else"
- What is "source code"?
  - A *well-structured*, textual representation of a program
  - Not: preprocessors, assemblers
- What is "something else?"
  - Another well-structured textual representation
  - Intermediate, binary representation (AOT compilers)
  - In-memory executable code (JIT compilers)

What is a Compiler?

- What is a compiler?
  - Something that transforms "source code" into "something else"
- What is "source code"?
  - A *well-structured*, textual representation of a program
  - Not: preprocessors, assemblers
- What is "something else?"
  - Another well-structured textual representation
  - Intermediate, binary representation (AOT compilers)
  - In-memory executable code (JIT compilers)
  - Executable images

Inside a Compiler

What does the compiler do with the source code?

- Lexical analysis produces tokens

Inside a Compiler

What does the compiler do with the source code?

- Lexical analysis produces tokens
- Parser eats tokens, produces AST

Inside a Compiler

What does the compiler do with the source code?

- Lexical analysis produces tokens
- Parser eats tokens, produces AST
- What is abstract about an AST?

Inside a Compiler

What does the compiler do with the source code?

- Lexical analysis produces tokens
- Parser eats tokens, produces AST
- What is abstract about an AST?
- What next after an AST?

Inside a Compiler

What does the compiler do with the source code?

- Lexical analysis produces tokens
- Parser eats tokens, produces AST
- What is abstract about an AST?
- What next after an AST?
- What is an intermediate form (IR)?

Inside a Compiler

What does the compiler do with the source code?

- Lexical analysis produces tokens
- Parser eats tokens, produces AST
- What is abstract about an AST?
- What next after an AST?
- What is an intermediate form (IR)?
- example: IRs in gcc: source $\rightarrow$ GENERIC $\rightarrow$ GIMPLE $\rightarrow$ RTL $\rightarrow$ backend

Inside a Compiler

What does the compiler do with the source code?

- Lexical analysis produces tokens
- Parser eats tokens, produces AST
- What is abstract about an AST?
- What next after an AST?
- What is an intermediate form (IR)?
- example: IRs in gcc: source $\rightarrow$ GENERIC $\rightarrow$ GIMPLE $\rightarrow$ RTL $\rightarrow$ backend
- What is Static Single Assignment (SSA) form?

Passes, Optimization and Code Generation

- What is a "pass"?

Passes, Optimization and Code Generation

- What is a "pass"?
- What is a "optimization"?

Passes, Optimization and Code Generation

- What is a "pass"?
- What is a "optimization"?
- What is a "code generator"?

Passes, Optimization and Code Generation

- What is a "pass"?
- What is a "optimization"?
- What is a "code generator"?
- What is a "code generator generator"?

Passes, Optimization and Code Generation

- What is a "pass"?
- What is a "optimization"?
- What is a "code generator"?
- What is a "code generator generator"?
- Buzzwords: Register allocator, vectorization, interprocedural and link-time optimizations, memory hierarchy optimizations

The Frontend and The Backend

- Everything before the first IR is usually called the frontend

The Frontend and The Backend

- Everything before the first IR is usually called the frontend
- And everything after as the backend

The Frontend and The Backend

- Everything before the first IR is usually called the frontend
- And everything after as the backend
- LLVM and llvm-py deal only with backends

The Frontend and The Backend

- Everything before the first IR is usually called the frontend
- And everything after as the backend
- LLVM and llvm-py deal only with backends
- Frontends tend to be simpler

The Frontend and The Backend

- Everything before the first IR is usually called the frontend
- And everything after as the backend
- LLVM and llvm-py deal only with backends
- Frontends tend to be simpler
- Many frontends for a backend is common (Scala, Groovy, Clojure: Java; C#, IronPython, etc: .NET)

The Frontend and The Backend

- Everything before the first IR is usually called the frontend
- And everything after as the backend
- LLVM and llvm-py deal only with backends
- Frontends tend to be simpler
- Many frontends for a backend is common (Scala, Groovy, Clojure: Java; C#, IronPython, etc: .NET)
- To build a front end in Python:

The Frontend and The Backend

- Everything before the first IR is usually called the frontend
- And everything after as the backend
- LLVM and llvm-py deal only with backends
- Frontends tend to be simpler
- Many frontends for a backend is common (Scala, Groovy, Clojure: Java; C#, IronPython, etc: .NET)
- To build a front end in Python:
  - hand-coded lexer, (rec-desc) parser

The Frontend and The Backend

- Everything before the first IR is usually called the frontend
- And everything after as the backend
- LLVM and llvm-py deal only with backends
- Frontends tend to be simpler
- Many frontends for a backend is common (Scala, Groovy, Clojure: Java; C#, IronPython, etc: .NET)
- To build a front end in Python:
  - hand-coded lexer, (rec-desc) parser
  - Antlr (generates Python)

The Frontend and The Backend

- Everything before the first IR is usually called the frontend
- And everything after as the backend
- LLVM and llvm-py deal only with backends
- Frontends tend to be simpler
- Many frontends for a backend is common (Scala, Groovy, Clojure: Java; C#, IronPython, etc: .NET)
- To build a front end in Python:
  - hand-coded lexer, (rec-desc) parser
  - Antlr (generates Python)
  - Yacc, Bison, Lemon (generates C, wrap to Python)

The Frontend and The Backend

- Everything before the first IR is usually called the frontend
- And everything after as the backend
- LLVM and llvm-py deal only with backends
- Frontends tend to be simpler
- Many frontends for a backend is common (Scala, Groovy, Clojure: Java; C#, IronPython, etc: .NET)
- To build a front end in Python:
  - hand-coded lexer, (rec-desc) parser
  - Antlr (generates Python)
  - Yacc, Bison, Lemon (generates C, wrap to Python)
  - Sphinx (C++, wrap to Python)

The Frontend and The Backend

- Everything before the first IR is usually called the frontend
- And everything after as the backend
- LLVM and llvm-py deal only with backends
- Frontends tend to be simpler
- Many frontends for a backend is common (Scala, Groovy, Clojure: Java; C#, IronPython, etc: .NET)
- To build a front end in Python:
  - hand-coded lexer, (rec-desc) parser
  - Antlr (generates Python)
  - Yacc, Bison, Lemon (generates C, wrap to Python)
  - Sphinx (C++, wrap to Python)
  - various Python parser generators

What is LLVM?

- Primarily a set of libraries..

What is LLVM?

- Primarily a set of libraries..
- ..with which you can make a compiler backend..

What is LLVM?

- Primarily a set of libraries..
- ..with which you can make a compiler backend..
- ..or a VM with JIT support (but hard to get this right)

What is LLVM?

- Primarily a set of libraries..
- ..with which you can make a compiler backend..
- ..or a VM with JIT support (but hard to get this right)
- It provides:

What is LLVM?

- Primarily a set of libraries..
- ..with which you can make a compiler backend..
- ..or a VM with JIT support (but hard to get this right)
- It provides:
  - IR data structure, with text and binary representations

What is LLVM?

- Primarily a set of libraries..
- ..with which you can make a compiler backend..
- ..or a VM with JIT support (but hard to get this right)
- It provides:
  - IR data structure, with text and binary representations
  - wide range of optimizations

What is LLVM?

- Primarily a set of libraries..
- ..with which you can make a compiler backend..
- ..or a VM with JIT support (but hard to get this right)
- It provides:
  - IR data structure, with text and binary representations
  - wide range of optimizations
  - code generator (partly description-based) for many platforms

What is LLVM?

- Primarily a set of libraries..
- ..with which you can make a compiler backend..
- ..or a VM with JIT support (but hard to get this right)
- It provides:
  - IR data structure, with text and binary representations
  - wide range of optimizations
  - code generator (partly description-based) for many platforms
  - JIT-compile-execute from IR

What is LLVM?

- Primarily a set of libraries..
- ..with which you can make a compiler backend..
- ..or a VM with JIT support (but hard to get this right)
- It provides:
  - IR data structure, with text and binary representations
  - wide range of optimizations
  - code generator (partly description-based) for many platforms
  - JIT-compile-execute from IR
  - link-time optimization (LTO)

What is LLVM?

- Primarily a set of libraries..
- ..with which you can make a compiler backend..
- ..or a VM with JIT support (but hard to get this right)
- It provides:
  - IR data structure, with text and binary representations
  - wide range of optimizations
  - code generator (partly description-based) for many platforms
  - JIT-compile-execute from IR
  - link-time optimization (LTO)
  - support for accurate garbage collection

LLVM Highlights

- Written in readable C++

LLVM Highlights

- Written in readable C++
- APIs and command-line tools

LLVM Highlights

- Written in readable C++
- APIs and command-line tools
- Reasonable documentation, helpful and mature community

LLVM Highlights

- Written in readable C++
- APIs and command-line tools
- Reasonable documentation, helpful and mature community
- clang is a now-famous subproject of LLVM

LLVM Highlights

- Written in readable C++
- APIs and command-line tools
- Reasonable documentation, helpful and mature community
- clang is a now-famous subproject of LLVM
- llvm-gcc is gcc frontend (C, C++, Java, Ada, Fortran, ObjC) + LLVM backend

LLVM Highlights

- Written in readable C++
- APIs and command-line tools
- Reasonable documentation, helpful and mature community
- clang is a now-famous subproject of LLVM
- llvm-gcc is gcc frontend (C, C++, Java, Ada, Fortran, ObjC) + LLVM backend
- LLVM users: Unladen Swallow, Iced Tea, Rubinus, llvm-lua, GHC, LDC

The LLVM IR

- The starting point of LLVM's work is the IR

The LLVM IR

- The starting point of LLVM's work is the IR
- Frontends construct an in-memory IR

The LLVM IR

- The starting point of LLVM's work is the IR
- Frontends construct an in-memory IR
- Equivalent textual respresentation called LLVM assembly

The LLVM IR

- The starting point of LLVM's work is the IR
- Frontends construct an in-memory IR
- Equivalent textual respresentation called LLVM assembly
- Equivalent binary respresentation called bitcode

The LLVM IR

- The starting point of LLVM's work is the IR
- Frontends construct an in-memory IR
- Equivalent textual respresentation called LLVM assembly
- Equivalent binary respresentation called bitcode

The LLVM IR

- The starting point of LLVM's work is the IR
- Frontends construct an in-memory IR
- Equivalent textual respresentation called LLVM assembly
- Equivalent binary respresentation called bitcode

"Hello world" in LLVM Assembly

```
@msg = private constant [15 x i8] c"Hello, world!\0A\00"

declare i32 @puts(i8*)

define i32 @main(i32 %argc, i8** %argv) {
entry:
  %0 = getelementptr [15 x i8]* @msg, i64 0, i64 0
  %1 = tail call i32 @puts(i8* %0)
  ret i32 undef
}
```

A Word About clang

- clang is a C, Objective C and C++ frontend

A Word About clang

- clang is a C, Objective C and C++ frontend
- it converts C/ObjC/C++ code to LLVM IR

A Word About clang

- clang is a C, Objective C and C++ frontend
- it converts C/ObjC/C++ code to LLVM IR
- which can be played around with using llvm-py

A Word About clang

- clang is a C, Objective C and C++ frontend
- it converts C/ObjC/C++ code to LLVM IR
- which can be played around with using llvm-py
- and then compiled "normally"

What is llvm-py?

- Python bindings for LLVM

What is llvm-py?

- Python bindings for LLVM
- Including JIT

What is llvm-py?

- Python bindings for LLVM
- Including JIT
- Excellent for experimenting and prototyping

What is llvm-py?

- Python bindings for LLVM
- Including JIT
- Excellent for experimenting and prototyping
- Available in Ubuntu, Debian, MacPorts (but may be out of date)

What is llvm-py?

- Python bindings for LLVM
- Including JIT
- Excellent for experimenting and prototyping
- Available in Ubuntu, Debian, MacPorts (but may be out of date)
- Current version (0.6) works with LLVM 2.7

llvm-py Internals

- Wraps LLVM's C API as a Python extension module

llvm-py Internals

- Wraps LLVM's C API as a Python extension module
- Extends LLVM C API as it is incomplete

llvm-py Internals

- Wraps LLVM's C API as a Python extension module
- Extends LLVM C API as it is incomplete
- Does not use binding generators (Boost.Python, swig etc)

llvm-py Internals

- Wraps LLVM's C API as a Python extension module
- Extends LLVM C API as it is incomplete
- Does not use binding generators (Boost.Python, swig etc)
- Public APIs are in Python, use extension module internally

llvm-py Internals

- Wraps LLVM's C API as a Python extension module
- Extends LLVM C API as it is incomplete
- Does not use binding generators (Boost.Python, swig etc)
- Public APIs are in Python, use extension module internally
- No dependencies other than Python, LLVM

Thanks!

Thanks for listening!

- LLVM: `http://www.llvm.org/`
- llvm-py: `http://www.mdevan.org/llvm-py/`
- Must read: *Steven S. Muchnick*. Advanced Compiler Design & Implementation.
- Me: Mahadevan R, mdevan@mdevan.org, `http://www.mdevan.org/`